Nonparametric/Blackbox Regression and Classification 1

See Syllabus for reference reading

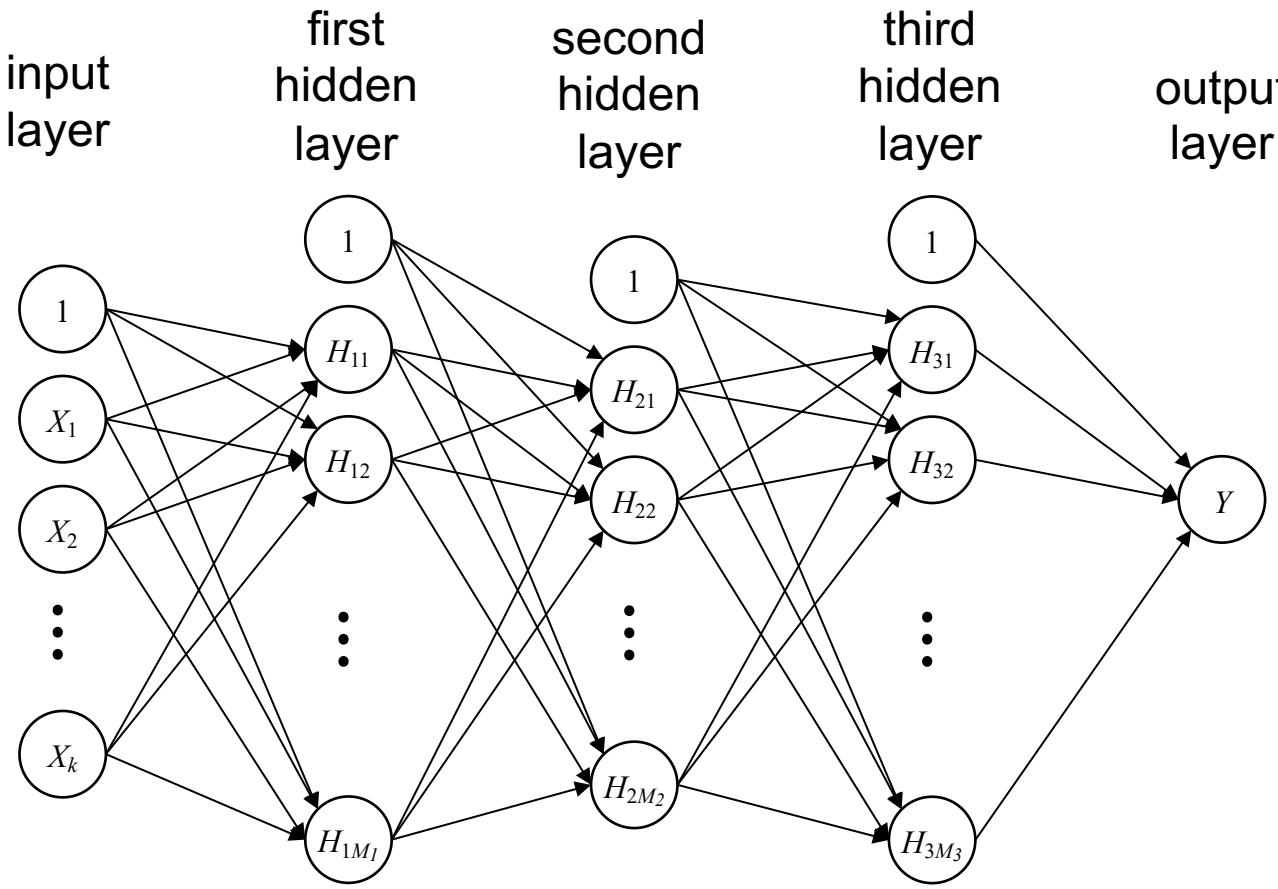
Specified Structure Versus Blackbox Approach

- If you have knowledge of the structure of the relationship between Y and $\mathbf{x}$, then the best approach is to use it (e.g., if you think it is an linear, exponential, quadratic, etc. relationship, then fit that model)
- For most data sets (and especially large "data mining" applications), we might doubt a linear model will fit but have no idea of the structure of the nonlinearities.
  - In this case, unless there are only a few predictors, polynomial (e.g., quadratic) models are not the preferred next step to try beyond linear models
  - Why not?
- There are way too many blackbox nonlinear modeling approaches to cover in one course, but we will cover some of the more popular ones that span the spectrum of methods
  - Almost all can be used equally well for either regression or classification

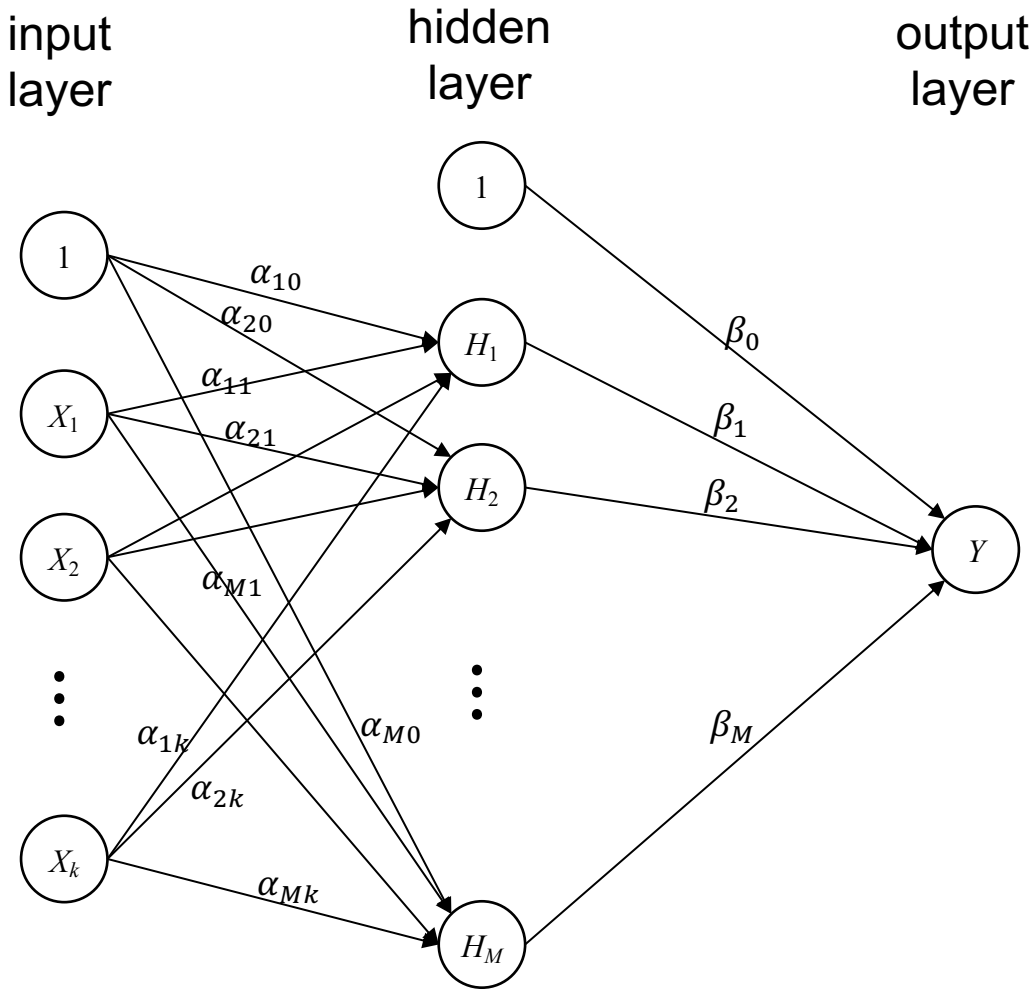
Neural Networks

- 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 blackbox 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 (Y) and the other columns are any number of predictor variables (X's)
  - The neural network hidden variables (H's) are internal variables that you do not enter or even care about

Standard Graphical Depiction of a Neural Network with Multiple (3 in this case) Hidden Layers



Standard Graphical Depiction of a Neural Network with a Single Hidden Layer



Mathematical Definition of What a Neural Network Model Really Is

- each "node" represents an activation function (labeled as the function output, with function input a linear combo of outputs from previous layer)
- X's: input (i.e., predictor) variables, in "input layer"
- Y: output (i.e. response) variable, in "output layer"
- H's: internal dummy variables, in "hidden layer"
- α's and β's: model parameters, to be estimated
- the NN model:

for $m = 1, 2, \ldots, M$,

$$H_m = \frac{\exp\left\{\alpha_{m,0} + \alpha_{m,1} X_1 + \cdots + \alpha_{m,k} X_k\right\}}{1 + \exp\left\{\alpha_{m,0} + \alpha_{m,1} X_1 + \cdots + \alpha_{m,k} X_k\right\}}$$

$$Y = \frac{exp\left\{\beta_0 + \beta_1 H_1 + \cdots + \beta_M H_M\right\}}{1 + exp\left\{\beta_0 + \beta_1 H_1 + \cdots + \beta_M H_M\right\}} + \varepsilon$$

Neural Network Activation Functions

- For classification, it is common to use the same sigmoidal (logistic) activation function for each node:

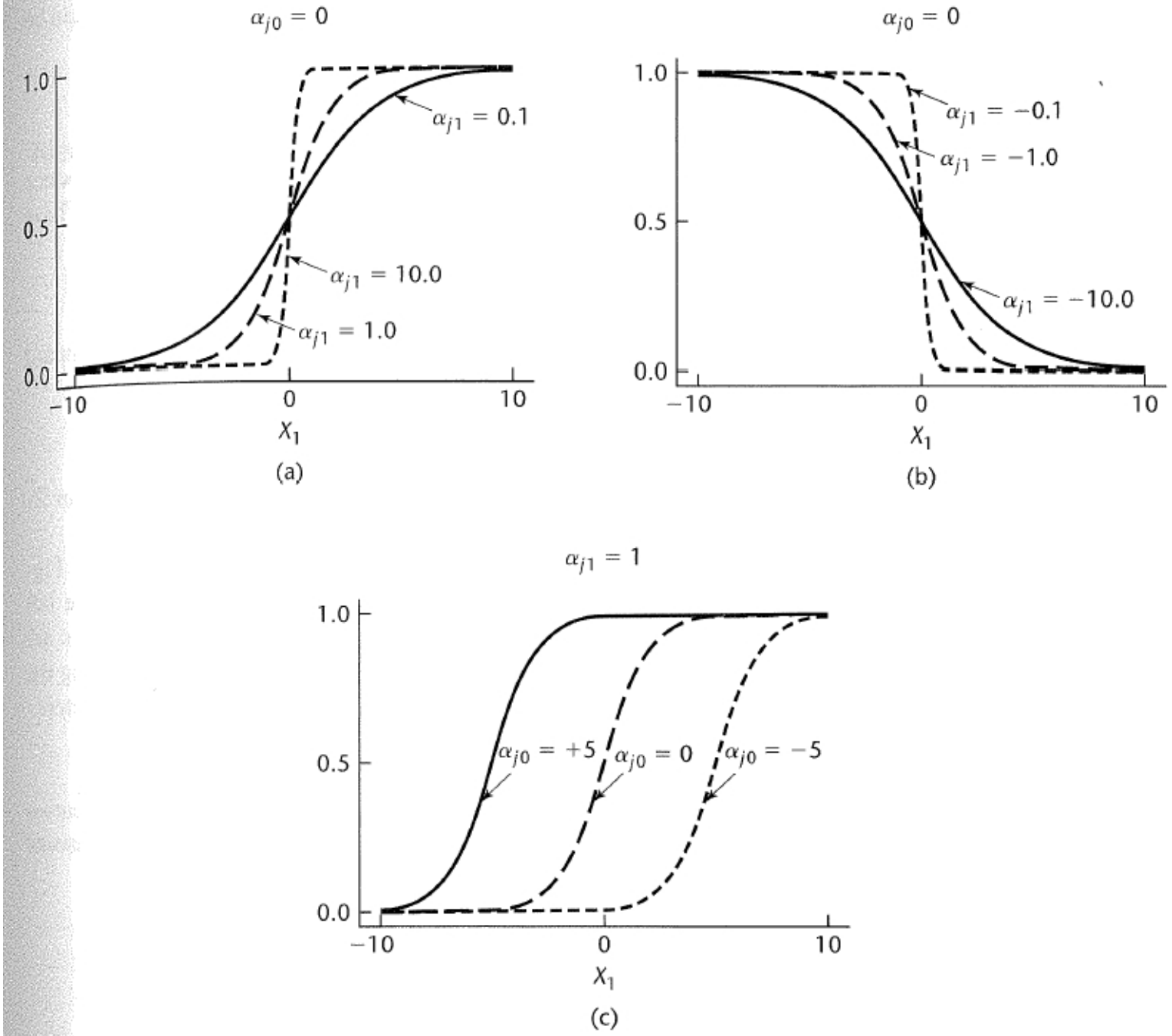
$$h(z) = \frac{exp\{z\}}{1 + exp\{z\}} = \frac{1}{1 + exp\{-z\}}$$

where z = linear combo of outputs from previous layer

- For regression, it is usually preferable to use sigmoidal activation functions for all hidden nodes and a linear activation [i.e., $h(z) = z$] function for the output layer nodes:

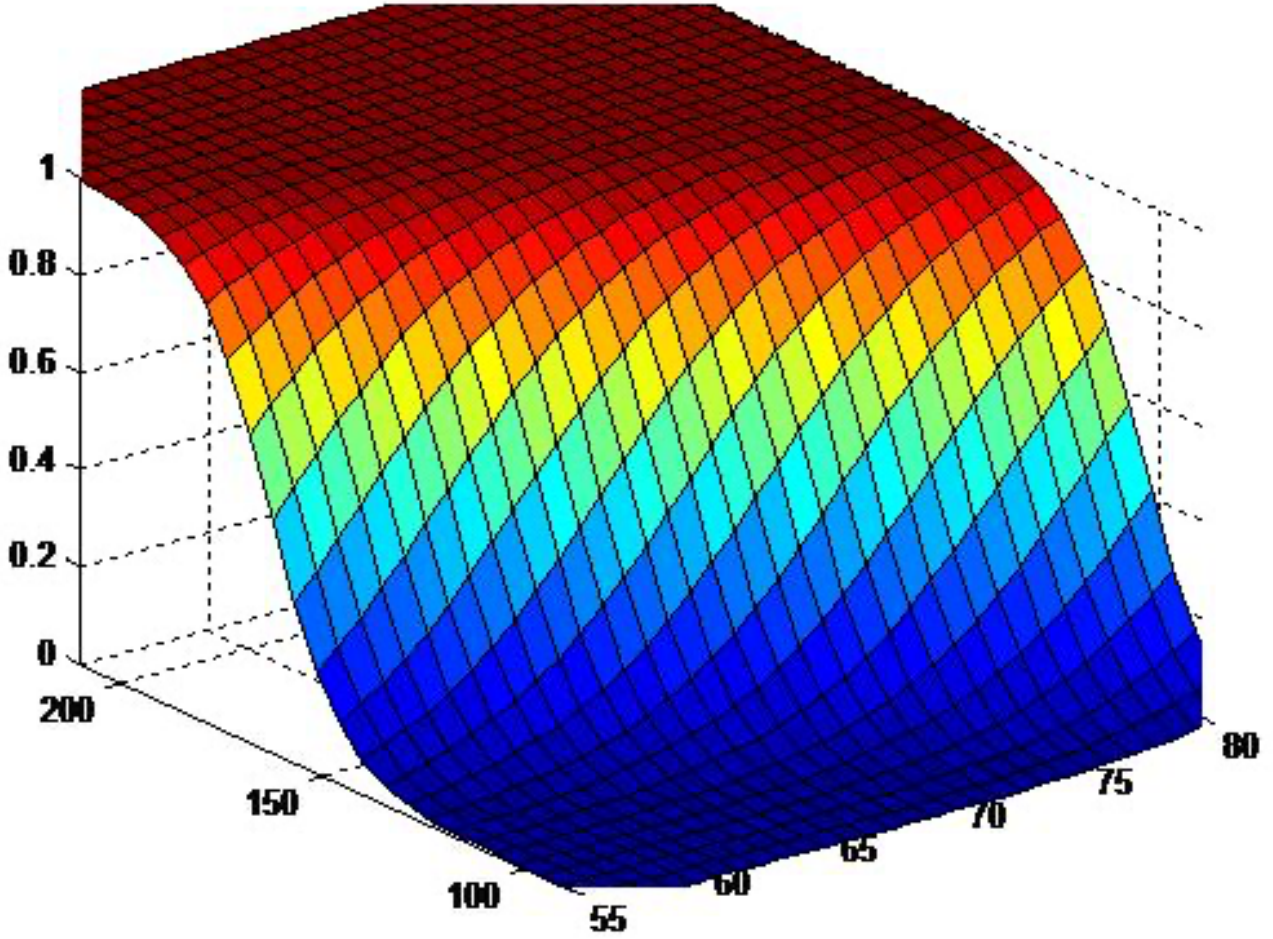
$$Y = \beta_0 + \beta_1 H_1 + \cdots + \beta_M H_M + \varepsilon$$

FIGURE 13.7 **Various Logistic Activation Functions for Single Predictor.**



An S-shaped function with multivariate input

- Recall that this is what the S-shaped logistic function looks like when there are multiple input variables



Discussion Points and Questions

- Y is an S-shaped (or sometimes linear) function of the dummy variables (the H's), which are in turn S-shaped functions of the predictors (the X's)
- When you combine them together, substituting for the H's to get Y as a function of the X's, you can think of the neural network model as

$$Y = g(\mathbf{x},\boldsymbol{\theta}) + \varepsilon$$

for some (very messy) $g(\mathbf{x},\boldsymbol{\theta})$ with $\boldsymbol{\theta} = \{\text{all } \alpha\text{'s and } \beta\text{'s}\}$

- What kind of functional $\mathbf{x} \rightarrow Y$ relationships can you capture with the neural network model structure?

Fitting A Neural Network Model

1) Standardize predictors via

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

$\bar{x}_j, s_{x_j}$ = average, stdev of jth predictor (jth column)

2) Also standardize the response. Or instead, if using logistic output activation function, scale response to interval [0,1] via

$$y_i \rightarrow \frac{y_i - y_{min}}{y_{max} - y_{min}}$$

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

Fitting A Neural Network Model, continued

3) Choose:
   - # hidden layers
   - # nodes in each hidden layer
   - output activation function (usually linear or logistic)
   - other options and tuning parameters (e.g. λ)

4) Software estimates parameters to minimize (nonlinear LS with shrinkage):

$$\sum_{i=1}^{n}\left[y_i - g(\mathbf{x}_i,\boldsymbol{\theta})\right]^2 + \lambda\left(\sum_{m=1}^{M}\sum_{j=0}^{k}\alpha_{m,j}^2 + \sum_{m=0}^{M}\beta_m^2\right)$$

$g(\mathbf{x}_i,\boldsymbol{\theta})$ denotes the neural network response prediction

where

$\boldsymbol{\theta}$ = {all α's and β's}

$$g(\mathbf{x}_i, \boldsymbol{\theta}) = \frac{exp\{\beta_0 + \beta_1 H_{i,1} + \cdots + \beta_M H_{i,M}\}}{1 + exp\{\beta_0 + \beta_1 H_{i,1} + \cdots + \beta_M H_{i,M}\}}$$

$$H_{i,m} = \frac{exp\{\alpha_{m,0} + \alpha_{m,1} x_{i,1} + \cdots + \alpha_{m,k} x_{i,k}\}}{1 + exp\{\alpha_{m,0} + \alpha_{m,1} x_{i,1} + \cdots + \alpha_{m,k} x_{i,k}\}}$$

λ = user-chosen shrinkage parameter

- Why do we need to include the shrinkage term?

Example: Predictive Modeling of Concrete Strength

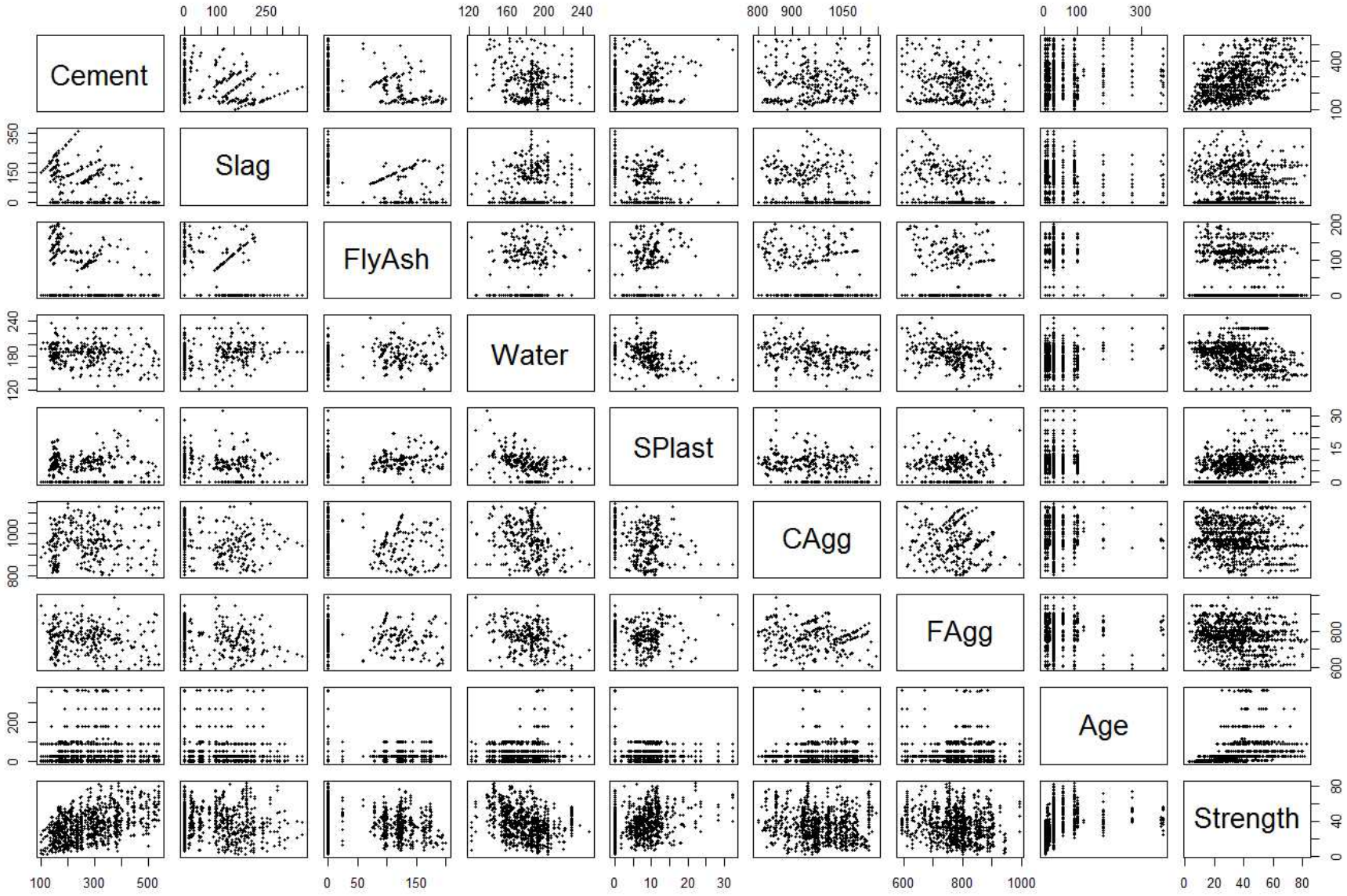
- Data in concrete.csv (from UCI Machine Learning Repository)
- 1030 cases with 8 predictor variables. Concrete is the most important material in civil engineering. The concrete compressive strength (the response) is a highly nonlinear function of age (days) and ingredients (kg/m^3 concentration). These ingredients include cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate.
- The objective is to learn the learn the complex effects of the age and ingredients on concrete strength to (i) understand the most important factors and their effects and (ii) optimize the concrete recipe to maximize strength, etc

- Example with a bigger data set coming up shortly

Neural Network Modeling of Concrete Data

```
########R code for reading in the concrete data, converting the response to [0,1] interval,
      and standardizing predictors################
CRT <- read.csv("concrete.csv",header=TRUE)
k<-ncol(CRT)-1  #number of predictors
CRT1 <- CRT  #will be standardized and scaled version of data
CRT1[1:k]<-sapply(CRT1[1:k], function(x) (x-mean(x))/sd(x)) #standardize predictors
CRT1[k+1]<-(CRT1[k+1]-min(CRT1[k+1]))/(max(CRT1[k+1])-min(CRT1[k+1]))
CRT[1:10,]
pairs(CRT, cex=.5, pch=16)
```

Matrix scatterplot of concrete data



Concrete Example Continued

```
###############Fit a neural network model to the CRT1 data######################
library(nnet)
nn1<-nnet(Strength~.,CRT1, linout=T, skip=F, size=10, decay=0.01, maxit=1000,
      trace=F)
yhat<-as.numeric(predict(nn1))
y<-CRT1[[9]]; e<-y-yhat
plot(yhat,y)
c(sd(y),sd(e))
 #repeat but using logistic output function, for which the response MUST BE SCALED TO
      [0,1] RANGE
nn1<-nnet(Strength~.,CRT1, linout=F, skip=F, size=10, decay=0.01, maxit=1000,
      trace=F)
yhat<-as.numeric(predict(nn1))
y<-CRT1[[9]]; e<-y-yhat
plot(yhat,y)
c(sd(y),sd(e))
##
summary(nn1)
```

```
> summary(nn1)  #this is for a fit with size=10
a 8-10-1 network with 101 weights
options were - decay=0.01
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1
 -1.15   0.18  -1.10   1.23  -0.34   0.52  -0.65  -0.60   0.23
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2
  0.44  -0.86  -0.63  -0.69   0.66  -0.32   0.56   0.31   0.42
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3
 -0.76   1.13  -0.01   0.82  -1.60  -0.37   0.02   0.76   0.20
 b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4 i7->h4 i8->h4
  0.52   0.31  -0.67  -1.11  -0.78  -0.27   1.08   0.83  -0.20
 b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5 i7->h5 i8->h5
 -0.52   0.08   0.74   0.12  -0.67   0.99  -1.48  -1.51  -0.24
 b->h6 i1->h6 i2->h6 i3->h6 i4->h6 i5->h6 i6->h6 i7->h6 i8->h6
  0.60   0.04  -0.92   0.38   1.19   0.28  -0.03  -0.87   0.22
 b->h7 i1->h7 i2->h7 i3->h7 i4->h7 i5->h7 i6->h7 i7->h7 i8->h7
 -1.93  -1.79  -0.82  -0.36  -0.50  -0.20  -0.24  -0.46  -0.01
 b->h8 i1->h8 i2->h8 i3->h8 i4->h8 i5->h8 i6->h8 i7->h8 i8->h8
 -1.86  -0.70  -0.82  -0.68  -0.93   1.11  -0.69   0.46  -0.75
 b->h9 i1->h9 i2->h9 i3->h9 i4->h9 i5->h9 i6->h9 i7->h9 i8->h9
  3.13  -0.32  -0.62  -0.33   0.02   0.21  -0.26  -0.29   4.19
 b->h10 i1->h10 i2->h10 i3->h10 i4->h10 i5->h10 i6->h10 i7->h10 i8->h10
   0.50   -0.02   -0.13   -0.04    0.53    1.38    0.01   -1.05    0.13
  b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  h8->o  h9->o h10->o
 -0.28  -1.37  -2.06   1.90  -1.34  -1.98  -1.66  -2.07  -1.45   3.71   1.95
```

Discussion Points and Questions

- Why do we need to standardize the predictors (and the response variable when using a linear output activation function)?
- How can we get r^2 for this example (the nnet function in R does not spit it out)
- Which predictor variables appear to be the most important, and what R output do we look at to determine this?
- What values of λ and size will give us the smallest training SSE?
- How can we decide the best values of λ and size?

Concrete Example Continued

```
########A function to determine the indices in a CV partition####################
CVInd <- function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices for each part
   m<-floor(n/K)  #approximate size of each part
   r<-n-m*K
   I<-sample(n,n)  #random reordering of the indices
   Ind<-list()  #will be list of indices for all K parts
   length(Ind)<-K
   for (k in 1:K) {
      if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
         else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
      Ind[[k]] <- I[kpart]  #indices for kth part of data
   }
   Ind
}
```

Concrete Example Continued

```
##Now use multiple reps of CV to compare Neural Nets and linear reg models###
Nrep<-3 #number of replicates of CV
K<-3  #K-fold CV on each replicate
n.models = 3 #number of different models to fit
n=nrow(CRT1)
y<-CRT1$Strength
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
     out<-nnet(Strength~.,CRT1[-Ind[[k]],], linout=F, skip=F, size=15, decay=.01, maxit=1000, trace=F)
     yhat[Ind[[k]],1]<-as.numeric(predict(out,CRT1[Ind[[k]],]))
     out<-nnet(Strength~.,CRT1[-Ind[[k]],], linout=F, skip=F, size=30, decay=0, maxit=1000, trace=F)
     yhat[Ind[[k]],2]<-as.numeric(predict(out,CRT1[Ind[[k]],]))
     out<-lm(Strength~.,CRT1[-Ind[[k]],])
     yhat[Ind[[k]],3]<-as.numeric(predict(out,CRT1[Ind[[k]],]))
  } #end of k loop
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
MSE
MSEAve<- apply(MSE,2,mean); MSEAve #averaged mean square CV error
MSEsd <- apply(MSE,2,sd); MSEsd   #SD of mean square CV error
r2<-1-MSEAve/var(y); r2  #CV r^2
```

Discussion Points and Questions

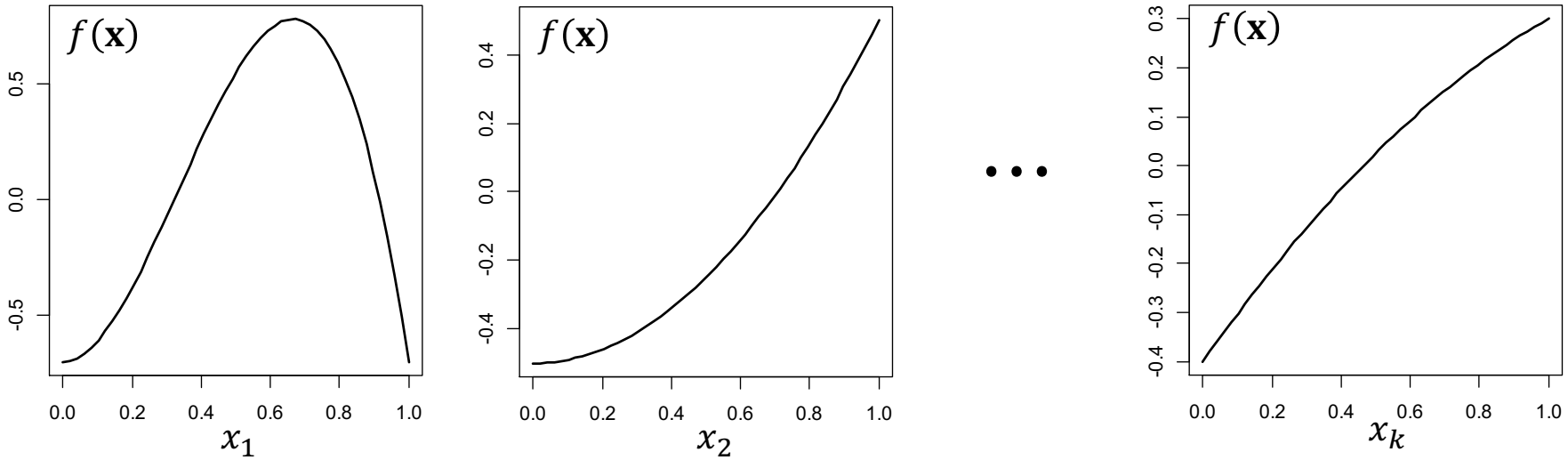
- The best value of λ is the value that results in the smallest SSE_{CV} (or equivalently, the largest CV r^2, smallest CV SD(e), etc).
- How can we decide the best number of hidden layer nodes?
- Why should we use the same CV partition when comparing two models?
- What are the pros and cons of n-fold CV versus K-fold CV for other K, e.g., 3, 5, or 10?

Understanding the Effects of the Predictors in Complex Supervised Learning Models

Goal: For generic black-box fitted supervised learning model $\hat{y} = f(\mathbf{x})$, plot some suitably "averaged" version of $f(\boldsymbol{X})$ vs. each of $X_1, X_2, \ldots, X_k$ (and vs. pairs of predictors for 2nd-order interaction effects) to help visualize and assess:

- Which predictors have the biggest effect on the response, and
- What is the nature of their effect on $\hat{y}$ (positive, negative, monotonic, nonlinear, interactions, etc)

e.g., we want to produce "main effects" plots like these:



- We obviously care about understanding the effects of the predictors if the purpose of the analysis is explanatory. Why do we care if the purpose is purely predictive?
  - Transparency is important for many reasons
  - If the effect of a predictor violates intuition, then either your model is unreliable or you have discovered something interesting
  - You may want to go beyond pure prediction and use the model for decision making and active intervention
  - Regulatory environments . . .
- Some approaches for visualizing the main and interaction effects of the predictors are:
  - Partial dependence (PD) plots (Friedman, 2001 – original gbm paper)
  - Accumulated local effects (ALE) plots (Apley & Zhu, 2019)
  - Marginal (M) plots (do NOT use these)

Definition of PD, M, and ALE Plots

The main effect function $f_1(x_1)$ of a predictor x_1 is calculated and plotted for each value of x_1 in some discrete set that covers the range of x_1. Likewise for $x_2, \ldots, x_k$.

PD plot x_1-effect and its estimator

$$f_{1,PD}(x_1) = E[f(x_1, X_2, \ldots, X_k)]$$

$$\hat{f}_{1,PD}(x_1) = \frac{1}{n}\sum_{i=1}^{n} f\left(x_1, x_{i,2}, \ldots, x_{i,k}\right)$$

M plot x_1-effect and its estimator

$$f_{1,M}(x_1) = E[f(x_1, X_2, \ldots, X_k)|X_1 = x_1]$$

$$\hat{f}_{1,M}(x_1) = \frac{1}{n(x_1)}\sum_{x_{i,1}\in N(x_1)} f\left(x_1, x_{i,2}, \ldots, x_{i,k}\right)$$

$N(x_1)$ is some neighborhood of x_1

$n(x_1)$ is the # of training $x_{i,1}$ in $N(x_1)$

ALE plot x_1-effect and its estimator

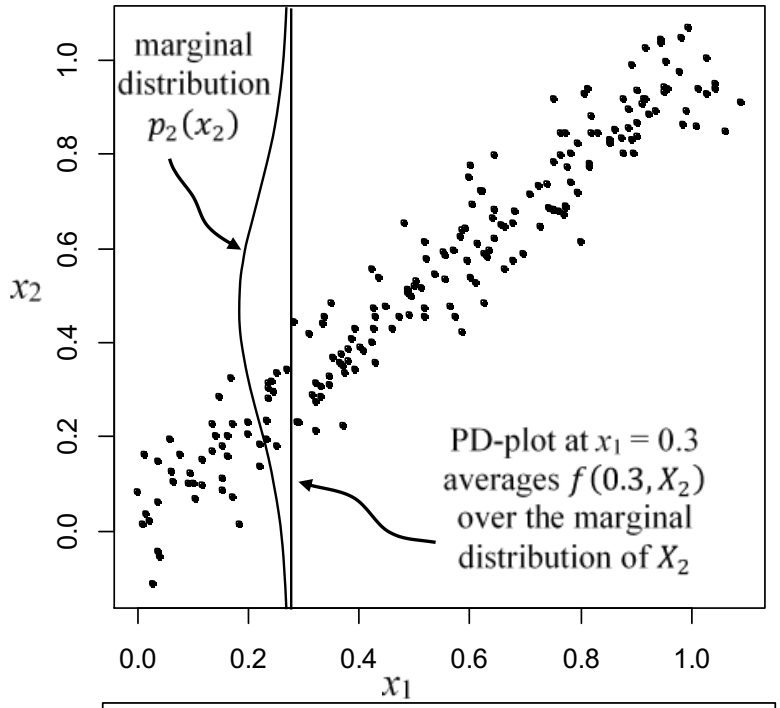
$$f_{1,ALE}(x_1) = \int_{x_{1,min}}^{x_1} E\left[\frac{\partial f(X_1, X_2, \ldots, X_k)}{\partial X_1}|X_1 = \tilde{x}_1\right] d\tilde{x}_1$$

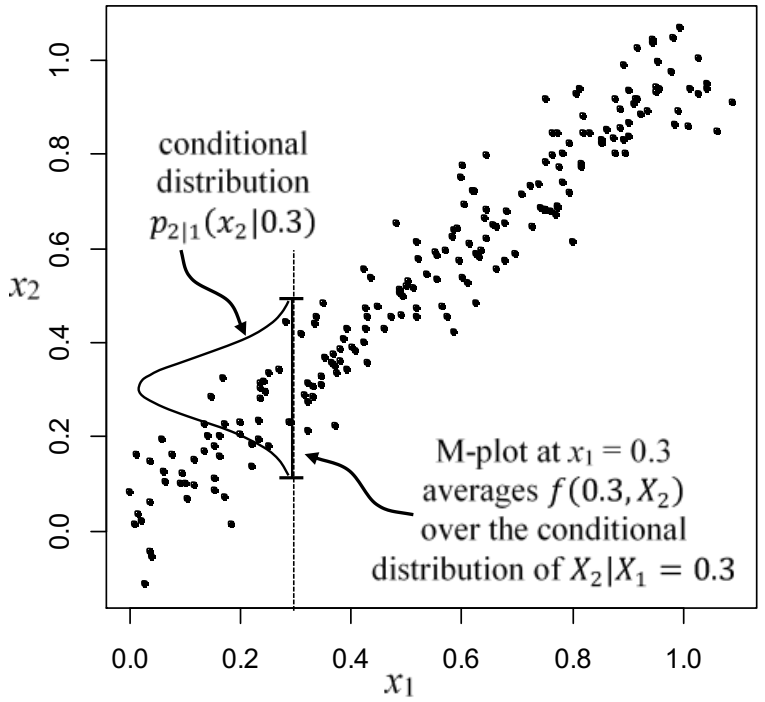
$\hat{f}_{1,ALE}(x_1) =$ same but substitute:

i) finite difference for derivative

ii) sample average for expectation

iii) summation for integral





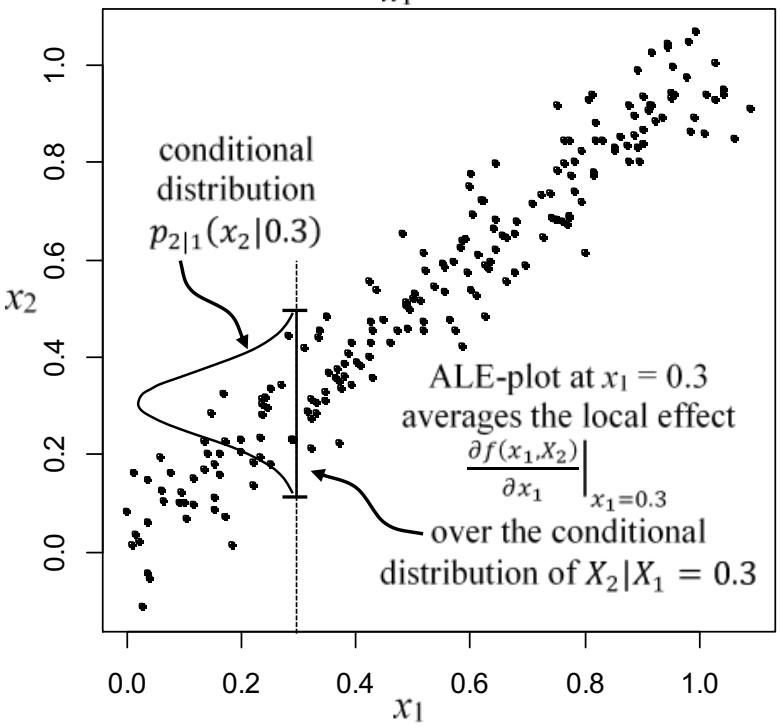
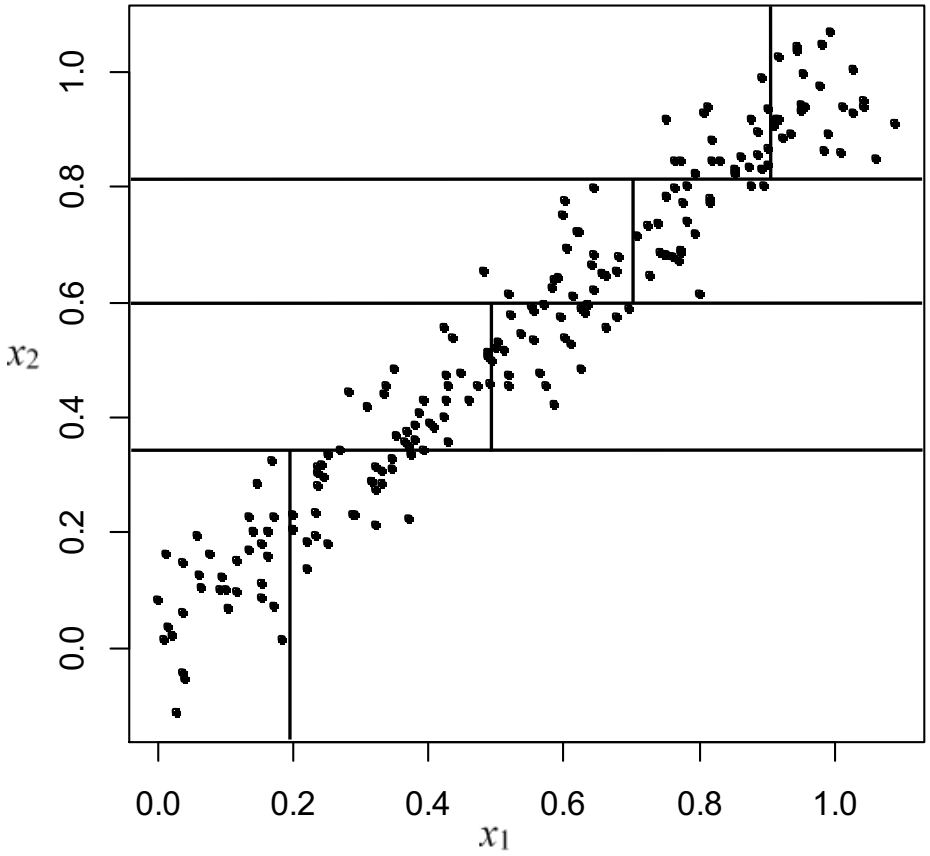
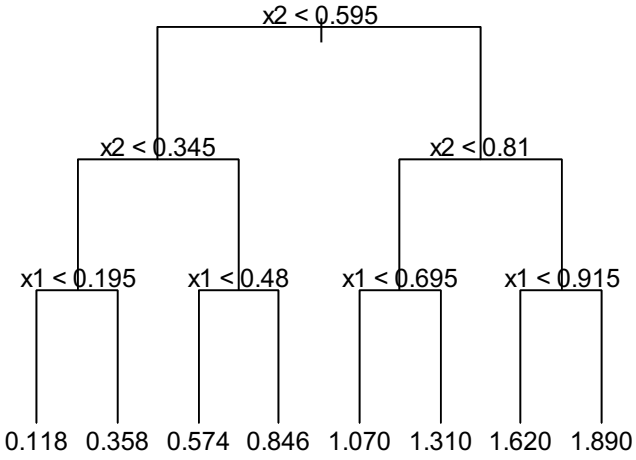


Illustration of PD, M, and ALE plot definitions for $k = 2$ predictors

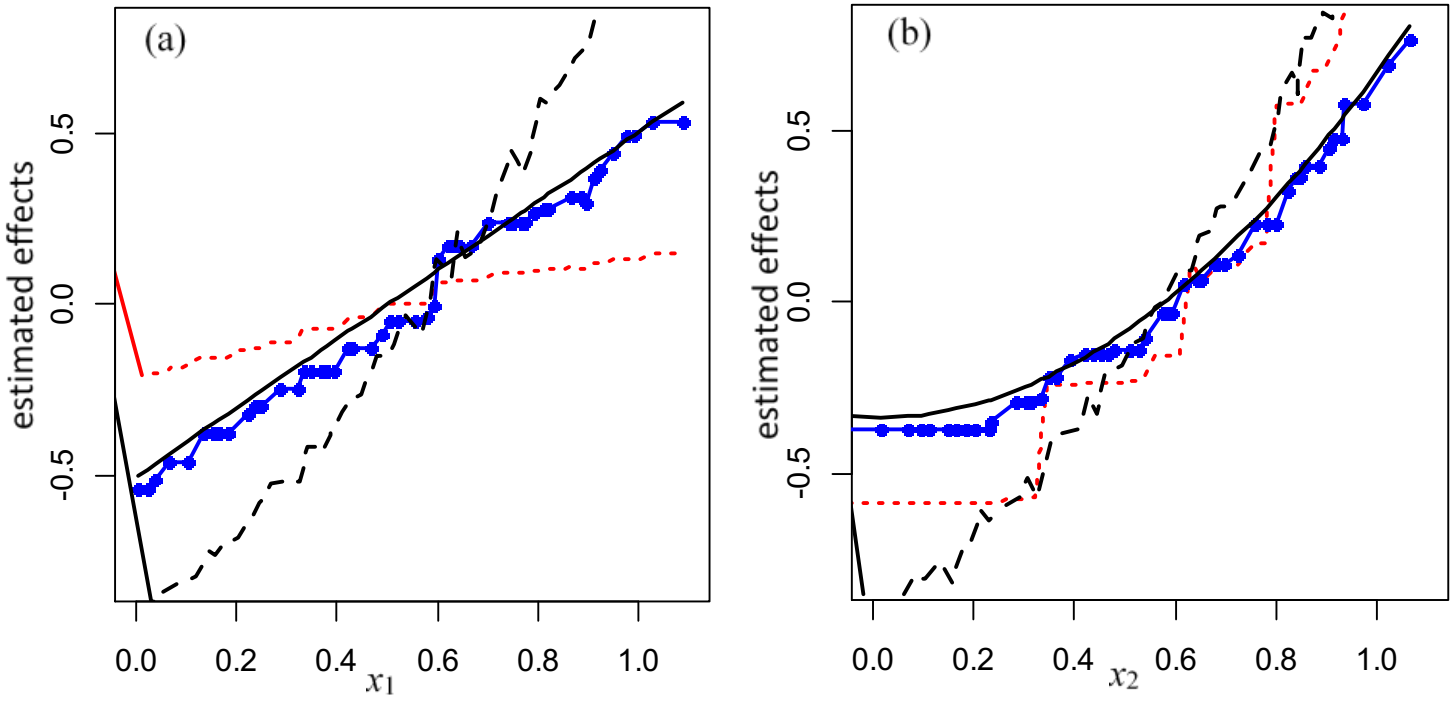
Example Illustrating Problems with PD and M Plots for Highly Correlated Predictors

The data are $n = 200$ observations of $Y = X_1 + X_2^2$ (no observation error) with correlated (X_1, X_2) distributed as shown below. A 100-node tree $f(x_1, x_2)$ was fit to the data, the first eight splits of which are shown below



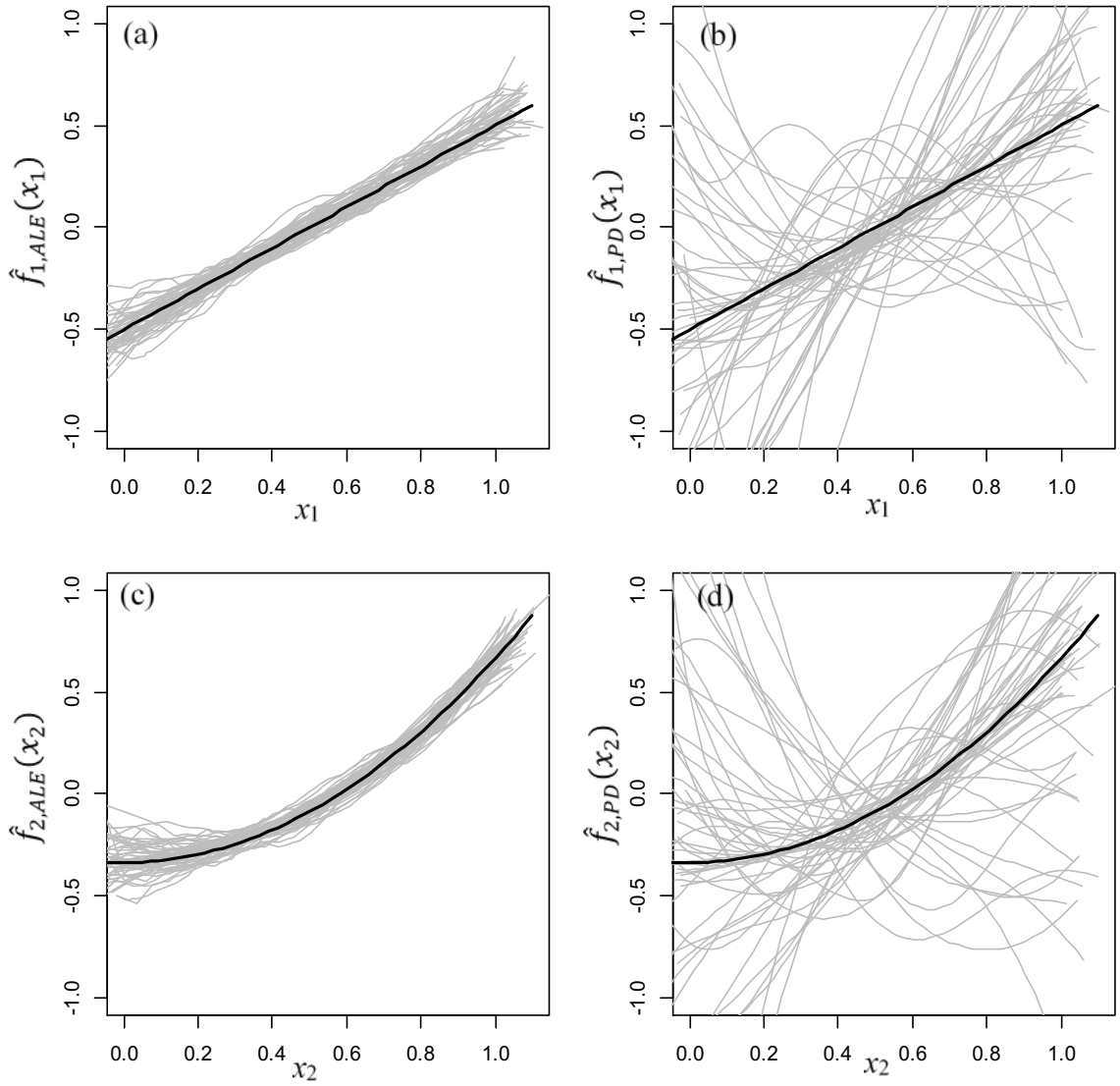


Example Continued: PD, M, and ALE Plot Results



- $\hat{f}_{j,ALE}(x_j)$ (blue line with bullets), $\hat{f}_{j,PD}(x_j)$ (red dotted line), $\hat{f}_{j,M}(x_j)$ (black dashed line), and the true main effect of x_j (black solid line) for **(a)** $j = 1$ and a true linear effect for x_1, and **(b)** $j = 2$ and a true quadratic effect for x_2.
- $\hat{f}_{j,ALE}(x_j)$ is much more accurate than either $\hat{f}_{j,PD}(x_j)$ or $\hat{f}_{j,M}(x_j)$

Same Example, but for 50 Replicates and $f(x_1, x_2)$ a Fitted Neural Network Model

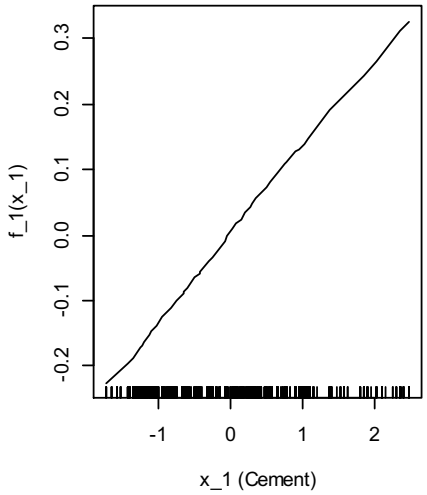


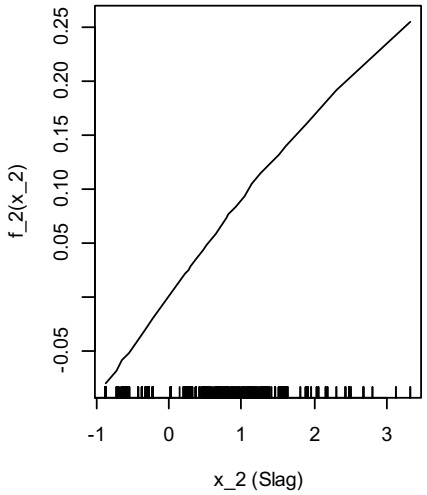
- (X_1, X_2) follow the same distribution
- The response is now $Y = X_1 + X_2^2 + \varepsilon$ with $\varepsilon \sim N(0, 0.1^2)$
- The neural networks fit very well, with cross-validation r^2 between 0.965 and 0.975 for all replicates
- PD plots (right panels) are terrible, whereas ALE plots (left panels) are quite accurate

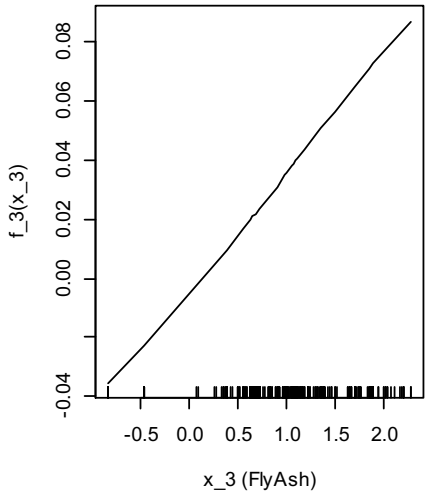
Concrete Example Continued: ALE Plots

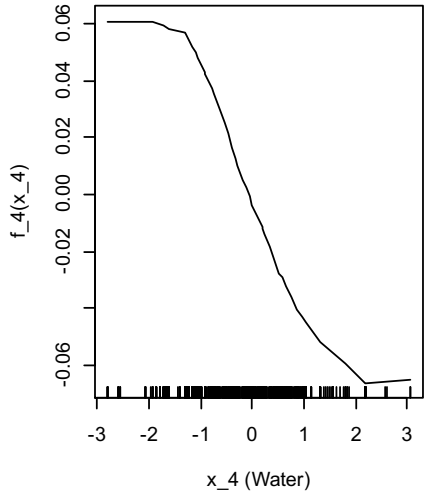
```
###############Visualizing the effects of the predictors#####################
nn1<-nnet(Strength~.,CRT1, linout=F, skip=F, size=15, decay=0.1, maxit=1000, trace=F)
      ##From CV, these are about the best tuning parameters
summary(nn1)

## Use ALEPlot package to create accumulated local effects (ALE) plots
library(ALEPlot)
yhat <- function(X.model, newdata) as.numeric(predict(X.model, newdata))
par(mfrow=c(2,4))
for (j in 1:8)  {ALEPlot(CRT1[,1:8], nn1, pred.fun=yhat, J=j, K=50, NA.plot = TRUE)
  rug(CRT1[,j]) }  ## This creates main effect ALE plots for all 8 predictors
par(mfrow=c(1,1))

par(mfrow=c(2,2))  ## This creates 2nd-order interaction ALE plots for x1, x2, x8
ALEPlot(CRT1[,1:8], nn1, pred.fun=yhat, J=c(1,2), K=50, NA.plot = TRUE)
ALEPlot(CRT1[,1:8], nn1, pred.fun=yhat, J=c(1,8), K=50, NA.plot = TRUE)
ALEPlot(CRT1[,1:8], nn1, pred.fun=yhat, J=c(2,8), K=50, NA.plot = TRUE)
par(mfrow=c(1,1))
```

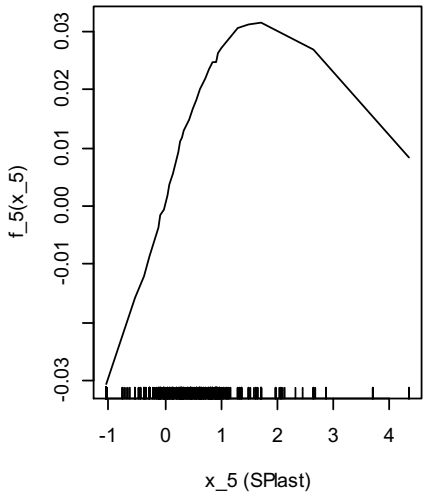
Main Effect ALE plots

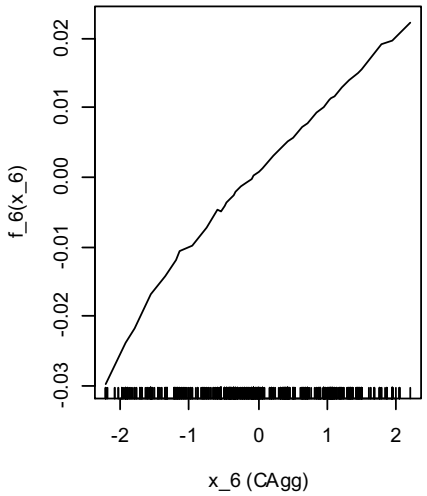


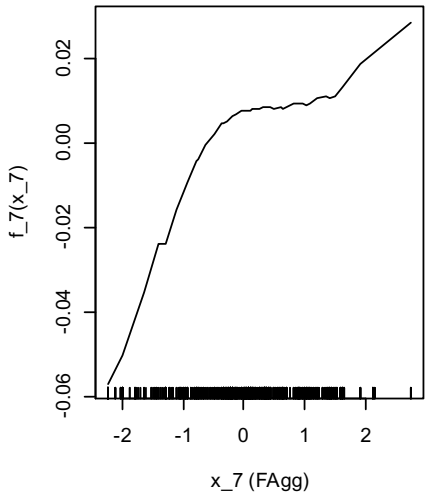


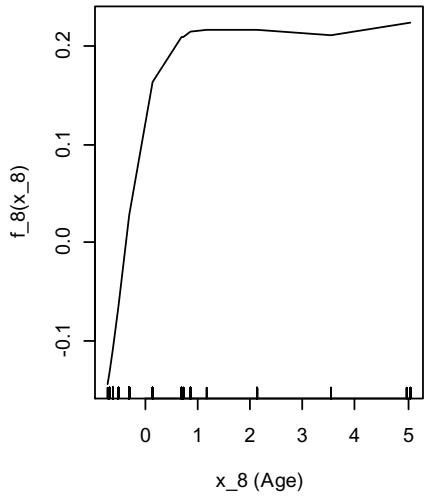




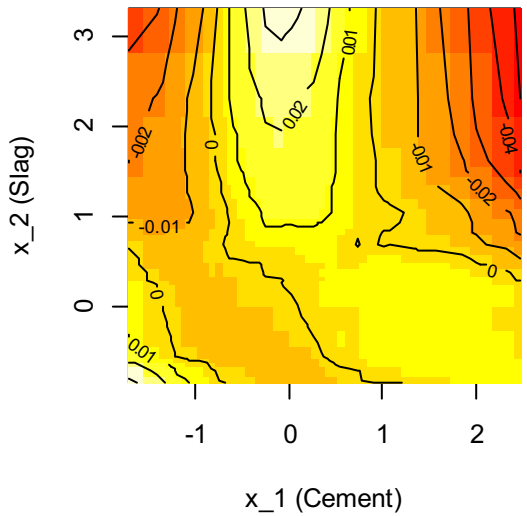


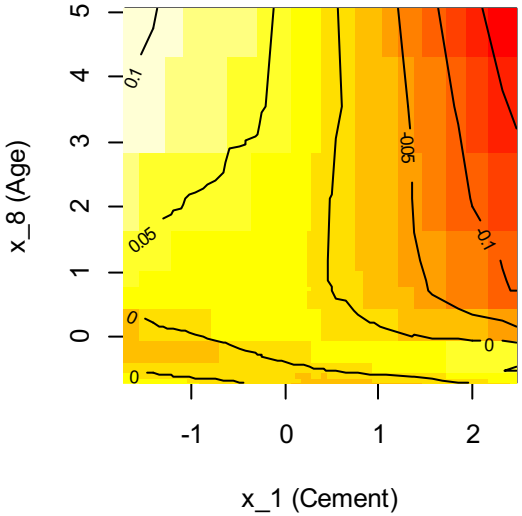


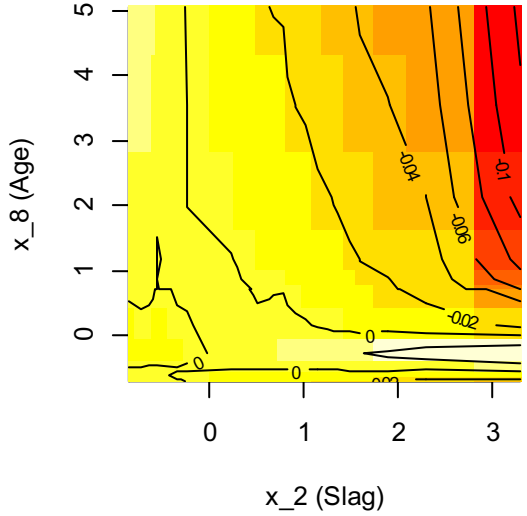




Some 2nd-Order Interaction ALE plots







Discussion Points and Questions

- How important each predictor is can be assessed based on how much its ALE plot main effect function and second-order interaction effect functions vary (pay attention to the vertical axes scaling, and use rug plots to see the distribution of predictor values)
- Based on the main effects, which predictors are the most important? What are their effects on concrete strength?
- How do you interpret the interaction plots?
  - What is the effect of cement when slag is low vs. when slag is high? (the overall effect is the main effect plus the interaction)
  - What is the effect of cement when age is low vs. when age is high?
  - What is the effect of slag when age is low vs. when age is high?
- Make sure you understand the difference between correlation between two predictors vs. an interaction

Interpreting Main and Interaction Effect Plots

- The PD plot interaction for any pair of predictors (say x_1 and x_2) is defined as $\hat{f}_{1,2,PD}(x_1, x_2) = \frac{1}{n}\sum_{i=1}^{n} f\left(x_1, x_2, x_{i,3}, \ldots, x_{i,k}\right)$, but "zeroed" so that its two main effects and its average are all zero.
- By definition, the ALE effects have all lower-order effects subtracted out (2nd-order interactions have zero main effects; main effects have zero mean)
- The second-order approximation of $\hat{y}(\mathbf{x})$ (i.e., $f(\mathbf{x})$) is the sum of all main-effects and second-order interactions, plus a constant $f_0 = E[f(\boldsymbol{X})]$

$$f_0 + f_1(x_1) + f_2(x_2) + \cdots + f_k(x_k) + f_{1,2}(x_1, x_2) + f_{1,3}(x_1, x_3) + \cdots + f_{k-1,k}(x_{k-1}, x_k)$$

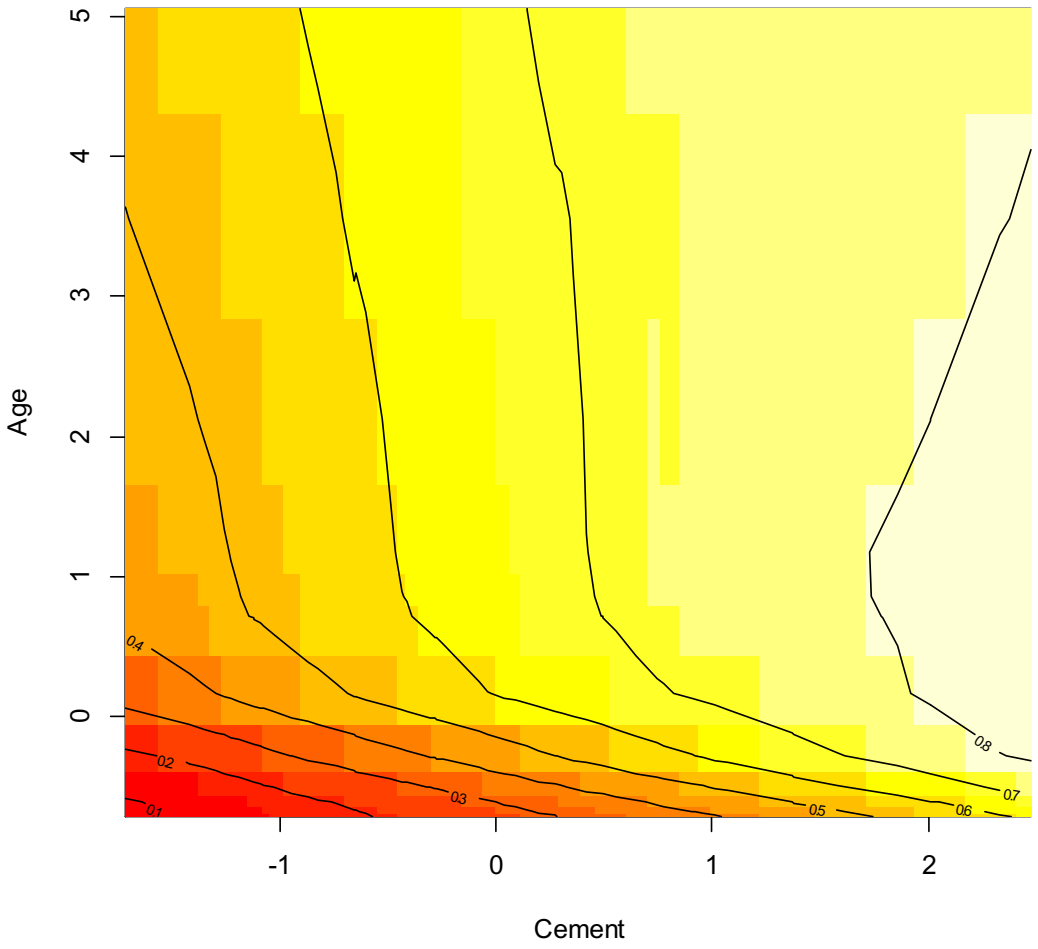
- The combined second-order effect of a single pair of variables (say x_j and x_l) is $f_0 + f_j\left(x_j\right) + f_l(x_l) + f_{j,l}\left(x_j, x_l\right)$, which is what you use to interpret the joint effect of a pair of predictors on the response

Combined Effect of Cement (x_1) and Age (x_8)

- The interaction $f_{1,8}(x_1, x_8)$ is not negligible relative to $f_1(x_1)$ and $f_8(x_8)$, so we should include it when analyzing the combined effect of x_1 and x_8
- Their combined effect is $f_0 + f_1(x_1) + f_8(x_8) + f_{1,8}(x_1, x_8)$, which you get by adding the ALE plot main effects of x_1 and x_8 together with their ALE plot interaction effect and f_0 (the average of all n training y's)

```
f0 <- mean(CRT1$Strength)
f1 <- ALEPlot(CRT1[,1:8], nn1, pred.fun=yhat, J=1, K=50, NA.plot = TRUE)
f8 <- ALEPlot(CRT1[,1:8], nn1, pred.fun=yhat, J=8, K=50, NA.plot = TRUE)
f18 <- ALEPlot(CRT1[,1:8], nn1, pred.fun=yhat, J=c(1,8), K=50, NA.plot = TRUE)
f18.combined <- f0 + outer(f1$f.values,rep(1,13)) + outer(rep(1,51),f8$f.values) +
   f18$f.values
image(f1$x.values, f8$x.values, f18.combined, xlab="Cement", ylab= "Age", xlim =
   range(f1$x.values), ylim = range(f8$x.values), xaxs = "i", yaxs = "i")
contour(f1$x.values, f8$x.values, f18.combined, add=TRUE, drawlabels=TRUE)
```

Plot of Combined Effect of Cement (x_1) and Age (x_8) Showing Their Interaction



Effect of x_1: $-1.5 \rightarrow +2.5$ when $x_8 = -1.0$ is $+0.08 \rightarrow +0.55 = +\mathbf{0.47}$

Effect of x_1: $-1.5 \rightarrow +2.5$ when $x_8 = +5.0$ is $+0.55 \rightarrow +0.78 = +\mathbf{0.23}$

Effect of x_8: $-1.0 \rightarrow +5.0$ when $x_1 = -1.5$ is $+0.08 \rightarrow +0.55 = +\mathbf{0.47}$

Effect of x_8: $-1.0 \rightarrow +5.0$ when $x_1 = +2.5$ is $+0.55 \rightarrow +0.78 = +\mathbf{0.23}$

The effect of Age depends on the value of Cement, and vice-versa, which is the definition of an interaction

Some Calculations for the Preceding

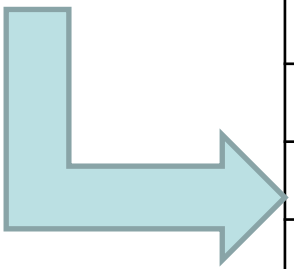
To illustrate the computations that went into the preceding combined effect $f_0 + f_1(x_1) + f_8(x_8) + f_{1,8}(x_1, x_8)$, look at two specific low and high values of x_1 (-1.5, +2.5) and x_8 (-1.0, +5.0)

cement-age interaction $= f_{1,8}(x_1, x_8)$		
	age	
cement	-1.0	+5.0
-1.5	0.02	0.11
+2.5	-0.03	-0.18

cement main effect	
cement	$f_1(x_1)$
-1.5	-0.20
+2.5	+0.32

age main effect	
age	$f_8(x_8)$
-1.0	-0.16
+5.0	+0.22

Average f_0
0.42



combined cement-age effect $= f_0 + f_1(x_1) + f_8(x_8) + f_{1,8}(x_1, x_8)$		
	age	
cement	-1	+5
-1.5	0.42-0.20-0.16+0.02 = 0.08	0.42-0.20+0.22+0.11 = +0.55
+2.5	0.42+0.32-0.16-0.03 = 0.55	0.42+0.32+0.22-0.18 = +0.78

Example: Predictive Modeling of CPU performance

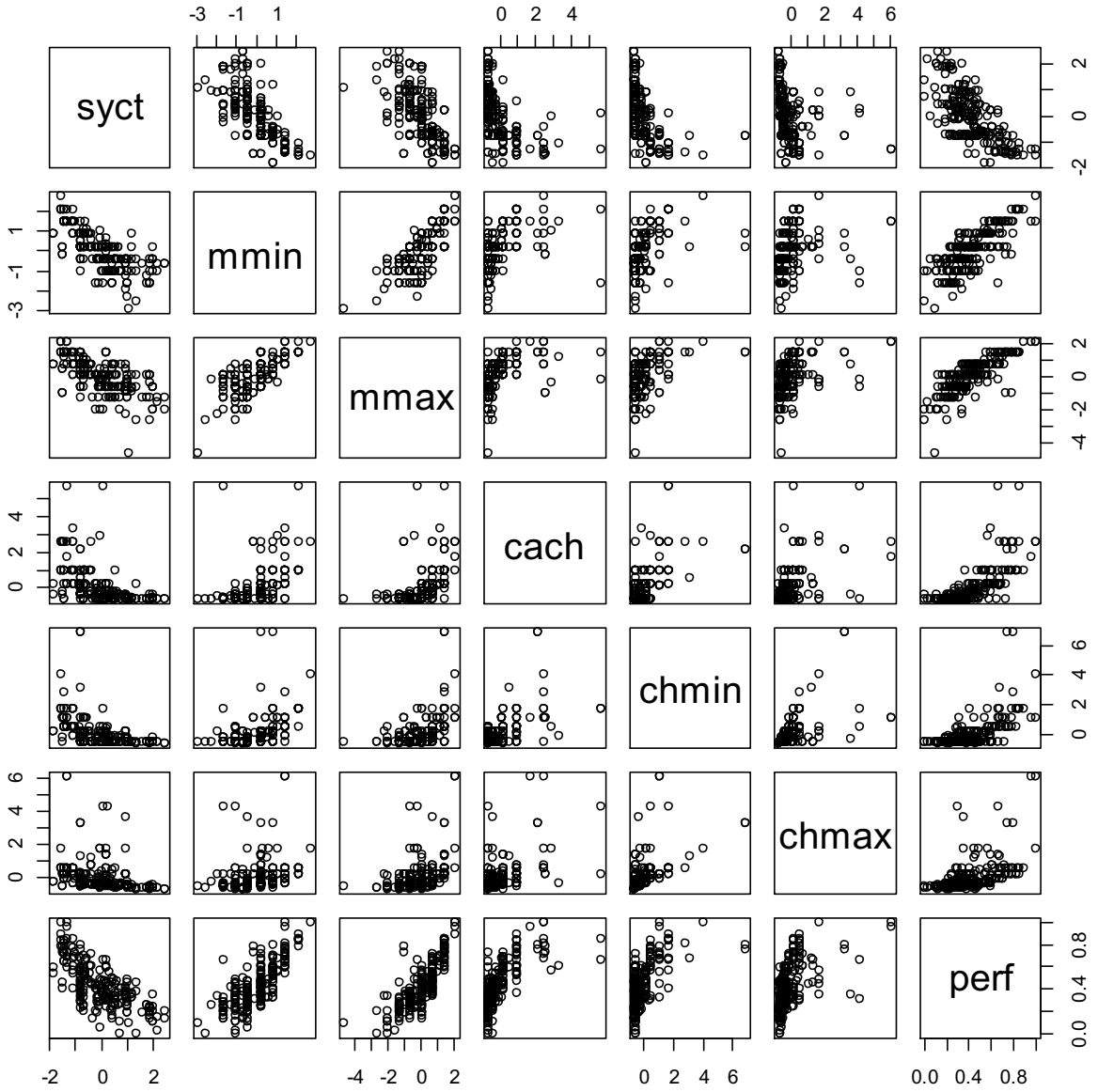
- Data in cpus.txt, which is the same as the cpus data in the MASS package
- 209 cases, with 9 variables and 6 predictor variables
- “perf” is the response, which is CPU performance. Ignore “estperf” which was another author’s estimated performance.
- The six numerical predictors are cycle time (nanoseconds), cache size, min and max main memory size, and min and max number of channels. See V&R for additional discussion
- The objective is to learn the predictive relationship between CPU performance and the predictor variables

Neural Network Modeling of CPU data

########R code for reading in cpus data set, taking log(response) and then converting to [0,1] interval, and standardizing predictors################

```
CPUS<-read.table("cpus.txt",sep="\t")
CPUS1<-CPUS[2:8]
k<-ncol(CPUS1);
CPUS1[c(1:3,k)]<-sapply(CPUS1[c(1:3,k)], log10)  #take log of first three predictors and response
CPUS1[1:(k-1)]<-sapply(CPUS1[1:(k-1)], function(x) (x-mean(x))/sd(x)) #standardize predictors
CPUS1[k]<-(CPUS1[k]-min(CPUS1[k]))/(max(CPUS1[k])-min(CPUS1[k]))
CPUS[1:10,]
pairs(CPUS1)
```

Matrix scatterplot of transformed cpus data



CPUS Example Continued

```
###############Fit a neural network model to the CPUS1 data#######################
library(nnet)
cpus.nn1<-nnet(perf~.,CPUS1,linout=T,skip=F,size=10,decay=.05,maxit=1000,trace=F)
yhat<-as.numeric(predict(cpus.nn1))
y<-CPUS1[[7]]; e<-y-yhat
plot(yhat,y)
c(sd(y),sd(e))
 #repeat but using logistic output function, for which the response MUST BE SCALED TO
    [0,1] RANGE
cpus.nn1<-nnet(perf~.,CPUS1,linout=F,skip=F,size=10,decay=0.05,maxit=1000,trace=F)
yhat<-as.numeric(predict(cpus.nn1))
y<-CPUS1[[7]]; e<-y-yhat
c(sd(y),sd(e))
##
summary(cpus.nn1)
```

Discussion Points and Questions

- Why do we need to standardize the predictors (and the response variable when using a linear output activation function)?
- How can we get r^2 for this example (the nnet function in R does not spit it out)
- Which predictor variables appear to be the most important, and what R output do we look at to determine this?
- What value of λ will give us the smallest training SSE?
- How can we decide the best value of λ?

CPUS Example Continued

```
########A function to determine the indices in a CV partition####################
CVInd <- function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices for each part
   m<-floor(n/K)  #approximate size of each part
   r<-n-m*K
   I<-sample(n,n)  #random reordering of the indices
   Ind<-list()  #will be list of indices for all K parts
   length(Ind)<-K
   for (k in 1:K) {
      if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
         else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
      Ind[[k]] <- I[kpart]  #indices for kth part of data
   }
   Ind
}
```

CPUS Example Continued

```
##Now use the same CV partition to compare Neural Net and linear reg models###
Ind<-CVInd(n=nrow(CPUS1),10)
K<-length(Ind)
y<-CPUS1$perf
yhat<-y
for (k in 1:K) {
  out<-nnet(perf~.,CPUS1[-
        Ind[[k]],],linout=T,skip=T,size=10,decay=0.05,maxit=1000,trace=F)
  yhat[Ind[[k]]]<-as.numeric(predict(out,CPUS1[Ind[[k]],]))
}
e1=y-yhat
#now compare to linear regression with same CV index partition
for (k in 1:K) {
  out<-lm(perf~.,CPUS1[-Ind[[k]],])
  yhat[Ind[[k]]]<-as.numeric(predict(out,CPUS1[Ind[[k]],]))
}
e2=y-yhat
c(sd(e1),sd(e2))
```

Discussion Points and Questions

- The best value of λ is the value that results in the smallest SSE_{CV} (or equivalently, the largest CV r^2, smallest CV SD(e), etc).
- How can we decide the best number of hidden layer nodes?
- Why should we use the same CV partition when comparing two models?
- What are the pros and cons of n-fold CV versus K-fold CV for other K, e.g., 3, 5, or 10?

Example: Predictive Modeling of Income Data

- Data in adult_train.csv is from the 1994 US Census (also see http://archive.ics.uci.edu/ml/datasets/Census+Income)
- 32561 cases, with 15 variables. This is a small sample from the US census with 15 potentially relevant variables. Each row represents a "similar" population segment with weight given by "fnlwgt"
- income has been converted to a binary categorical variable (<= or > 50k) with roughly a 75%/25% population split
- Later we will fit predictive models to classify income based on the other variables (classification). Here, the objective is to predict the number of hours per week spent working based on the other variables (regression)
- This is already a very cleaned data set, but we may need to do a little additional cleaning
  - What should we do about the missing "?" values

The First Few Rows

age	workclass	fnlwgt	education	education-	marital-sta	occupatio	relationshi	race	sex	capital-gai	capital-los	hours-per-	native-cou	income
39	State-gov	77516	Bachelors	13	Never-ma	Adm-cleri	Not-in-far	White	Male	2174	0	40	United-Sta	<=50K
50	Self-emp-	83311	Bachelors	13	Married-c	Exec-man	Husband	White	Male	0	0	13	United-Sta	<=50K
38	Private	215646	HS-grad	9	Divorced	Handlers-	Not-in-far	White	Male	0	0	40	United-Sta	<=50K
53	Private	234721	11th	7	Married-c	Handlers-	Husband	Black	Male	0	0	40	United-Sta	<=50K
28	Private	338409	Bachelors	13	Married-c	Prof-speci	Wife	Black	Female	0	0	40	Cuba	<=50K
37	Private	284582	Masters	14	Married-c	Exec-man	Wife	White	Female	0	0	40	United-Sta	<=50K
49	Private	160187	9th	5	Married-s	Other-ser	Not-in-far	Black	Female	0	0	16	Jamaica	<=50K
52	Self-emp-	209642	HS-grad	9	Married-c	Exec-man	Husband	White	Male	0	0	45	United-Sta	>50K
31	Private	45781	Masters	14	Never-ma	Prof-speci	Not-in-far	White	Female	14084	0	50	United-Sta	>50K
42	Private	159449	Bachelors	13	Married-c	Exec-man	Husband	White	Male	5178	0	40	United-Sta	>50K
37	Private	280464	Some-coll	10	Married-c	Exec-man	Husband	Black	Male	0	0	80	United-Sta	>50K
30	State-gov	141297	Bachelors	13	Married-c	Prof-speci	Husband	Asian-Pac	Male	0	0	40	India	>50K
23	Private	122272	Bachelors	13	Never-ma	Adm-cleri	Own-child	White	Female	0	0	30	United-Sta	<=50K
32	Private	205019	Assoc-acd	12	Never-ma	Sales	Not-in-far	Black	Male	0	0	50	United-Sta	<=50K
40	Private	121772	Assoc-voc	11	Married-c	Craft-repa	Husband	Asian-Pac	Male	0	0	40	?	>50K
34	Private	245487	7th-8th	4	Married-c	Transport	Husband	Amer-Indi	Male	0	0	45	Mexico	<=50K
25	Self-emp-	176756	HS-grad	9	Never-ma	Farming-fi	Own-child	White	Male	0	0	35	United-Sta	<=50K
32	Private	186824	HS-grad	9	Never-ma	Machine-	Unmarried	White	Male	0	0	40	United-Sta	<=50K
38	Private	28887	11th	7	Married-c	Sales	Husband	White	Male	0	0	50	United-Sta	<=50K
43	Self-emp-	292175	Masters	14	Divorced	Exec-man	Unmarried	White	Female	0	0	45	United-Sta	>50K
40	Private	193524	Doctorate	16	Married-c	Prof-speci	Husband	White	Male	0	0	60	United-Sta	>50K
54	Private	302146	HS-grad	9	Separated	Other-ser	Unmarried	Black	Female	0	0	20	United-Sta	<=50K
35	Federal-go	76845	9th	5	Married-c	Farming-fi	Husband	Black	Male	0	0	40	United-Sta	<=50K
43	Private	117037	11th	7	Married-c	Transport	Husband	White	Male	0	2042	40	United-Sta	<=50K
59	Private	109015	HS-grad	9	Divorced	Tech-supp	Unmarried	White	Female	0	0	40	United-Sta	<=50K
56	Local-gov	216851	Bachelors	13	Married-c	Tech-supp	Husband	White	Male	0	0	40	United-Sta	>50K
19	Private	168294	HS-grad	9	Never-ma	Craft-repa	Own-child	White	Male	0	0	40	United-Sta	<=50K
54	?	180211	Some-coll	10	Married-c	?	Husband	Asian-Pac	Male	0	0	60	South	>50K
39	Private	367260	HS-grad	9	Divorced	Exec-man	Not-in-far	White	Male	0	0	80	United-Sta	<=50K

Read in the Data

```
XX<-read.table("adult_train.csv",sep=",",header=TRUE,strip.white=TRUE,na.strings="?")
XX<-na.omit(XX)
INCOME<-XX
INCOME[,c(1,5,11,12,13)]<-scale(INCOME[,c(1,5,11,12,13)]) #standardize the
    continuous variables
```

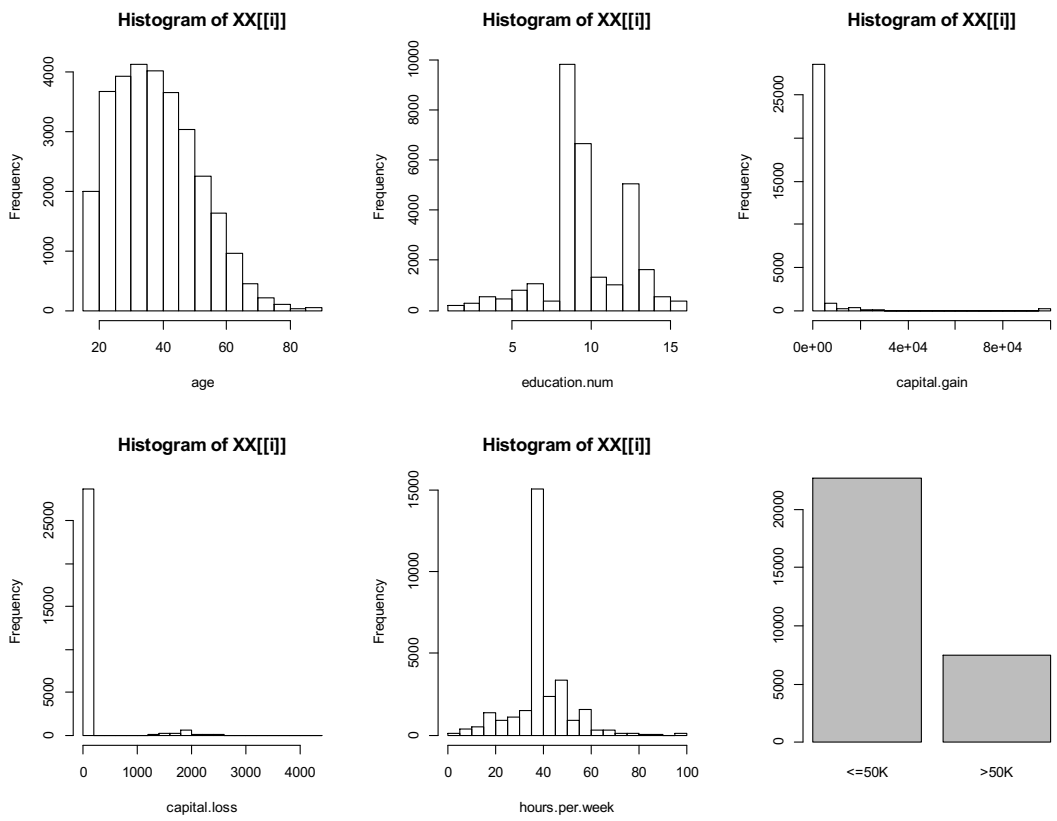
- strip.white=TRUE removes leading/trailing blank space in character fields, so it can recognize the "?" missing values

- What should we do about "fnlwgt", which is a weight assigned to the segment of the population represented by each row?

Some Preliminary Exploratory Analyses

```
##exploring individual variables
summary(INCOME)
par(mfrow=c(2,3)); for (i in c(1,5,11,12,13)) hist(XX[[i]],xlab=names(XX)[i]); plot(XX[[15]])
par(mfrow=c(1,1)); plot(XX[[2]],cex.names=.7)
for (i in c(2,4,6,7,8,9,10,14,15)) print(round(table(XX[[i]])/nrow(XX),3))
```

Should we be concerned with anything here or do any further cleaning?

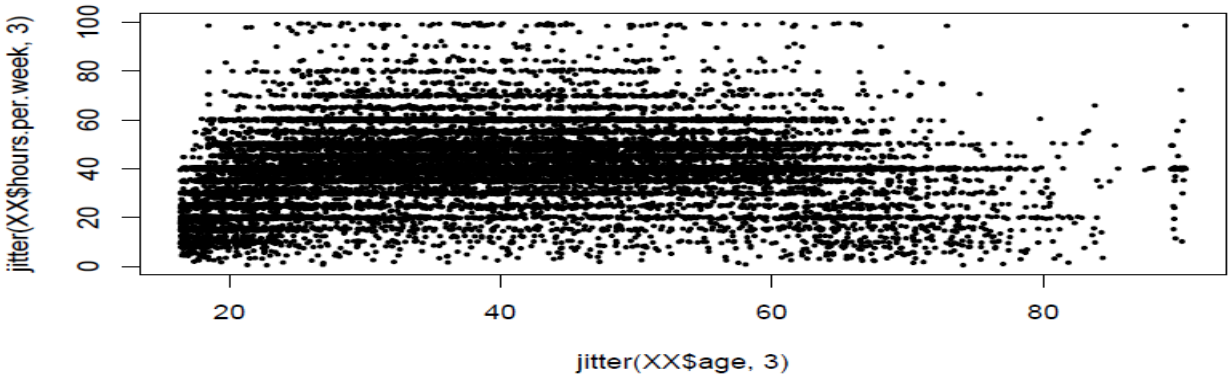


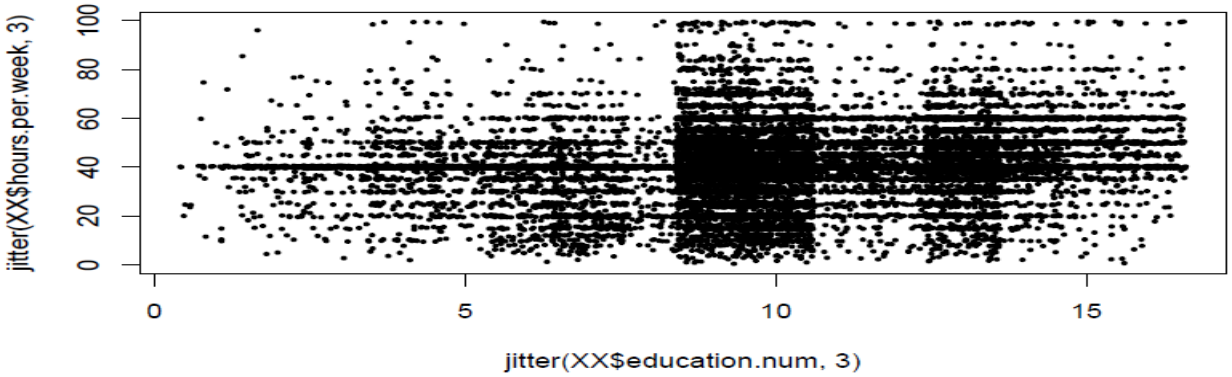
Some Preliminary Exploratory Analyses

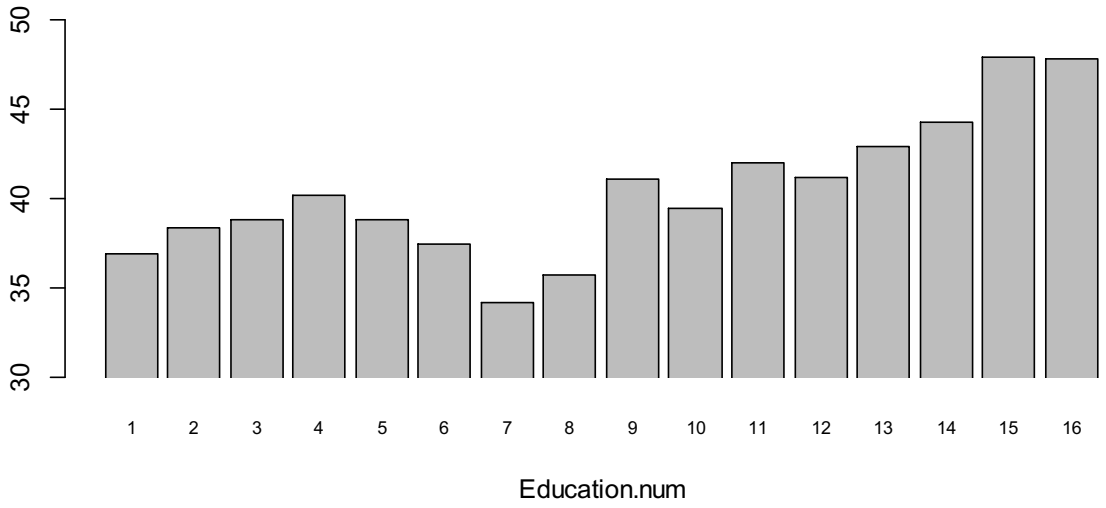
```
##exploring pairwise predictor/response relationships
par(mfrow=c(2,1))
plot(jitter(XX$age,3),jitter(XX$hours.per.week,3),pch=16,cex=.5)
plot(jitter(XX$education.num,3),jitter(XX$hours.per.week,3),pch=16,cex=.5)
par(mfrow=c(1,1))
barplot(tapply(XX$hours.per.week,XX$education.num,mean),ylim=c(30,50),cex.names=.7,xpd=F,
      xlab="Education.num")
for (i in c(2,4,6,7,8,9,14,15)) {print(round(tapply(XX$hours.per.week,XX[[i]],mean),2)); cat("\n")}
```

Some points to consider regarding correlation versus functional dependence (points that apply to ANY regression analysis)

- If hours.per.week appears correlated with another variable, it does not mean that hours.per.week has a functional dependence on that variable
- The two could appear correlated because they both depend on another variable (either one of the existing variables or an unrecorded nuisance variable)
- If you have recorded enough nuisance variables, a multiple regression analysis can sometimes distinguish which correlations are truly due to a functional dependence
- If your goal is pure prediction (and not explanatory), does it matter?


jitter(XX$hours.per.week, 3)
0
20
40
60
80
100
20
40
60
80
jitter(XX$age, 3)


jitter(XX$hours.per.week, 3)
0
20
40
60
80
100
0
5
10
15
jitter(XX$education.num, 3)


30
35
40
45
50
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Education.num

A Typical Next Step in Predictive Modeling

```
##linear regression with all predictors included
Inc.lm<-lm(hours.per.week ~ .,data=INCOME[,-3])
summary(Inc.lm)
##
```

- Why does the output for the predictor education.num say "NA"?
- What are the considerations and pros and cons of using education vs. education.num as a predictor?
- What conveys clearer information – the scatteplot of hours.per.week versus education.num on the previous slide, or the barplot?

Some Typical Next Steps

```
##linear regression including interactions (generally not a good next step)
Inc.lm.full<-lm(hours.per.week ~ .^2,data=INCOME[,-c(3,4,14)])
summary(Inc.lm.full)
```

- Why remove native.country when including interactions?
- How do you interpret the model with interactions?

```
##stepwise linear regression
Inc.lm0<-lm(hours.per.week ~ 1,data=INCOME[,-c(3,4)])
Inc.lm<-lm(hours.per.week ~ .,data=INCOME[,-c(3,4)])
Inc.lm.step<-step(Inc.lm0, scope=formula(Inc.lm), direction="both", trace=0)
summary(Inc.lm.step)
```

- How do you interpret the coefficients? How much do the various predictors impact hours.per.week?
- Why might stepwise linear regression give a better idea of which predictors are important than the t-test P-values for the model with all predictors included?
- Why not use scope=formula(Inc.lm.full) in the preceding stepwise regression?

Now Try a Neural Network Model

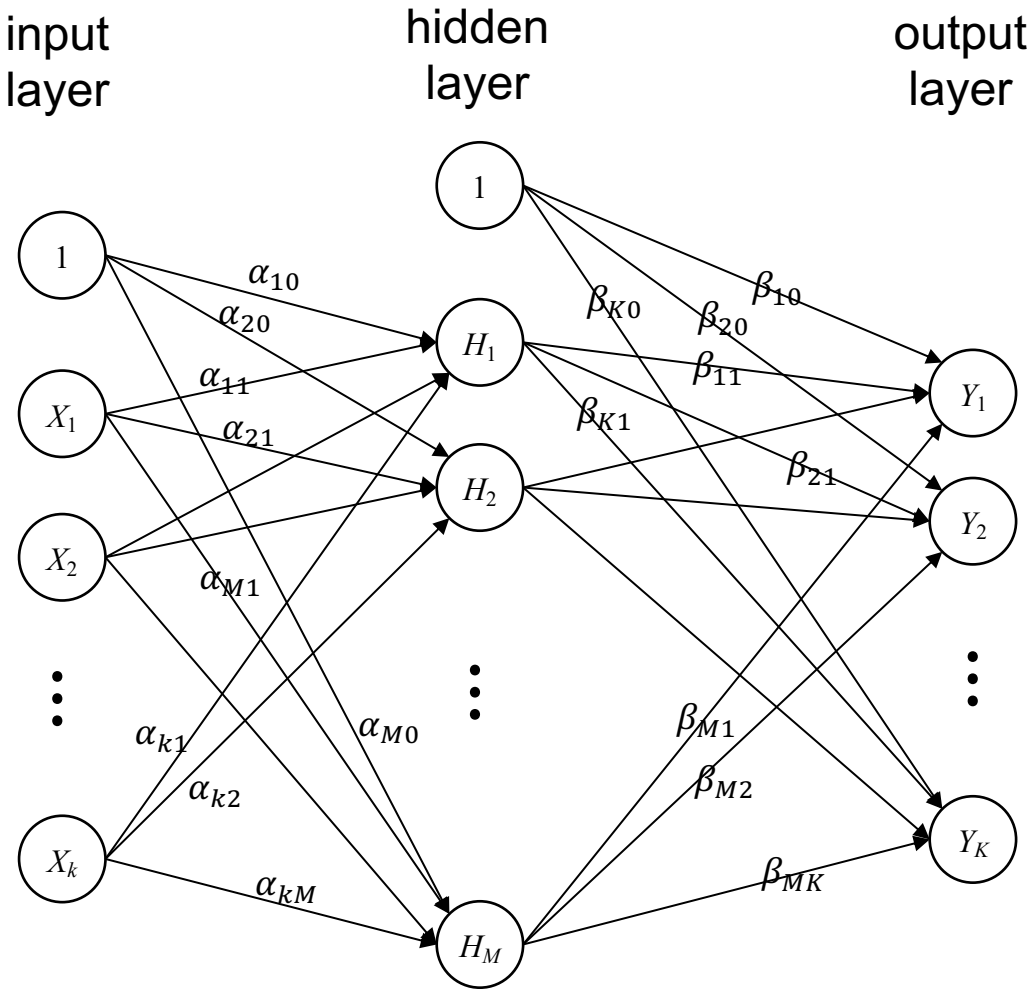
```
##Neural network model
library(nnet)
Inc.nn1<-nnet(hours.per.week ~ . ,INCOME[,-c(3,4)], linout=T, skip=F, size=10, decay=.05,
      maxit=100, trace=F)
yhat<-as.numeric(predict(Inc.nn1))
y<-INCOME$hours.per.week; e<-y-yhat
1-var(e)/var(y) #training r^2
summary(Inc.nn1)
yhat <- function(X.model, newdata) as.numeric(predict(X.model, newdata))
ALEPlot(INCOME[,-c(3,4,13)], Inc.nn1, pred.fun=yhat, J=1, K=500, NA.plot = TRUE);
```

- How did the nnet() function treat the categorical predictors?
- Why is maxit reduced to 100?
- training r^2 ~ 29% for size = 10 and decay = 0.05 (although it varies from implementation to implementation), which is a little better than for the linear regression with interactions (r^2_adj ~ 24.7%). In comparing the two models, what factors should you consider?
- If you used CV to evaluate the model (which you absolutely should do before selecting the best model), how would you implement CV?

Multi-Response Neural Networks

- Neural networks also apply to the situation in which we have more than one (say K) response variables
- We handle this by including K nodes in the output layer (see the following slide)
- This is different than fitting K separate neural networks, one for each response, because the K responses share the same hidden layer node functions
- This is generally more effective than fitting K separate neural networks models if the response variables have similar functional dependencies on the predictors. If the responses have completely different dependencies on the predictors, then you are better off fitting K separate neural networks models

Graphical Depiction of a (single hidden layer) Neural Network with K Response Variables



Neural Networks for Classification

- The most common application of multi-response neural networks is for classification when we have a categorical response with K categories (aka classes). Note that this also applies to binary responses (K = 2)
- To handle this (most software does this internally), make a K-length 0/1 response vector, e.g., for the fgl data:

Type	y_1	y_2	y_3	y_4	y_5	y_6
WinF	1	0	0	0	0	0
WinNF	0	1	0	0	0	0
Veh	0	0	1	0	0	0
Con	0	0	0	1	0	0
Tab1	0	0	0	0	1	0
Head	0	0	0	0	0	1

Example: Predicting Glass Type in Forensics

- Data in fgl.txt, which is the same as the FGL data in the MASS package. See V&R for additional discussion
- 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

Read the Data and Transform some Variables

```
######Read data, convert response to binary, and standardize predictors#####
FGL<-read.table("fgl.txt",sep="\t")
z<-(FGL$type == "WinF") | (FGL$type == "WinNF")
y<-as.character(FGL$type)
y[z]<-"Win"; y[!z]<-"Other"
FGL<-data.frame(FGL,"type_bin"=as.factor(y))  #add a binary factor response column
y[y == "Win"]<-1;y[y == "Other"]<-0;
FGL<-data.frame(FGL,"type01"=as.numeric(y))  #also add a binary numeric response
      column
FGL1<-FGL
k<-ncol(FGL)-3;
FGL1[1:k]<-sapply(FGL1[1:k], function(x) (x-mean(x))/sd(x)) #standardize predictors
FGL
```

First Few Rows of fgl.txt data

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	type
3.01	13.64	4.49	1.1	71.78	0.06	8.75	0	0	WinF
-0.39	13.89	3.6	1.36	72.73	0.48	7.83	0	0	WinF
-1.82	13.53	3.55	1.54	72.99	0.39	7.78	0	0	WinF
-0.34	13.21	3.69	1.29	72.61	0.57	8.22	0	0	WinF
-0.58	13.27	3.62	1.24	73.08	0.55	8.07	0	0	WinF
-2.04	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26	WinF
-0.57	13.3	3.6	1.14	73.09	0.58	8.17	0	0	WinF
-0.44	13.15	3.61	1.05	73.24	0.57	8.24	0	0	WinF
1.18	14.04	3.58	1.37	72.08	0.56	8.3	0	0	WinF
-0.45	13	3.6	1.36	72.99	0.57	8.4	0	0.11	WinF
-2.29	12.72	3.46	1.56	73.2	0.67	8.09	0	0.24	WinF
-0.37	12.8	3.66	1.27	73.01	0.6	8.56	0	0	WinF
-2.11	12.88	3.43	1.4	73.28	0.69	8.05	0	0.24	WinF
-0.52	12.86	3.56	1.27	73.21	0.54	8.38	0	0.17	WinF
-0.37	12.61	3.59	1.31	73.29	0.58	8.5	0	0	WinF
-0.39	12.81	3.54	1.23	73.24	0.58	8.39	0	0	WinF
-0.16	12.68	3.67	1.16	73.11	0.61	8.7	0	0	WinF
3.96	14.36	3.85	0.89	71.36	0.15	9.15	0	0	WinF
1.11	13.9	3.73	1.18	72.12	0.06	8.89	0	0	WinF
-0.65	13.02	3.54	1.69	72.73	0.54	8.44	0	0.07	WinF

Mathematical Definition of K-Class Neural Network Model

for $m = 1, 2, \ldots, M$,

$$H_m = \frac{exp\{\alpha_{m,0} + \alpha_{m,1} x_1 + \cdots + \alpha_{m,k} x_k\}}{1 + exp\{\alpha_{m,0} + \alpha_{m,1} x_1 + \cdots + \alpha_{m,k} x_k\}} \text{ (same as before)}$$

for $l = 1, 2, \ldots, K$,

$$Pr[Y_l = 1 \mid \mathbf{x}] = \frac{exp\{\beta_{l,0} + \beta_{l,1} H_1 + \cdots + \beta_{l,M} H_M\}}{\sum_{j=1}^{K} exp\{\beta_{j,0} + \beta_{j,1} H_1 + \cdots + \beta_{j,M} H_M\}}$$

(multinomial logistic model)

Note: For $K = 2$, this reduces to:

$$Pr[Y_1 = 1 \mid \mathbf{x}] = \frac{exp\{\beta_0 + \beta_1 H_1 + \cdots + \beta_M H_M\}}{1 + exp\{\beta_0 + \beta_1 H_1 + \cdots + \beta_M H_M\}}$$

Fitting A Neural Network Model for Classification

1--3) The first three steps are the same as before

4) For classification, software estimates parameters to minimize (MLE with shrinkage):

$$-\sum_{i=1}^{n}\sum_{l}^{K} y_{i,l}\, log\, \hat{Pr}\left[Y_{i,l}=1 \mid \mathbf{x}_i\right] + \lambda\left(\sum_{all\ m}\sum_{all\ j} a_{m,j}^2 + \sum_{all\ l}\sum_{all\ m} \beta_{l,m}^2\right)$$

(–log-likelihood + shrinkage penalty)

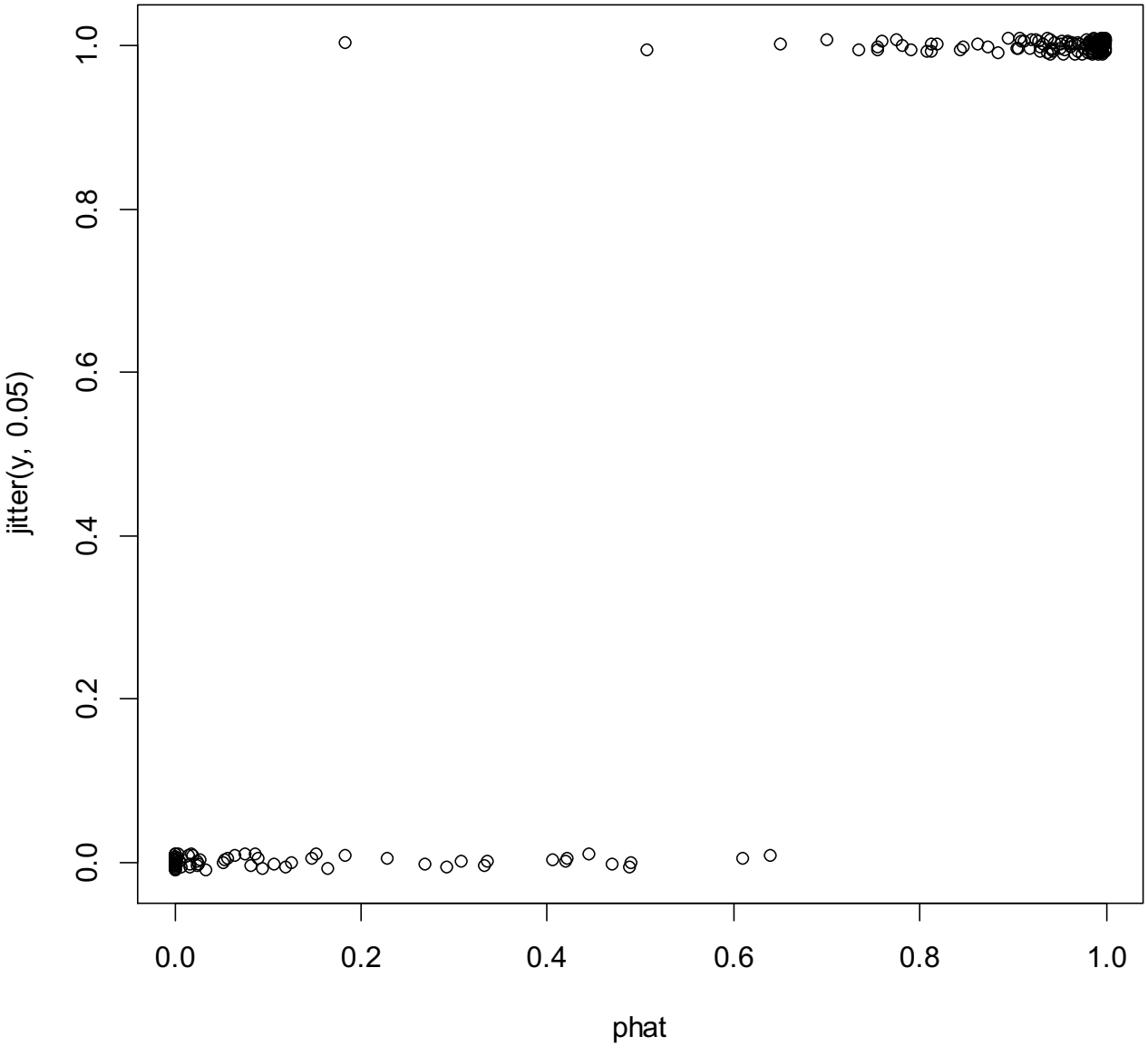
5) CV should be used to choose any tuning parameters (λ, number of nodes, etc)

Fitting a Neural Net Classifier for the FGL Data (binary response case)

```
###############Fit a neural network classification model to the FGL1 data######
library(nnet)
fgl.nn1<-nnet(type_bin~., FGL1[,c(1:9,11)], linout=F, skip=F, size=10, decay=.05,
    maxit=1000, trace=F)
phat<-as.numeric(predict(fgl.nn1))
y<-FGL1[[12]]
yhat<-as.numeric(phat >= 0.5)  #classify as 1 if predicted probability >= 0.5
sum(y != yhat)/length(y)  #misclassification rate
summary(fgl.nn1)
plot(phat,jitter(y,0.05))
```

- Note that nnet() can handle a response that is a factor (even if there are K > 2 categories). You do NOT have to convert it to K 0/1 dummy variables (this is done internally).
- Questions: The training misclassification rate is around 2%
  - Is this good?
  - What factors should we consider to evaluate how good this is?

response vs. predicted probability for fgl data



Using CV to Compare Models for the FGL Data

```
##Now use multiple reps of CV to compare Neural Nets and logistic reg models###
Nrep<-20 #number of replicates of CV
K<-3  #K-fold CV on each replicate
n.models = 3 #number of different models to fit
n=nrow(FGL1)
y<-FGL1[[12]]
yhat=matrix(0,n,n.models)
CV.rate<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
     out<-nnet(type_bin~.,FGL1[-Ind[[k]],c(1:9,11)],linout=F,skip=F,size=10,decay=.3, maxit=1000,trace=F)
     phat<-as.numeric(predict(out,FGL1[Ind[[k]],c(1:9,11)]));  yhat[Ind[[k]],1]<-as.numeric(phat >= 0.5)
     out<-nnet(type_bin~.,FGL1[-Ind[[k]],c(1:9,11)],linout=F,skip=F,size=10,decay=0, maxit=1000,trace=F)
     phat<-as.numeric(predict(out,FGL1[Ind[[k]],c(1:9,11)]));  yhat[Ind[[k]],2]<-as.numeric(phat >= 0.5)
     out<-glm(type_bin~.,FGL1[-Ind[[k]],c(1:9,11)],family=binomial(link="logit"))
     phat<-as.numeric(predict(out,FGL1[Ind[[k]],c(1:9,11)],type="response"));  yhat[Ind[[k]],3]<-as.numeric(phat >= 0.5)
  } #end of k loop
  CV.rate[j,]=apply(yhat,2,function(x) sum(y != x)/n)
} #end of j loop
CV.rate
CV.rateAve<- apply(CV.rate,2,mean); CV.rateAve #averaged CV misclass rate
CV.rateSD <- apply(CV.rate,2,sd); CV.rateSD   #SD of CV misclass rate
```

Discussion Points and Questions

- What is the best neural network model, in terms of the tuning parameters (decay, size, etc)?
- What is the best CV misclassification rate?
- Is this good?
- Which predictors are the most important, and what are their effects?
- What other model(s) would you compare to the best neural network?

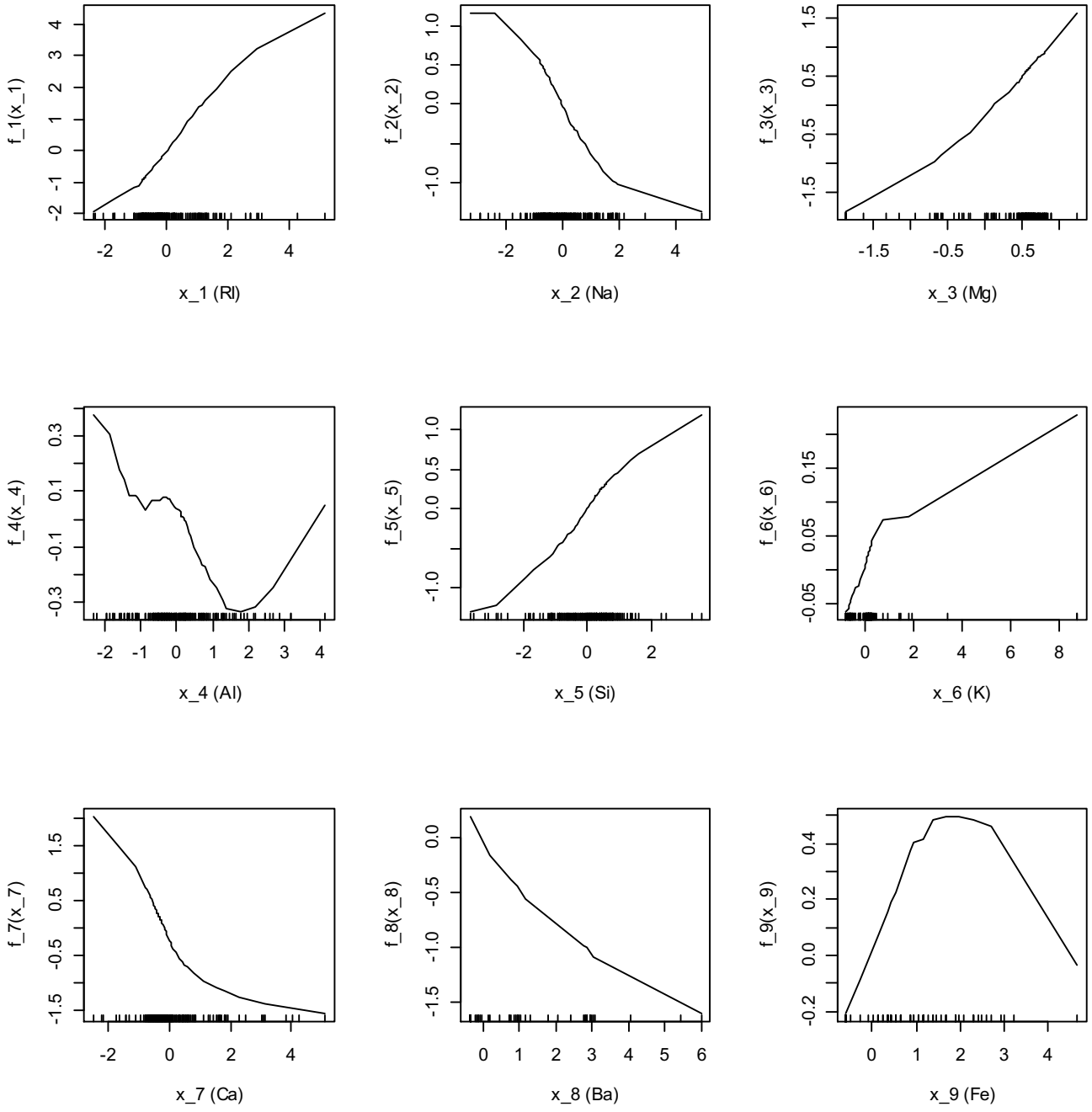
PD and ALE Plots for Classification Models

- ALE and PD plots can be constructed for any predictive function $f(\boldsymbol{x})$
- For regression models, choosing $f(\boldsymbol{x}) = \hat{y}(\boldsymbol{x})$ almost always makes the most sense
- For binary classification models, we could choose either:
  - $f(\boldsymbol{x}) = \widehat{Pr}[Y = 1|\boldsymbol{X} = \boldsymbol{x}]$, or
  - $f(\boldsymbol{x}) = log\left(\frac{\widehat{Pr}[Y=1|\boldsymbol{X}=\boldsymbol{x}]}{1-\widehat{Pr}[Y=1|\boldsymbol{X}=\boldsymbol{x}]}\right)$ (i.e., log-odds)
  - I generally prefer the log-odds (why?)
- For classification models with > 2 response classes, can choose $f(\boldsymbol{x}) = log\left(\frac{\widehat{Pr}[Y=k|\boldsymbol{X}=\boldsymbol{x}]}{1-\widehat{Pr}[Y=k|\boldsymbol{X}=\boldsymbol{x}]}\right)$ for as many response classes k as we like

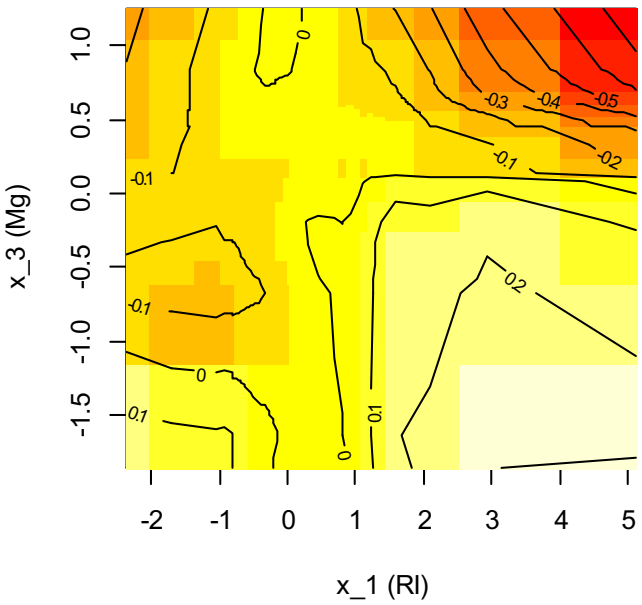
ALE Plots for the FGL Example

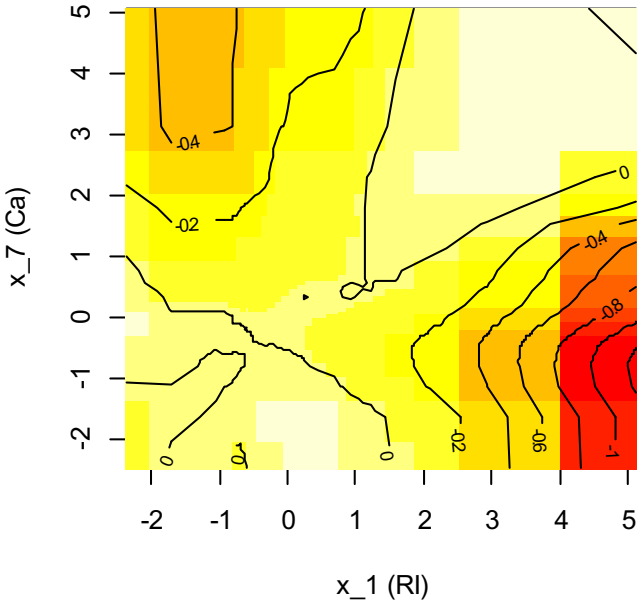
```
###############Visualizing the effects of the predictors#####################
fgl.nn1<-nnet(type_bin~., FGL1[,c(1:9,11)], linout=F, skip=F, size=10, decay=.3,
      maxit=1000, trace=F) ##From CV, these are about the best tuning parameters
summary(fgl.nn1)

## Use ALEPlot package to create accumulated local effects (ALE) plots
library(ALEPlot)
yhat <- function(X.model, newdata) {p.hat = as.numeric(predict(X.model, newdata,
      type="raw")); log(p.hat/(1-p.hat))}  ## to plot the log-odds
par(mfrow=c(3,3))
for (j in 1:9)  {ALEPlot(FGL1[,1:9], fgl.nn1, pred.fun=yhat, J=j, K=50, NA.plot = TRUE)
  rug(FGL1[,j]) }  ## This creates main effect ALE plots for all 9 predictors
par(mfrow=c(1,1))

par(mfrow=c(2,2))  ## This creates 2nd-order interaction ALE plots for x1, x3, x7
ALEPlot(FGL1[,1:9], fgl.nn1, pred.fun=yhat, J=c(1,3), K=50, NA.plot = TRUE)
ALEPlot(FGL1[,1:9], fgl.nn1, pred.fun=yhat, J=c(1,7), K=50, NA.plot = TRUE)
ALEPlot(FGL1[,1:9], fgl.nn1, pred.fun=yhat, J=c(3,7), K=50, NA.plot = TRUE)
par(mfrow=c(1,1))
```

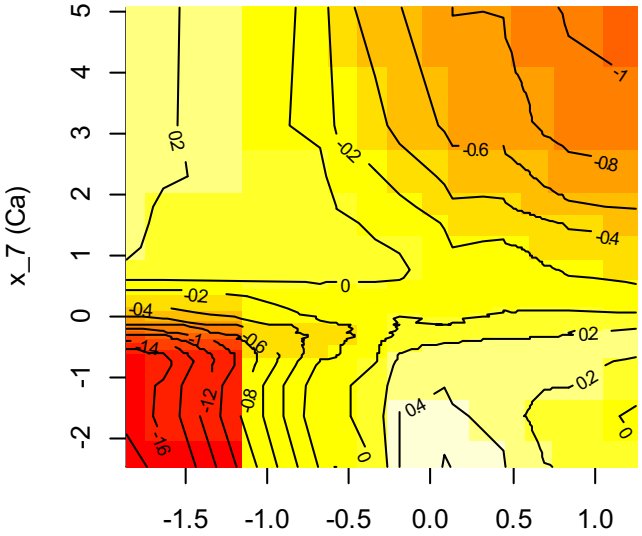
Main Effect ALE plots



Some 2nd-Order Interaction ALE plots







Classification for the 6-Class FGL Response

```
##############Same, but use the original 6-category response#######
library(nnet)
fgl.nn1<-
   nnet(type~.,FGL1[,c(1:10)],linout=F,skip=F,size=10,decay=.05,maxit=1000,trace=F)
##output the class probabilities
phat<-predict(fgl.nn1,type="raw")
phat[1:20,]
apply(phat,1,sum)  #you can see that the 6 predicted class probabilities sum to 1.0
##output the class with the largest class probability
yhat<-predict(fgl.nn1,type="class")
yhat
y<-FGL1$type
sum(y != yhat)/length(y)  #training misclassification rate
```

- Misclassification rates are generally worse the more response classes there are
- As always, you should use the CV misclassification rate, not the training misclassification rate, to evaluate the model (left as an exercise)

Neural Network Classification of Income Data

- Reconsider the data in adult_train.csv
- Instead of predicting the number of hours (regression), we will now predict the binary income categorization (<= 50k vs. > 50k) using the other predictor variables
- Recall that for the entire sample, we 75% are <= 50k and 25% are > 50k

Read the Data and Fit Models

```
XX<-read.table("adult_train.csv",sep=",",header=TRUE,strip.white=TRUE,na.strings="?")
XX<-na.omit(XX)
INCOME<-XX
INCOME[,c(1,5,11,12,13)]<-scale(INCOME[,c(1,5,11,12,13)]) #standardize the
      continuous variables
library(nnet)
Inc.nn1<-nnet(income ~ . ,INCOME[,-c(3,4)], linout=F, skip=F, size=10, decay=.05,
   maxit=100, trace=F)
y<-INCOME$income
phat<-predict(Inc.nn1, type="raw")   #vector of predicted  probabilities
yhat<-predict(Inc.nn1, type="class")  #vector of predicted classes
sum(y != yhat)/length(y)  #training misclassification rate
plot(phat,jitter(as.numeric(y==">50K"), 1.5))

##compare to a logistic regression model
Inc.glm <- glm(income ~ . ,family = binomial(link = "logit"), data=INCOME[,-c(3,4)])
phat <- predict(Inc.glm, type="response")
summary(Inc.glm)
sum((y == ">50K") != (phat > 0.5))/length(y)
```

Discussion Points and Questions

- Which model – neural network or logistic regression – appears to be better?
- How good does it appear?

Pros and Cons of Neural Networks

- Pros:
  - very flexible; with enough nodes, can model almost any nonlinear relationship
  - can efficiently model linear behavior if the relationship is truly linear
  - often very good predictive power
  - a lot of ways to customize the architecture to tackle specific modeling problems (recurrent NNs; convolutional NNs; autoencoders; other “deep learning” architectures ...)
- Cons:
  - model fitting can be unstable and sensitive to initial guesses
  - for very large data sets, model fitting can be very slow relative to some methods like trees and linear models, which makes CV very computationally expensive
  - overfitting (but can avoid by using CV to choose λ)
  - sensitive to user-chosen "tuning parameters” (but can use CV to choose them wisely)
  - poor interpretability, but this can be improved with ALE or PD plots

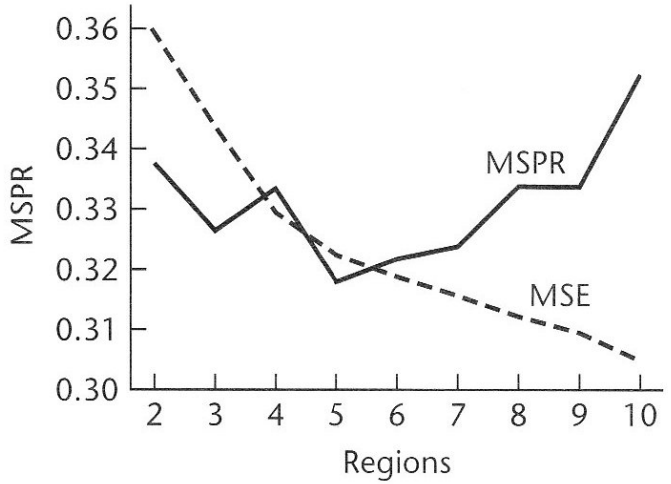
Classification and Regression Tree (CART) Models

- Perhaps the single most widely used generic nonlinear modeling method
- Very simple idea and very interpretable models
- They usually do not have the best predictive power, but they serve as the basis for many more advanced supervised learning methods (e.g., boosting, random forests) that have excellent predictive power
- As with neural networks (and most of the methods we will cover), you can use tree models for either regression or classification. We will start with regression.

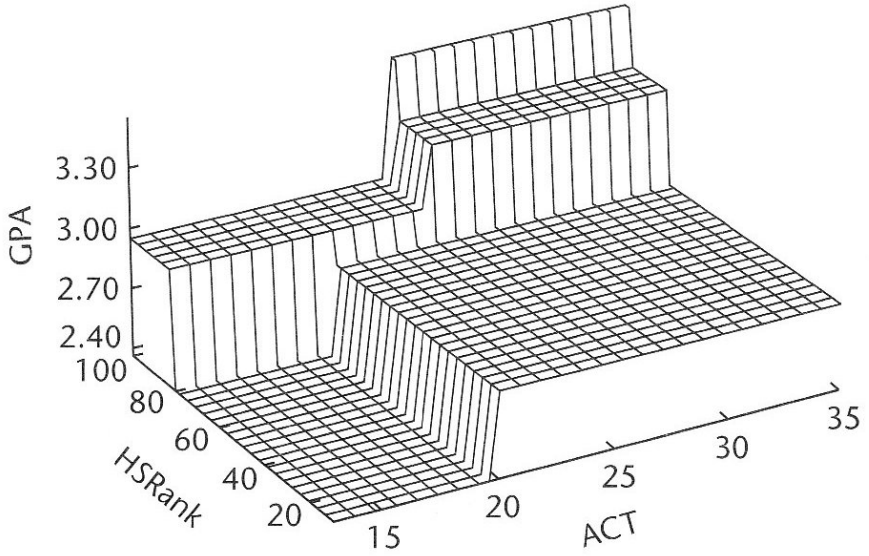
Structure of a Regression Tree

- A final fitted CART model divides the predictor ($\mathbf{x}$) space by successively splitting into rectangular regions and models 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 (aka leaf) represents one region and indicates the value of the predicted response in that region
- The following slide illustrates a fitted tree model for an example from the KNN text (Figure 11.12), in which the objective is to predict college GPA (the response) as a function of HS rank and ACT score (two predictors)
- To use a fitted CART to predict a new case, you start at the root node and follow the splitting rules down to a leaf

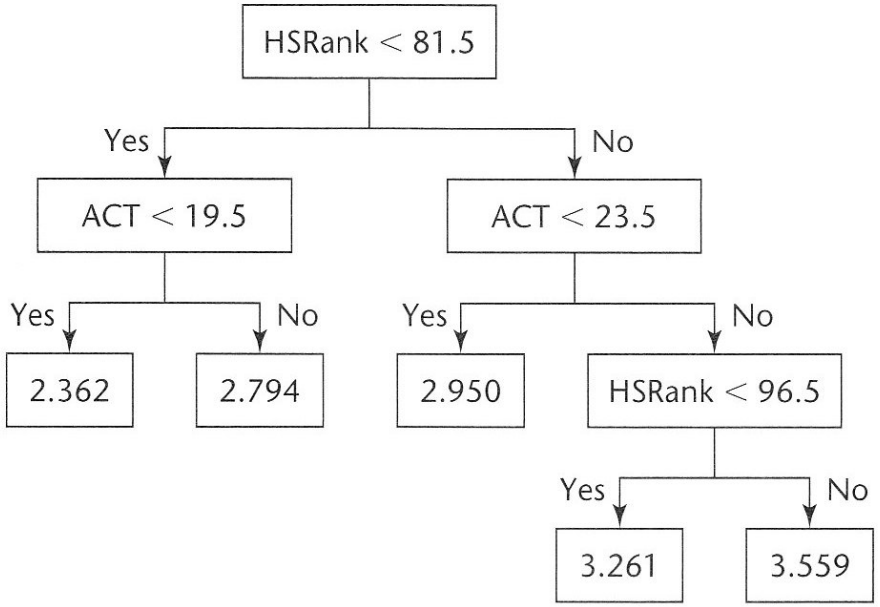
FIGURE 11.12 S-Plus Regression Tree Results—University Admissions Example.



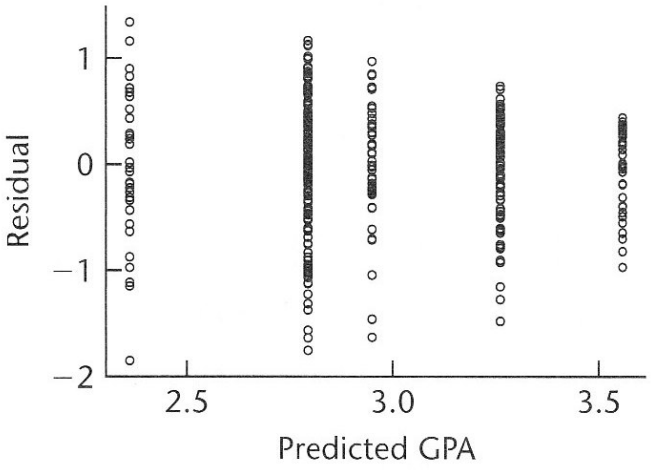
(a)



(b)



(c)



(d)

Mathematical Representation of Regression Tree

Can still view tree model as $Y = g(\mathbf{x};\boldsymbol{\theta}) + \varepsilon$ where:

$$g(\mathbf{x};\boldsymbol{\theta}) = \sum_{m=1}^{M} c_m I(\mathbf{x} \in R_m)$$

M = total number of regions (terminal nodes)

R_m = mth region

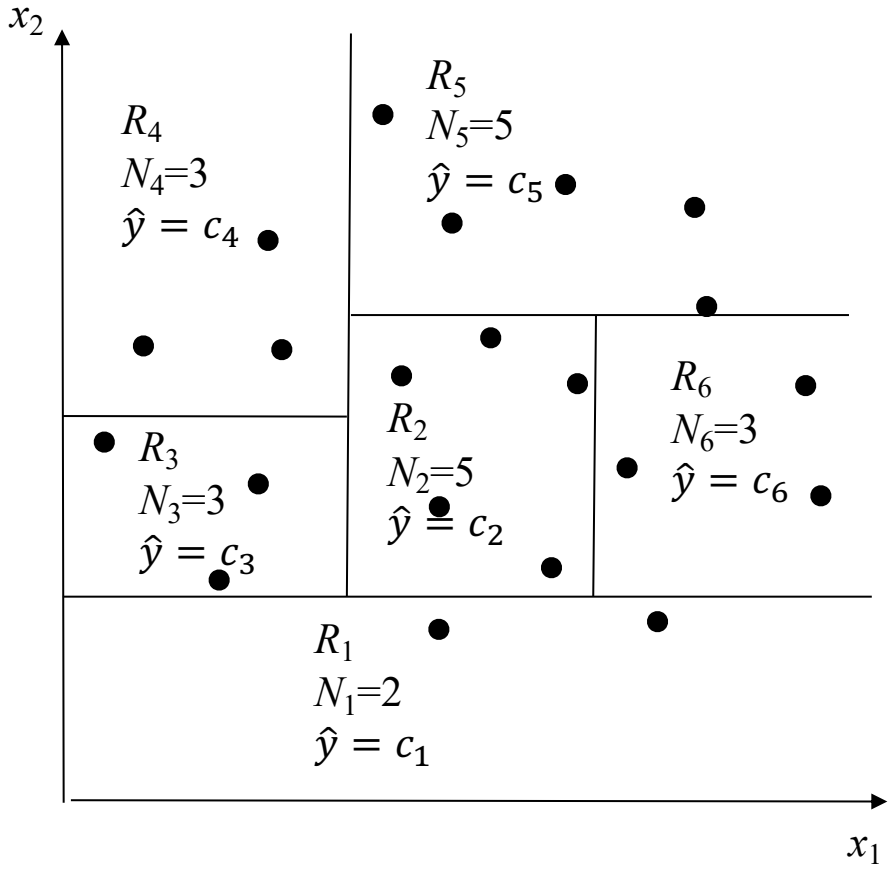
$I(\mathbf{x} \in R_m)$ = indicator function = $\begin{cases} 1: & \mathbf{x} \in R_m \\ 0: & \mathbf{x} \notin R_m \end{cases}$

c_m =constant predictor over R_m

$\boldsymbol{\theta}$ = all parameters and structure (M, splits in R_m's, c_m's, etc)

Note that for $\mathbf{x}_i \in R_j$, $g(\mathbf{x}_i;\boldsymbol{\theta}) = \sum_{m=1}^{M} c_m I(\mathbf{x}_i \in Rm) = c_j$

Illustration of the Notation for $M = 6$ Regions, $k = 2$ Predictors, and $n = 21$ Training Observations



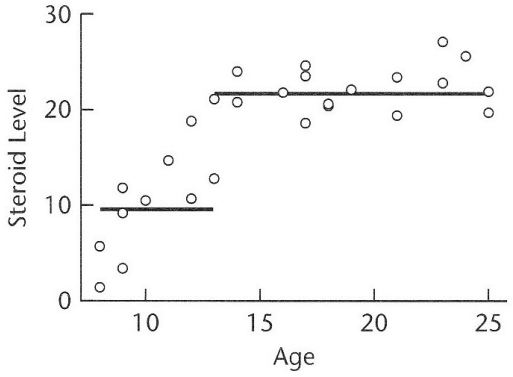
Discussion Points and Questions

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

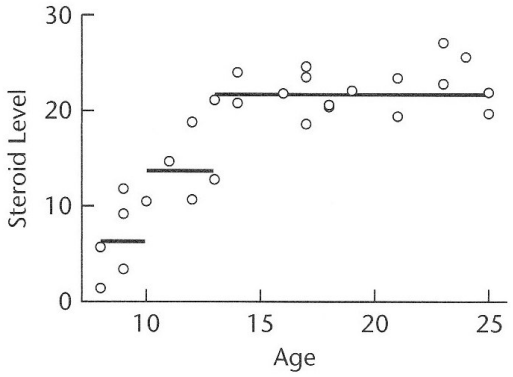
Fitting a Regression Tree

- A CART model is fit using an array of training data structured just like in regression (one response column and many predictor columns)
- Fitting the model entails growing the tree one node at a time (see next slide for an example)
  - 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
- After fitting a CART model, software spits out the final fitted tree, which can be used for prediction/interpretation

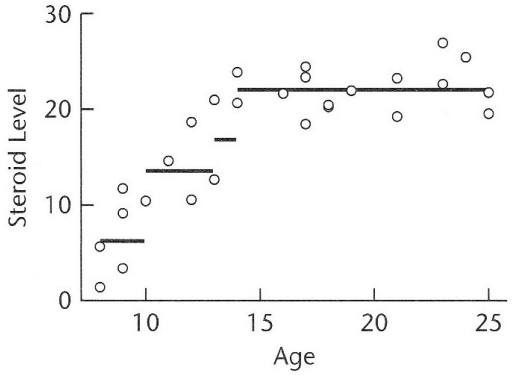
FIGURE 11.10
Growing the Regression Tree—Steroid Level Example.



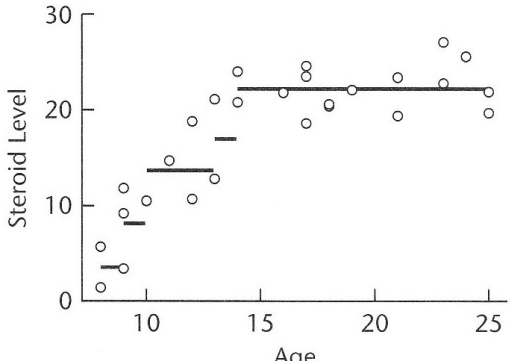
(a)

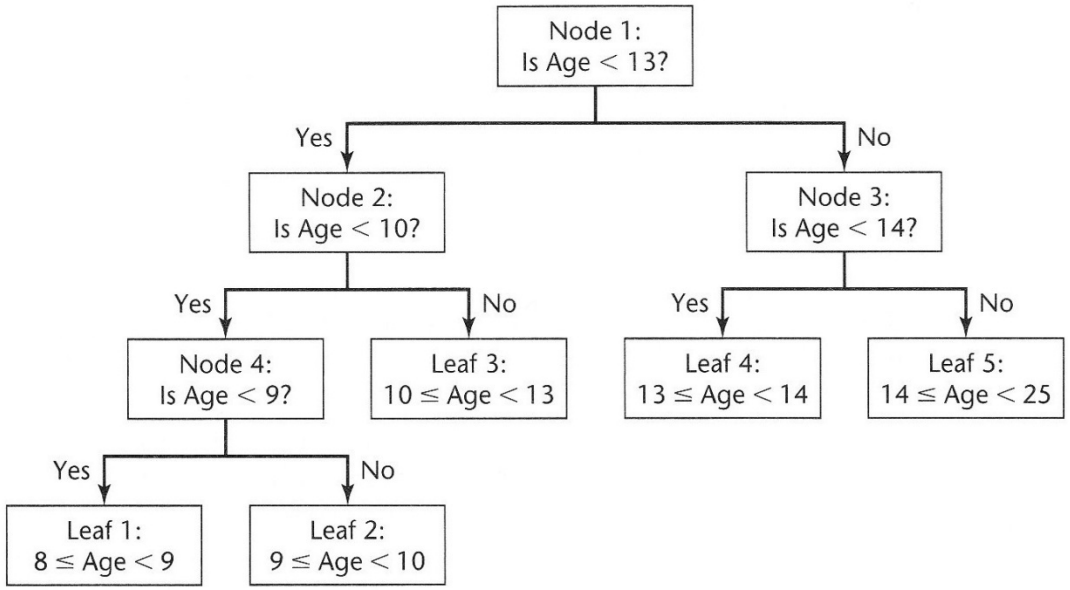


(b)



(c)





SSE is Calculated as Follows

For given set of splits:

$$\hat{c}_m = ave\{y_i | \mathbf{x}_i \in R_m\} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} y_i$$

$$N_m = \#\{\mathbf{x}_i \in R_m\} \quad = \text{"size" of } m^{\text{th}} \text{ terminal node (region)}$$

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

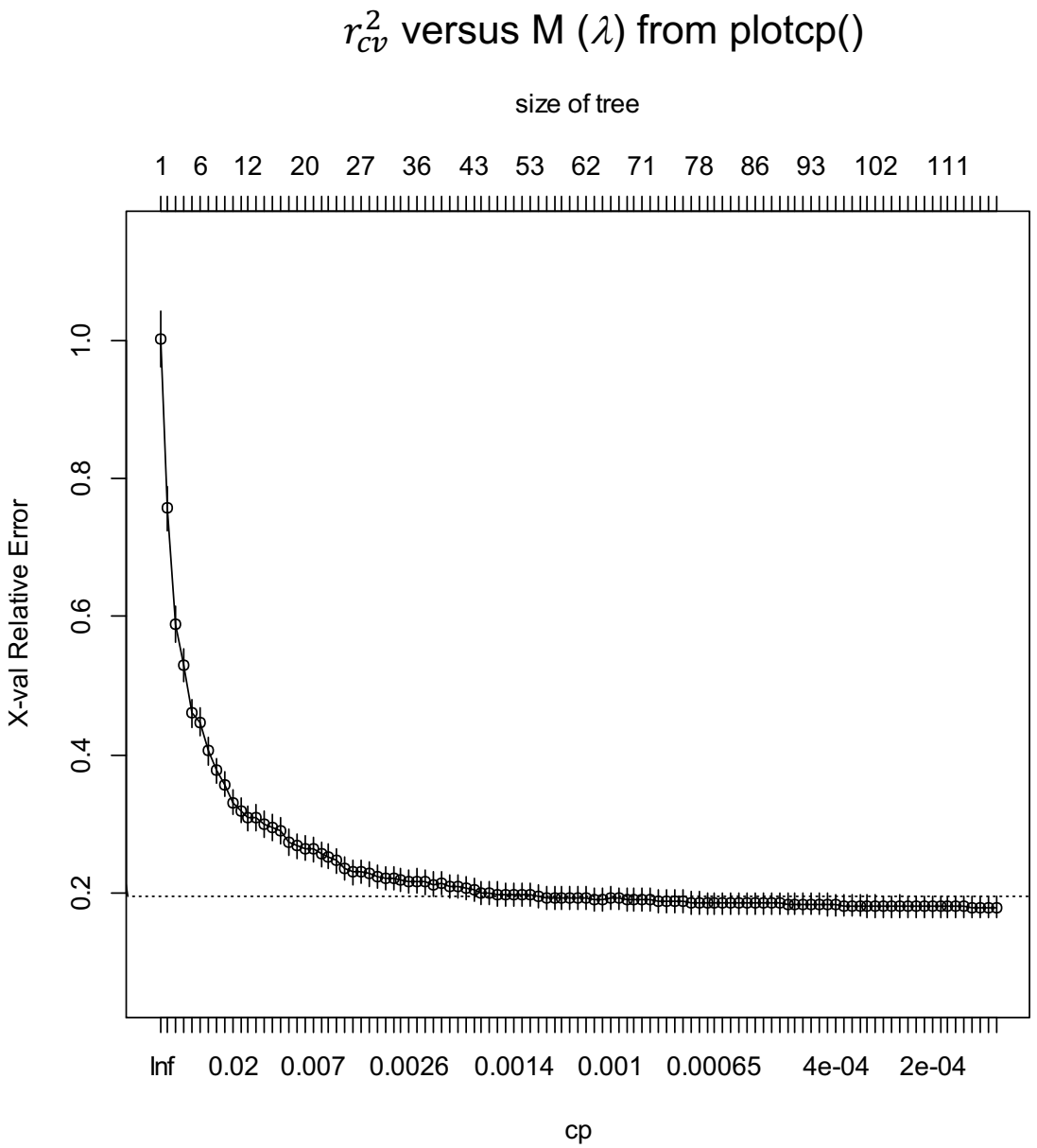
Note that for $\mathbf{x}_i \in R_j$, $\quad \hat{y}_i = g(\mathbf{x}_i; \hat{\boldsymbol{\theta}}) = \sum_{m=1}^{M} \hat{c}_m I(\mathbf{x}_i \in R_m) = \hat{c}_j$

Pruning

- Pruning a branch means that you collapse one of the internal nodes into a single terminal node
- Pruning the tree means that you prune a number of branches
- Pruning algorithms in software will usually optimally prune back a tree in a manner that minimizes $SSE + \lambda M$, where M and SSE are for the pruned tree. The best value for λ is determined via CV
- There is a nice computational trick ("weakest link pruning") that allows this optimal pruning to be done very fast. See HTF for further discussion.

Regression Tree Ex. (Concrete data)

```
#do not have to standardize or transform predictors to fit trees
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$cptable
#prune back to optimal size, according to plot of CV 1-r^2
CRT.tr1 <- prune(CRT.tr, cp=0.0015)  #approximately the best size pruned tree
CRT.tr1$variable.importance
CRT.tr1$cptable[nrow(CRT.tr1$cptable),] #shows training and CV 1-r^2, and other things
#prune and plot a little smaller tree than the optimal one, just for display
CRT.tr2 <- prune(CRT.tr, cp=0.007)  #bigger cp gives smaller size tree
CRT.tr2
par(cex=.9); plot(CRT.tr2, uniform=F); text(CRT.tr2, use.n = T); par(cex=1)
##
yhat<-predict(CRT.tr1); e<-CRT$Strength-yhat
c(1-var(e)/var(CRT$Strength), 1-CRT.tr1$cptable[nrow(CRT.tr1$cptable),3]) #check to
    see training r^2 agrees with what is in cptable
```

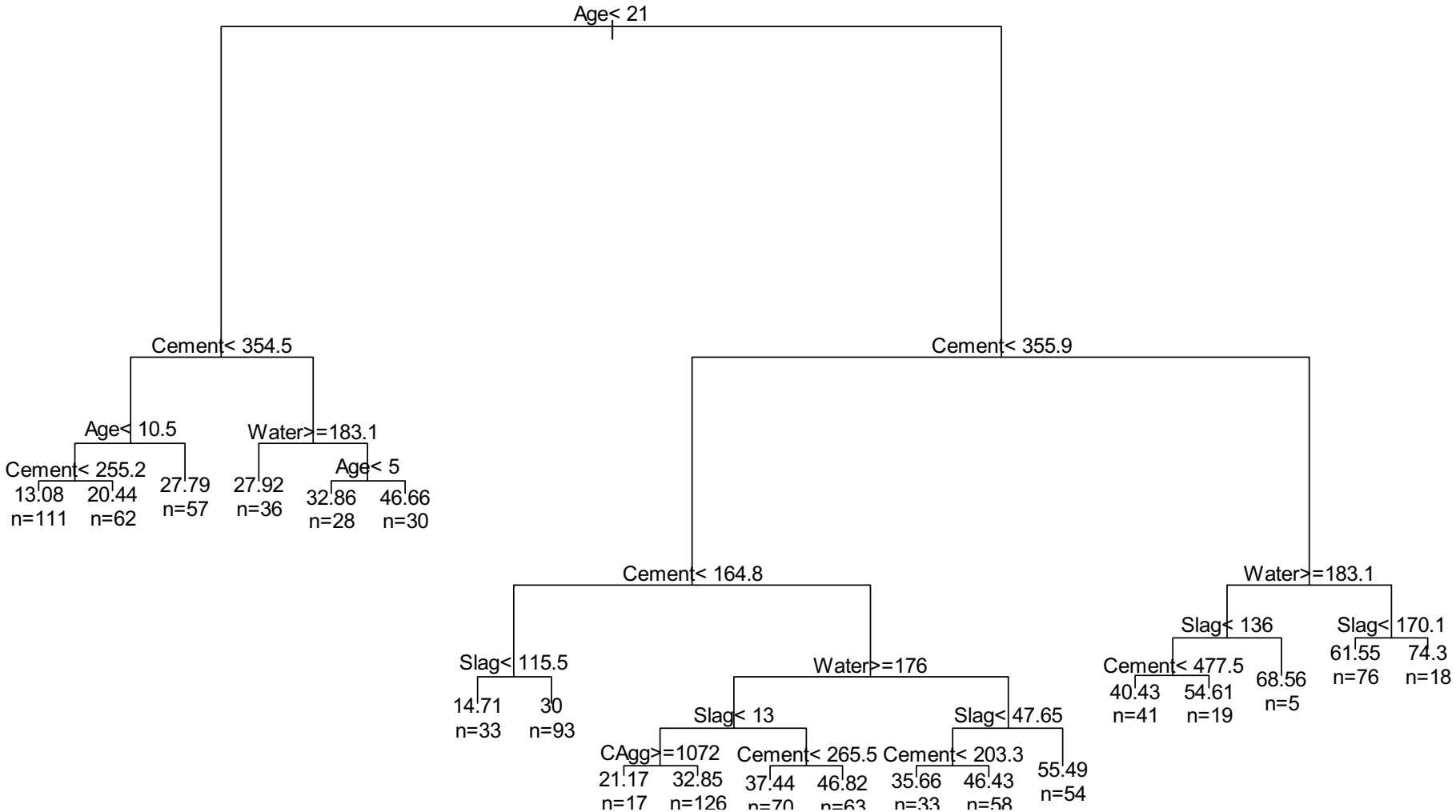


The horizontal line is the lowest 1-r_{cv}^2 plus one SE

A common strategy is to choose best size (M), or equivalently the best complexity parameter cp (λ), as the left-most value below the horizontal line (around size = 50 in this plot)

Or simply choosing the best size or cp as the one with minimum 1-r_{cv}^2 usually gives the model with the best predictive power (albeit larger than using the rule above)

the pruned tree
(but a little smaller than the best tree)



```
node), split, n, deviance, yval
      * denotes terminal node

 1) root 1030 287175.20000 35.81796
   2) Age< 21 324  49754.23000 23.54123
     4) Cement< 354.5 230  18387.25000 18.70622
       8) Age< 10.5 173   9063.38200 15.71393
        16) Cement< 255.25 111   2983.66300 13.07559 *
        17) Cement>=255.25 62   3923.75900 20.43742 *
       9) Age>=10.5 57   3073.50100 27.78807 *
     5) Cement>=354.5 94  12834.18000 35.37160
      10) Water>=183.05 36   3801.30200 27.91917 *
      11) Water< 183.05 58   5792.48500 39.99724
       22) Age< 5 28    913.24500 32.86000 *
       23) Age>=5 30   2121.67600 46.65867 *
   3) Age>=21 706 166177.90000 41.45204
     6) Cement< 355.95 547  88933.96000 36.95020
      12) Cement< 164.8 126  10420.69000 25.99714
       24) Slag< 115.5 33    484.49760 14.71152 *
       25) Slag>=115.5 93   4241.72600 30.00172 *
      13) Cement>=164.8 421  58873.04000 40.22831
       26) Water>=176 276  27953.72000 36.48475
         52) Slag< 13 143   7604.80700 31.46049
          104) CAgg>=1072.1 17    286.76640 21.17294 *
          105) CAgg< 1072.1 126   5276.12300 32.84849 *
         53) Slag>=13 133  12857.96000 41.88677
          106) Cement< 265.5 70   5072.34100 37.44357 *
          107) Cement>=265.5 63   4868.19400 46.82365 *
       27) Water< 176 145  19688.94000 47.35400
         54) Slag< 47.65 91   7681.11800 42.52582
          108) Cement< 203.35 33   1420.37200 35.65606
*
          109) Cement>=203.35 58   3817.25400 46.43448
*
         55) Slag>=47.65 54   6311.66400 55.49037 *
     7) Cement>=355.95 159  28020.16000 56.93950
      14) Water>=183.05 65   7903.75400 46.73969
       28) Slag< 136 60   5270.86800 44.92100
         56) Cement< 477.5 41   1731.99300 40.43098 *
         57) Cement>=477.5 19    928.64360 54.61000 *
       29) Slag>=136 5     52.92572 68.56400 *
      15) Water< 183.05 94   8677.97700 63.99255
       30) Slag< 170.1 76   5661.75300 61.55013 *
       31) Slag>=170.1 18    648.61360 74.30500 *
```

Discussion Points and Questions

- What is the best size tree for the concrete example?
- What is r_{cv}^2 for the best tree?
- Provide an interpretation of the model and which predictor variables are most important
- Do there appear to be any interactions between Age and any other predictor?
- To grow the initial tree larger (it is better to overgrow the initial tree, then prune it back) you can decrease minbucket and/or cp
- What is the variable importance measure for trees?
- For large trees that are difficult to interpret, ALE or PD plots can be used, just like for neural networks

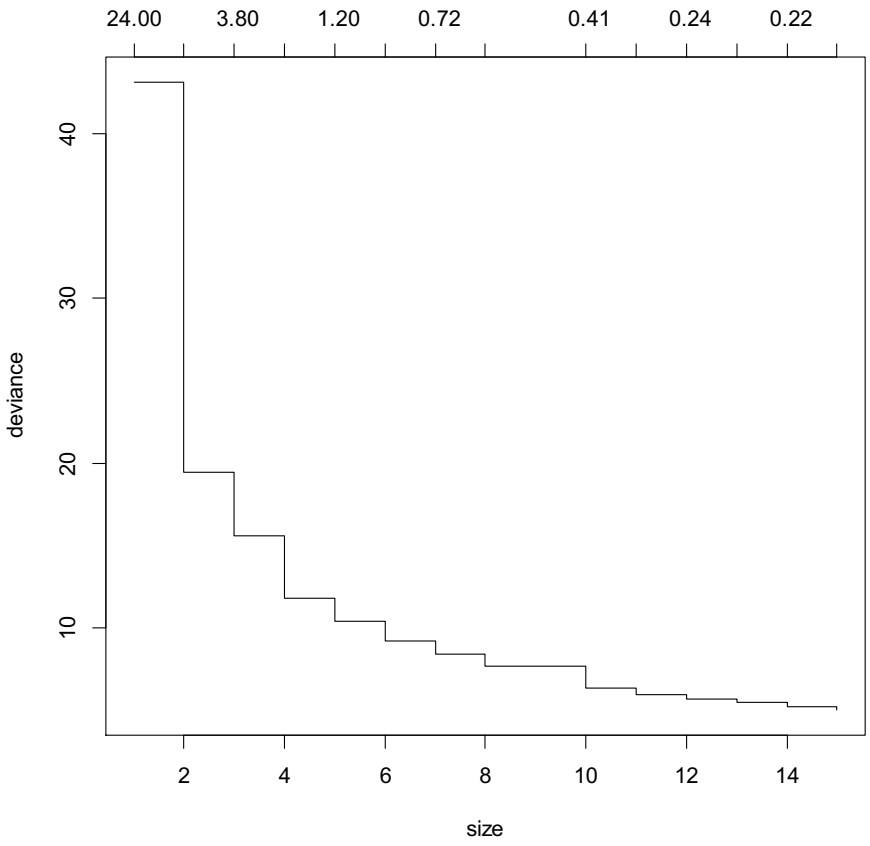
Numerical Assessment of Variable Importance

- 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 (if using uniform=F in plot.rpart)
- 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 $-2\log f(\mathbf{y},\boldsymbol{\theta})$. For a nonlinear regression model with normal errors, the deviance is the SSE (why?). Thus, you can get the deviance from the $frame object of the rpart output or from a textual print of the tree

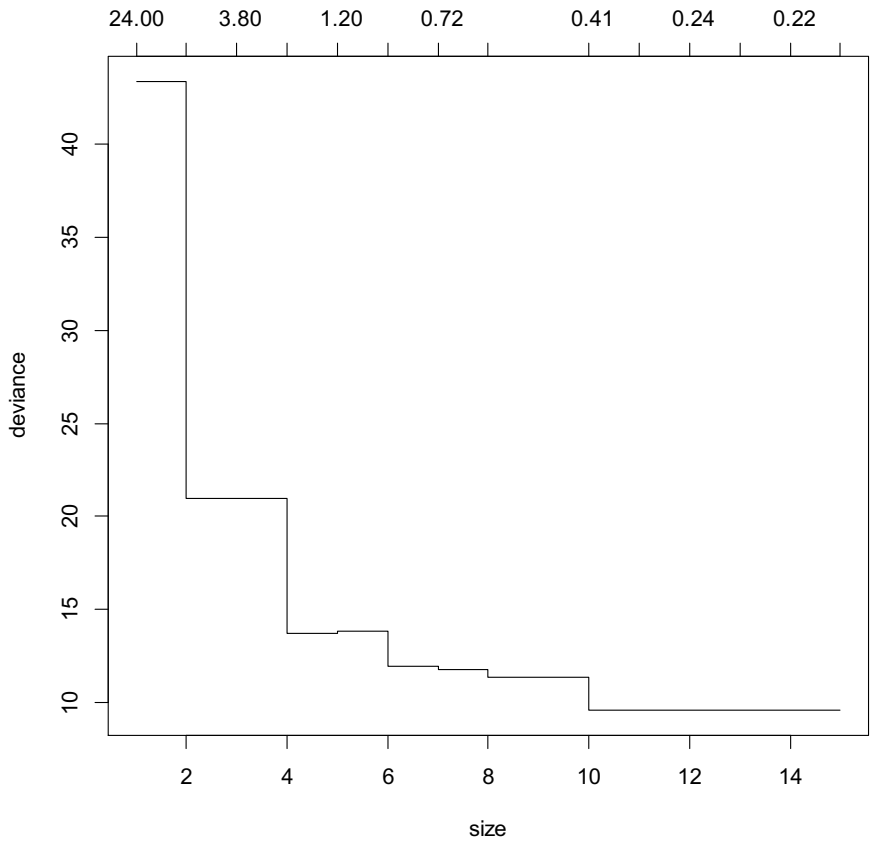
Regression Tree Ex. (cpus data) using the tree package, instead of rpart

```
#do not have to standardize or transform predictors to fit trees
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)
plot(cpus.tr,type="u"); text(cpus.tr,digits=2)  #type="p" plots proportional branch lengths
######now prune tree and plot deviance vs. complexity parameter
cpus.tr1<-prune.tree(cpus.tr)
plot(cpus.tr1)
######now plot CV deviance vs complexity parameter
plot(cv.tree(cpus.tr, , prune.tree))
######now find the final tree with the best value of complexity parameter
cpus.tr1<-prune.tree(cpus.tr, k=0.4) #can replace replace argument “k=0.4” by “best=11”
cpus.tr1
plot(cpus.tr1,type="u");text(cpus.tr1,digits=3)
```

deviance vs k (λ) from prune()



deviance vs k (λ) from cv.tree()



Discussion Points and Questions

- 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
- The "deviance" measure that is plotted versus tree size is $-2\log f(\mathbf{y},\boldsymbol{\theta})$. For a nonlinear regression model with normal errors, the deviance is the SSE (why?). Thus, you can get the CV SSE from the $dev object of the cv.tree output

Classification Trees – Overview

- Fitting and using classification trees with a K-category response is similar to fitting and using regression trees.
- For classification trees, we model $p_k(\mathbf{x}) = Pr\{Y = k \mid \mathbf{x}\}$ (k=1,2,. . .,K) as constant over each region
- Compare to regression trees, for which we model $g(\mathbf{x}; \boldsymbol{\theta}) = E[Y \mid \mathbf{x}]$ 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

Classification Trees – Some Details

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

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} I\left(y_i = k\right)$$ (class-k sample fraction in region R_m)

$$k_m = arg\,max_k \left\{\hat{p}_{m,k}\right\}$$ (most common class in region R_m)

- Some common impurity measures:

Misclassification error: $\sum_{m=1}^{M} \sum_{\mathbf{x}_i \in R_m} I\left(y_i \neq k_m\right) = \sum_{m=1}^{M} N_m \left(1 - \hat{p}_{m,k_m}\right)$

Gini index: $\sum_{m=1}^{M} N_m \sum_{k=1}^{K} \hat{p}_{m,k}\left(1 - \hat{p}_{m,k}\right)$

deviance: $-2 \sum_{m=1}^{M} N_m \sum_{k=1}^{K} \hat{p}_{m,k} log\left(\hat{p}_{m,k}\right)$ (– 2 log-likelihood)

Example Illustrating the Notation

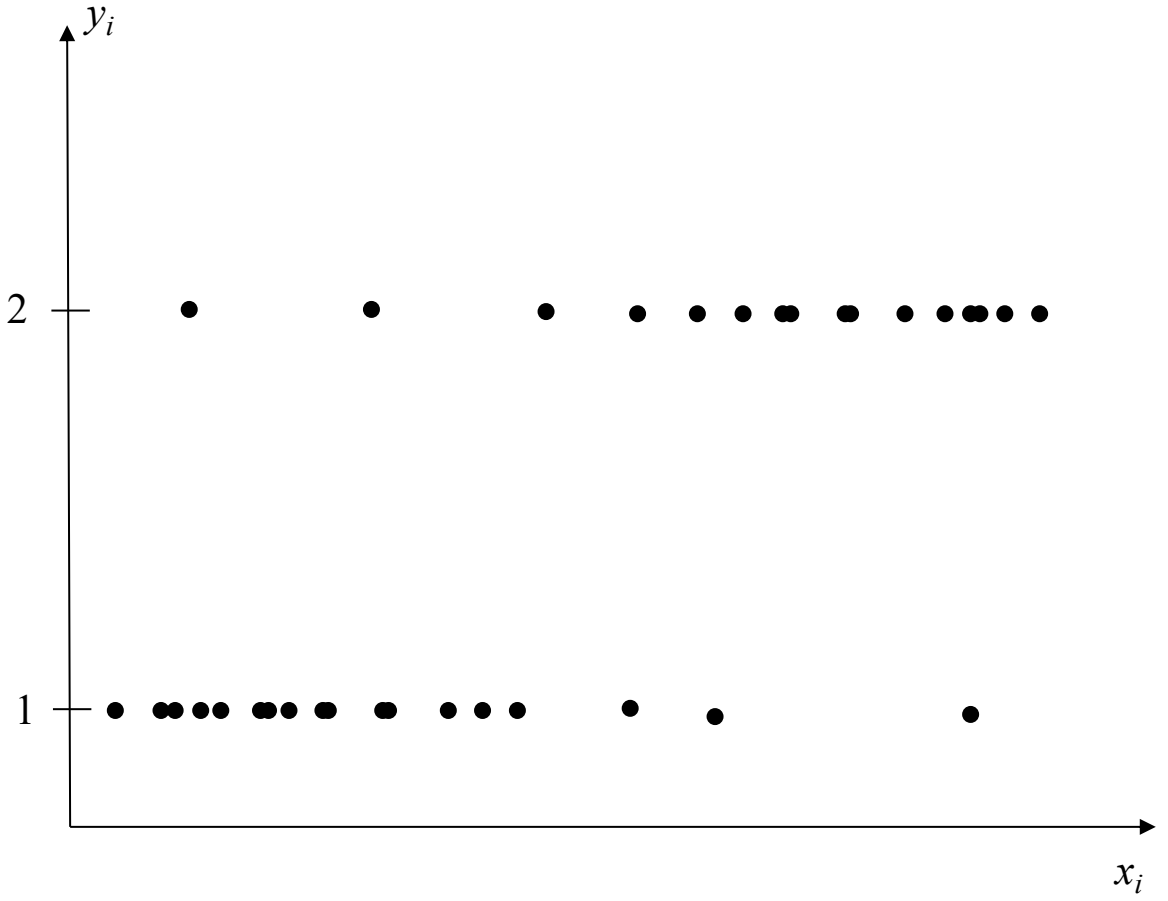
- Suppose you have $K = 4$ classes, and the predictors for $N_m = 100$ training cases fall into a particular region R_m. For those 100 cases, suppose we have the following breakdown of the number of cases with response value that fell into the four categories:

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 $k = 1, 2, 3, 4$?
- What is k_m?

$K = 2$ Class Example Illustrating Notation and Splitting Based on Impurity

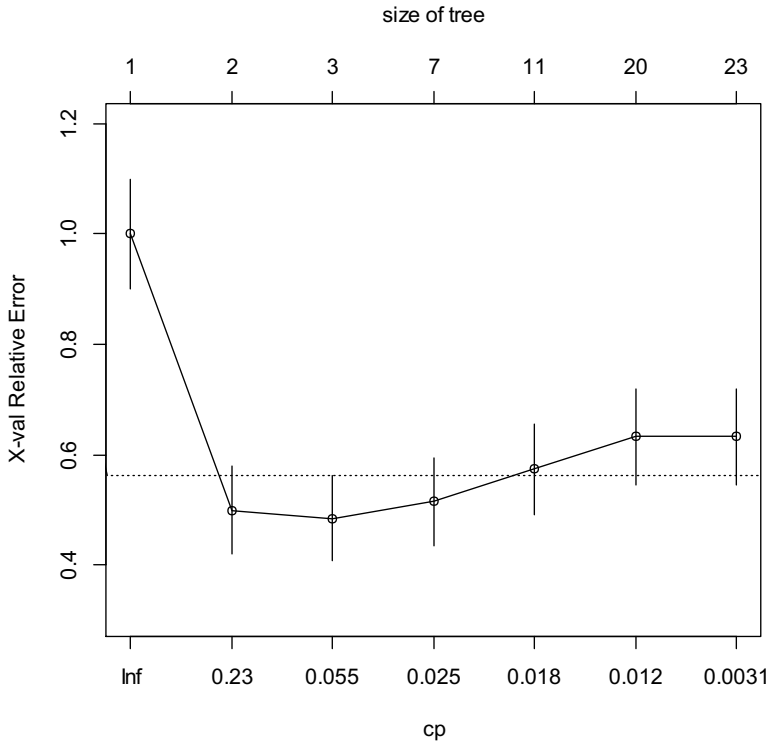
For the following training data (with a single predictor), where would the first split that minimizes the misclassification rate be, and what would the $\hat{p}_{m,k}$ and k_m be?

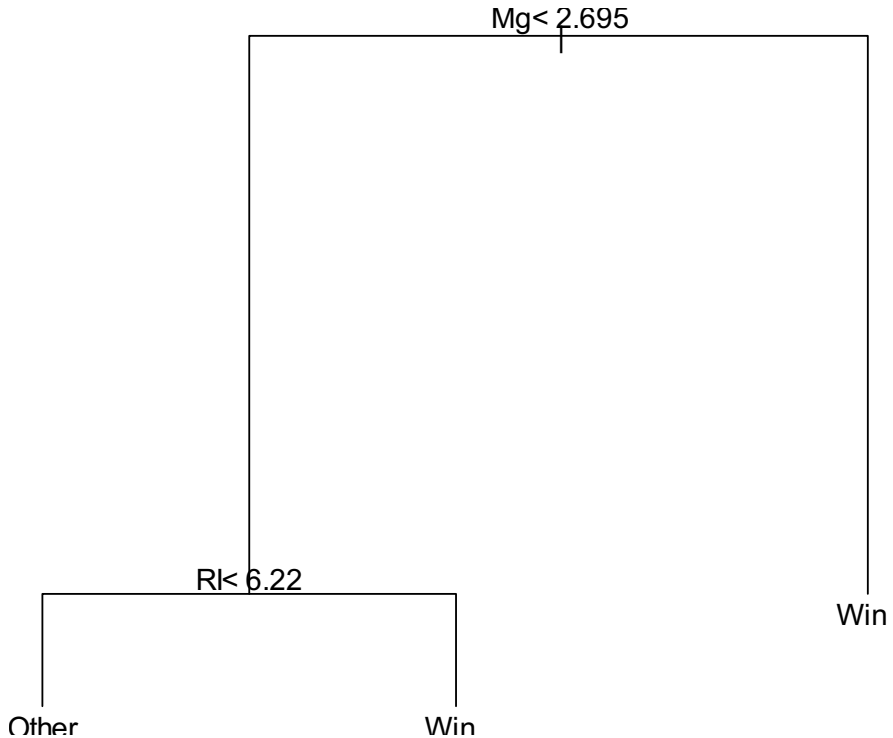


Classification Tree Ex. (fgl data)

```
library(rpart)
control <- rpart.control(minbucket = 1, cp = 0.001, maxsurrogate = 0, usesurrogate = 0,
    xval = 10)
FGL.tr <- rpart(type_bin~., FGL[,c(1:9,11)], method = "class", control = control)
plotcp(FGL.tr)
printcp(FGL.tr)  #same info in FGL.tr$cptable
#prune to optimal size
FGL.tr1 <- prune(FGL.tr, cp=0.055)  #approximately the cp corresponding to the optimal
    size
FGL.tr1
par(cex=1); plot(FGL.tr1, uniform=F); text(FGL.tr1, use.n = F); par(cex=1)
FGL.tr1$variable.importance
FGL.tr1$cptable[nrow(FGL.tr1$cptable),]
#calculate training and CV misclass rates
FGL.tr1$cptable[nrow(FGL.tr1$cptable),c(3,4)]*min(table(FGL$type_bin)/nrow(FGL))
    #training and cv misclass rates
yhat<-predict(FGL.tr1, type="class")
sum(yhat != FGL$type_bin)/nrow(FGL) #check the training misclass rate
```

deviance r_{cv}^2 versus M (λ) from plotcp()





```
> FGL.tr1
n= 214

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 214 68 Win (0.3177570 0.6822430)
  2) Mg< 2.695 61 13 Other (0.7868852 0.2131148)
    4) RI< 6.22 54  6 Other (0.8888889 0.1111111) *
    5) RI>=6.22 7  0 Win (0.0000000 1.0000000) *
  3) Mg>=2.695 153 20 Win (0.1307190 0.8692810) *

> par(cex=1); plot(FGL.tr1, uniform=F); text(FGL.tr1, use.n = F); par(cex=1)

> FGL.tr1$variable.importance
      Mg       RI
37.55479  9.79235

> FGL.tr1$cptable[nrow(FGL.tr1$cptable),]
        CP     nsplit  rel error     xerror       xstd
0.05500000 2.00000000 0.38235294 0.48529412 0.07769274

> FGL.tr1$cptable[nrow(FGL.tr1$cptable),][c(3,4)]*min(table(FGL$type_bin)/nrow(FGL))
       #training and cv misclass rates
rel error    xerror
0.1214953 0.1401869
> yhat<-predict(FGL.tr1, type="class")
> sum(yhat != FGL$type_bin)/nrow(FGL)
[1] 0.1214953
```

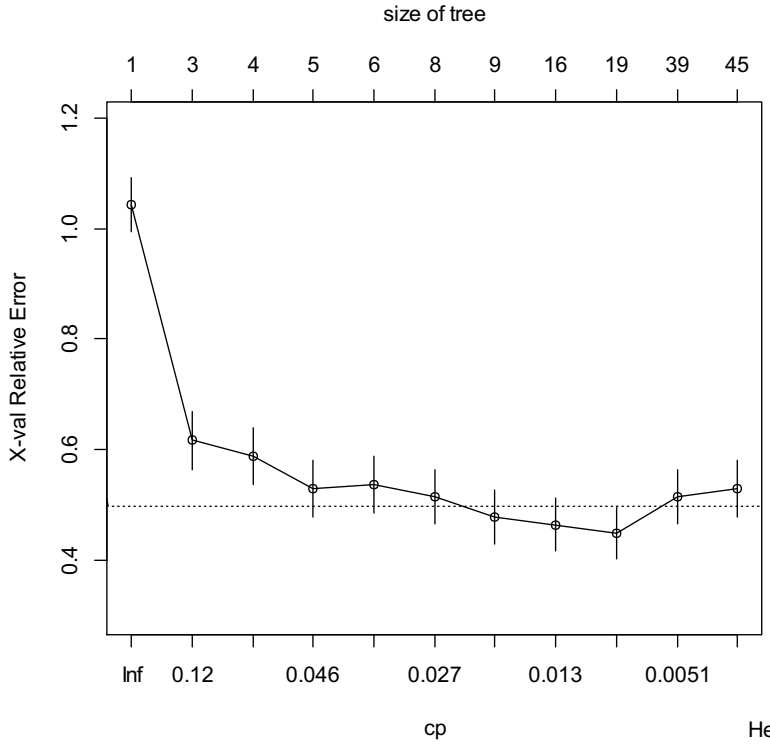
Discussion Points and Questions

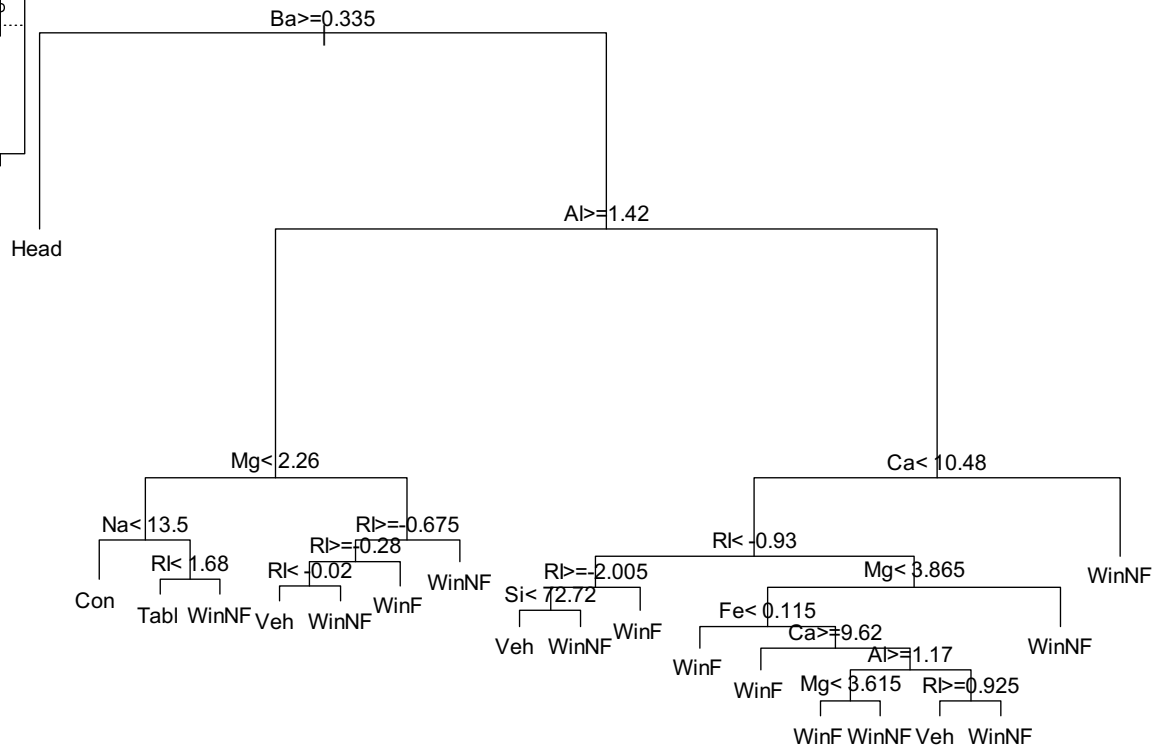
- What is the best tree size for the FGL data?
- Which predictors appear to be the most important, and what are their effect(s)?
- If you want a summary measure of the predictive quality of the tree model, what would it be?
- If you wanted to decide whether the neural network is better than the tree for predicting glass type, how would you do this?
- How can you tell what impurity measure rpart used to fit the model?

Same but for the original 6-category response

```
library(rpart)
control <- rpart.control(minbucket = 1, cp = 0.00001, maxsurrogate = 0, usesurrogate = 0, xval = 10)
FGL.tr <- rpart(type~.,FGL[,c(1:10)], method = "class", control = control)
plotcp(FGL.tr)
printcp(FGL.tr)  #same info in FGL.tr$cptable
#prune to optimal size
FGL.tr1 <- prune(FGL.tr, cp=0.01)  #approximately the cp corresponding to the optimal size
FGL.tr1
par(cex=1); plot(FGL.tr1, uniform=F); text(FGL.tr1, use.n = F); par(cex=1)
FGL.tr1$variable.importance
FGL.tr1$cptable[nrow(FGL.tr1$cptable),]
#see what the predicted class probabilities are
yhat<-predict(FGL.tr1, type="prob")
yhat[1:20,]
```

The Best 6-Class Tree for the FGL Data





```
> FGL.tr1$variable.importance
       Ba        RI        Mg        Al        Ca        Na        Fe        Si
26.044912 22.861814 20.020468 19.585776 10.768054  6.116667  3.743814  2.500000
> FGL.tr1$cptable[nrow(FGL.tr1$cptable),]
         CP      nsplit   rel error      xerror        xstd
 0.01000000 18.00000000  0.19565217  0.44927536  0.04808744
> #calculate training misclass rate
> yhat<-predict(FGL.tr1, type="prob")
> yhat[1:20,]
      Con       Head       Tabl        Veh       WinF      WinNF
 [1,]   0 0.00000000 0.00000000 0.12500000 0.1250000 0.75000000
 [2,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
 [3,]   0 0.00000000 0.00000000 0.02500000 0.0750000 0.90000000
 [4,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
 [5,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
 [6,]   0 0.00000000 0.00000000 0.02500000 0.0750000 0.90000000
 [7,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
 [8,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
 [9,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[10,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[11,]   0 0.00000000 0.00000000 0.02500000 0.0750000 0.90000000
[12,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[13,]   0 0.00000000 0.25000000 0.00000000 0.7500000 0.00000000
[14,]   0 0.00000000 0.00000000 0.00000000 1.0000000 0.00000000
[15,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[16,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[17,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[18,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[19,]   0 0.01754386 0.01754386 0.05263158 0.8596491 0.05263158
[20,]   0 0.00000000 0.00000000 0.00000000 1.0000000 0.00000000
```

Discussion Points and Questions

- You choose the best size the same way – choosing the size with the lowest CV error measure (the best M was about 15—20)
- Which predictors appear to be the most important?
- If you want a summary measure of the predictive quality for the 6-class tree model, what would it be?

Pros and Cons of Neural Networks

- Pros:
  - very flexible; with enough nodes, can model almost any nonlinear relationship
  - can efficiently model linear behavior if the relationship is truly linear
  - often very good predictive power
  - a lot of ways to customize the architecture to tackle specific modeling problems (recurrent NNs; convolutional NNs; autoencoders; other “deep learning” architectures ...)
- Cons:
  - model fitting can be unstable and sensitive to initial guesses
  - for very large data sets, model fitting can be very slow relative to some methods like trees and linear models, which makes CV very computationally expensive
  - overfitting (but can avoid by using CV to choose λ)
  - sensitive to user-chosen "tuning parameters” (but can use CV to choose them wisely)
  - poor interpretability, but this can be improved with ALE or PD plots

Pros and Cons of CART 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
  - for quantitative responses or binary categorical responses, easy to handle and interpret nominal categorical predictors with many categories (even easier to handle ordinal categorical predictors)
  - 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

Handling Nominal Categorical Predictors

- Recall that most other methods handle a nominal categorical predictor by creating $c - 1$ 0/1 numerical predictors, where c is the number of categories
- Trees have a much better built-in way to handle nominal categorical predictors, even with very large c:
  - Note that splitting a region of the tree into two subregions based on a categorical predictor means finding a partition of the c categories into two groups of categories, such that one group goes out the left branch of the node and the other group goes out the right branch
  - There are $2^{c-1} - 1$ ways to partition the c categories into two (nonempty) groups, which sounds like a lot of partitions to check
  - Fortunately, we can exploit the fact that the optimal partition must be one of only $c - 1$ possible partitions that are consistent with ordering the categories according to (*i*) the within-category average response in that region (for regression) or (*ii*) the within-category sample fraction of responses that are 1 in that region (0/1 binary classification)
  - Thus, we first order the categories according to the response observations in that region, and then we only have to check which of the $c - 1$ split points gives the smallest loss
- A fitted tree with nominal categorical predictors has really nice interpretative benefits related to segmenting categories into like groups, based on their effect on the response

Handling Missing Predictor Values

- For many supervised learning methods, if you are missing the value of one or more predictors in a row, you must either:
  - Throw out the entire row (which could result in throwing out most of your data if there are many predictors), or
  - Fill in (aka impute) missing values using anything from the average of that predictor (very simple) to some predicted value for that predictor from a regression on all other predictors (very complicated)
- Trees have nicer ways to handle missing values for predictors:
  1) **Probabilistic splits** (for prediction, not training). Suppose a case to be predicted is missing the predictor value corresponding to a node in the tree. Instead of sending that case to only one child node, we split it across all child nodes with weights proportional to the fraction of observations not missing that predictor that fall into each child node
  2) **Surrogate splits** (for prediction and training). During training, after finding the best variable and split point for a node, find a different surrogate variable and split point that best mimics the original variable and split point (in terms of how it partitions the observations in that node into two subgroups). Then find the next best surrogate variable and split point, and so on. During prediction, if a new case is missing the predictor value for a node, use the first surrogate variable that is available
  3) **Treat "missing" as a new category** (for prediction and training). This is trivial for categorical predictors. For numerical predictors assign a very large value to "missing"

A Good Way to Do Preliminary Variable Selection

- Motivation: Including irrelevant predictors in a model reduces the predictive power and interpretability. If we start with a data set with very large k (and large n) how can we screen out irrelevant predictors?
- For linear $\mathbf{x} \rightarrow Y$ relationships, can use stepwise or lasso. For nonlinear relationships, can use the following:
  - Begin with forward stepwise for a linear/logistic regression model to select a subset S_1 of predictors, and denote the residuals by e
  - Fit a tree with e as the response and all k predictors included, to select a second subset S_2 of important predictors
  - Fit any desired model with $S_1 \cup S_2$ as the selected predictors
- Beware: Unless $n \gg p$, initially selecting a subset of variables using stepwise will cause overfitting when fitting the final model, even if cross-validation is used properly in the final stage.

Comparing Trees with Other Models

- Use CV to compare trees with any other model
- In the previous regression tree example (concrete data), the best value of complexity parameter was λ = 2800, which translated to M = 13 terminal nodes.
- To compare a regression tree with a neural network model we would:
  - Form a random CV partition (e.g. using the CVInd function)
  - Compute the CV SSE for a neural network model with 10 hidden layer nodes and λ = 0.05 (which CV earlier said was roughly the best value)
  - Compute the CV SSE for a regression tree model with either λ = 2800 or M = 13 **using the same partition** (for each tree fit in CV, overgrow and prune to M = 13 just like for the original tree)
  - Repeat the previous 3 steps as many times as you can, averaging the results, and select the best model as the one with the lower average CV SSE

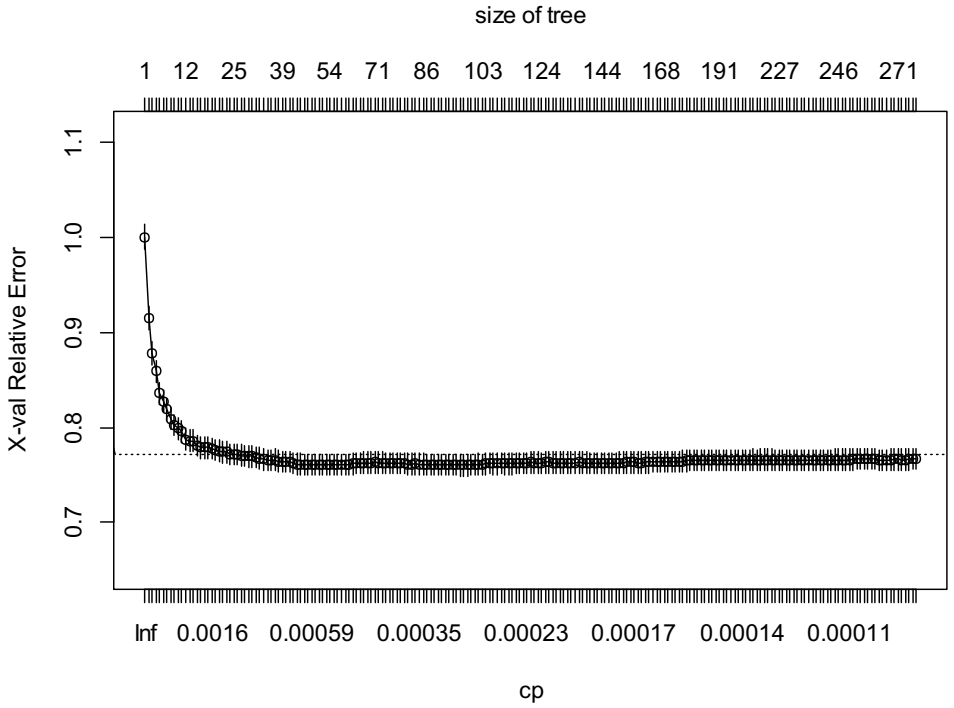
Return to the Income Data Example

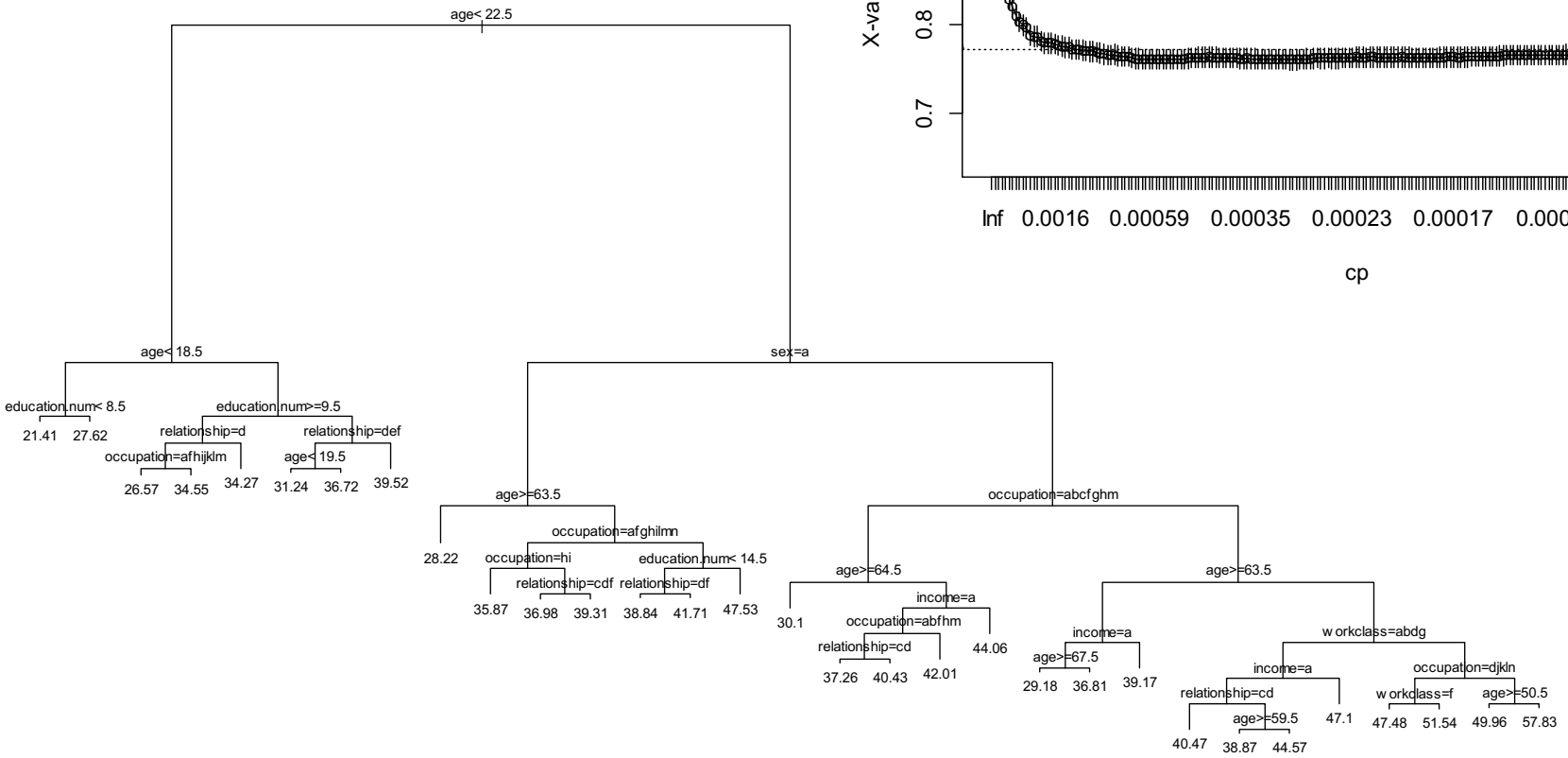
- Reconsider the data in adult_train.csv
- Before, we fit a neural network model for regression, predicting the number of hours per week worked. And we also fit a neural network for classification, predicting the binary income categorization (<= 50k vs. > 50k)
- Here, we will fit similar regression and classification models, but using trees instead of neural networks

Try a Regression Tree

```
library(rpart)
XX<-read.table("adult_train.csv",sep=",",header=TRUE,strip.white=TRUE,na.strings="?")
XX<-na.omit(XX)
INCOME<-XX  #there is no need to standardize the predictors with trees (why not)
control <- rpart.control(minbucket = 20, cp = 0.0001, maxsurrogate = 0, usesurrogate = 0,
      xval = 10)
INC.tr <- rpart(hours.per.week ~ ., INCOME[,-c(3,4)], method = "anova", control = control)
plotcp(INC.tr)
printcp(INC.tr)
#prune back to optimal size, according to plot of CV r^2
INC.tr1 <- prune(INC.tr, cp=0.001)  #approximately the cp corresponding to the best size
INC.tr1
par(cex=.9); plot(INC.tr1, uniform=F); text(INC.tr1, use.n = F); par(cex=1)
INC.tr1$variable.importance
INC.tr1$cptable[nrow(INC.tr1$cptable),] #shows training and CV r^2, and other things
```

The Best-sized Regression Tree for INCOME Data





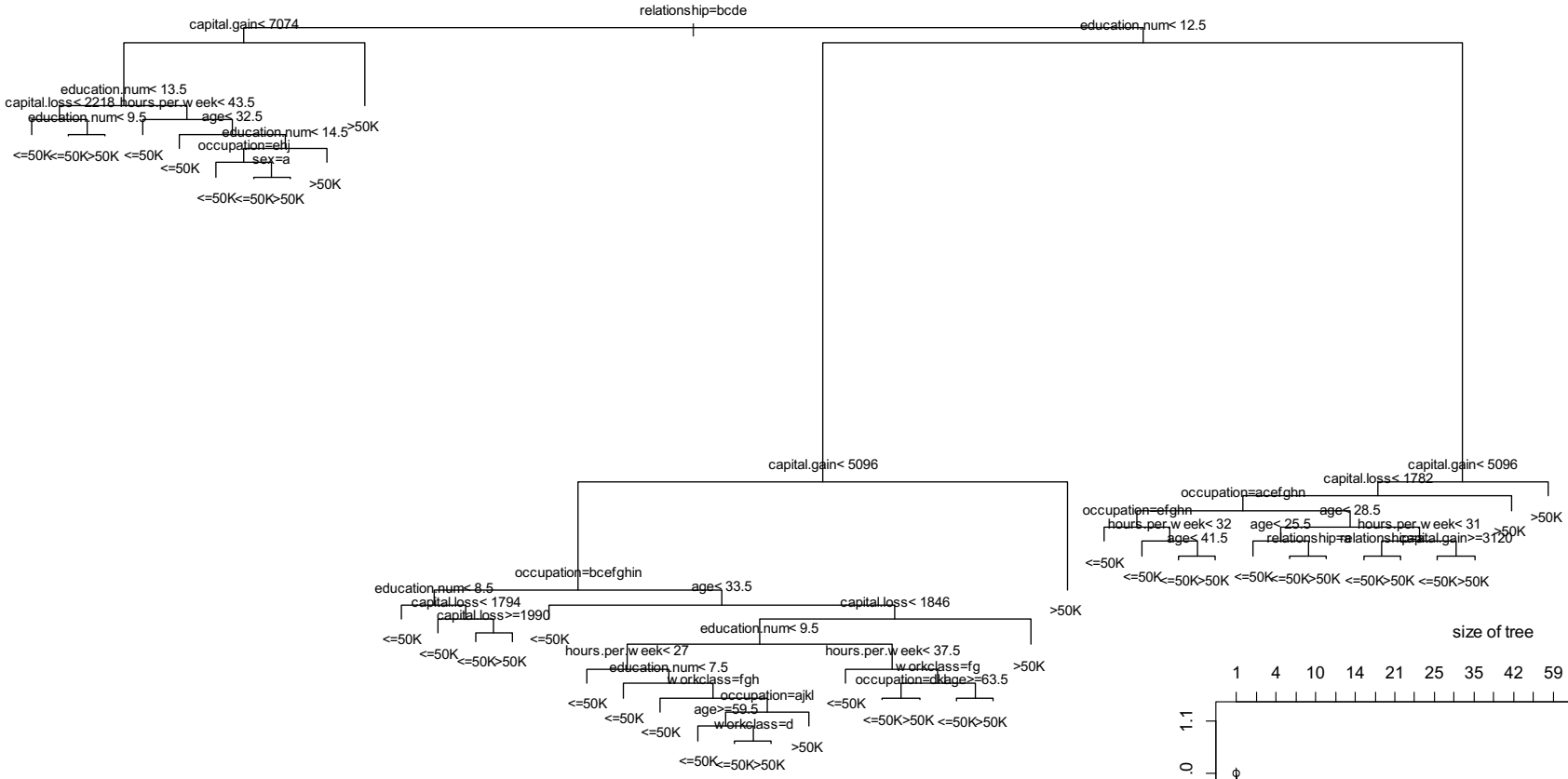
Discussion Points and Questions

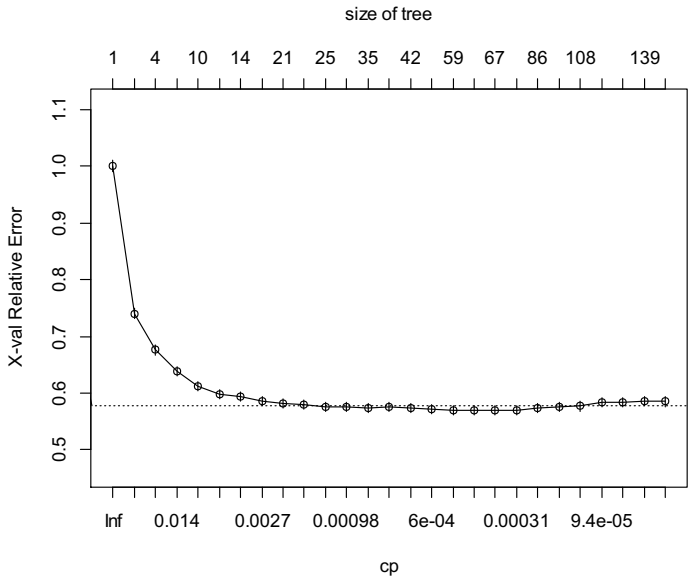
- Relative to the CRT example, would you increase or decrease minbucket?
- As always, you should deliberately overgrow the tree and then prune it back. How do know if you have overgrown the tree enough?
- Comparing the tree to the neural network
  - Which was faster to fit?
  - Which had better predictive quality, and how can you tell?
  - Which was easier to interpret (without using ALE or PD plots)?
- How about for comparing a tree to a linear regression?

Try a Classification Tree

```
library(rpart)
XX<-read.table("adult_train.csv",sep=",",header=TRUE,strip.white=TRUE,na.strings="?")
XX<-na.omit(XX)
INCOME<-XX  #there is no need to standardize the predictors with trees (why not)
control <- rpart.control(minbucket = 20, cp = 0.00001, maxsurrogate = 0, usesurrogate =
    0, xval = 10)
INC.tr <- rpart(income ~ ., INCOME[,-c(3,4)], method = "class", control = control)
plotcp(INC.tr)
printcp(INC.tr)
#prune back to optimal size, according to plot of CV r^2
INC.tr1 <- prune(INC.tr, cp=0.0007)  #approximately the cp corresponding to the best
    size
INC.tr1
par(cex=.7); plot(INC.tr1, uniform=F); text(INC.tr1, use.n = F); par(cex=1)
INC.tr1$variable.importance
INC.tr1$cptable[nrow(INC.tr1$cptable),]
INC.tr1$cptable[nrow(INC.tr1$cptable),][c(3,4)]*min(table(XX$income)/nrow(XX))
    #training and cv misclass rates
yhat<-predict(INC.tr1, type="class"); sum(yhat != XX$income)/nrow(XX) #check training
    misclass rate
```

The Best-sized Class Tree for INCOME Data





Why does the first split have such a short branch length?