Machine Learning for Microeconometrics
Part 4: More Prediction Methods

A. Colin Cameron
Univ.of California - Davis

April 2024

Course Outline

- **1.** Variable selection and cross validation
- **2.** Shrinkage methods
  - ridge, lasso, elastic net
- **3.** ML for causal inference using lasso
  - OLS with many controls, IV with many instruments
- **Part 4: Other methods for prediction**
  - **nonparametric regression, principal components, splines**
  - **neural networks**
  - **regression trees, random forests, bagging, boosting**
- **5.** More ML for causal inference
  - ATE with heterogeneous effects and many controls.
- **6.** Classification and unsupervised learning
  - classification (categorical y) and unsupervised learning (no y).

Introduction

- Basics used OLS regression
  - though with potentially rich set of regressors with interactions
  - and regularized OLS using lasso, ridge and elastic net.
- Now consider various other flexible methods for regress y on $\mathbf{x}$
  - some existed before ML became popular
    - ★ nonparametric and semiparametric regression
    - ★ principal components
    - ★ basis functions: polynomials, splines, sieves
  - some are more recent
    - ★ neural networks
    - ★ regression trees and random forests
  - in this part consider supervised learning (y and $\mathbf{x}$) for continuous y.
- Based on the two books by Hastie and Tibshirani and coauthors and Geron (2022).

Flexible methods

- These slides present many methods
- Which method is best (or close to best) varies with the application
  - e.g. deep learning (neural nets) works very well for Google Translate
  - all require setting tuning parameters which may not be straightforward.
- In forecasting competitions the best forecasts are **ensembles**
  - a weighted average of the forecasts obtained by several different methods
  - the weights can be obtained by OLS regression in a test sample
    - ★ e.g. given three forecast methods minimize w.r.t. τ_1 and τ_2 $\sum_{i=1}^{n}\{y_i - \tau_1\widehat{y}_i^{(1)} - \tau_2\widehat{y}_i^{(2)} - (1-\tau_1-\tau_2)\widehat{y}_i^{(3)}\}^2$.

Implementation

- Stata has built-in commands for
  - lasso and ridge (already covered)
  - nonparametric regression, principal components and basis functions.
- For other models (neural network, random forests, ...)
  - R has many commands
  - Python is viewed as best
  - Stata has add-ons that often are front-ends to R or Python so require also installing R or Python
    - ★ `crtrees` does trees and random forests directly in Stata
    - ★ `srtree` is a wrapper for R functions `tree()`, `randomForest()`, and `gbm()`
    - ★ `pylearn` does trees, random forests and neural nets directly in Stata and requires installation of Python and the Python scikit-learn library.

Overview

1. Nonparametric and semiparametric regression
2. Dimension reduction (principal components)
3. Flexible regression (polynomials, splines, sieves)
4. Neural networks
5. Regression trees and random forests
    1. Regression trees
    2. Bagging
    3. Random forests
    4. Boosting
6. Prediction Example
7. Prediction for Economics

1. Nonparametric and Semiparametric Regression

- We want a **flexible model** for $f(\mathbf{x}_0) = E[y|\mathbf{x} = \mathbf{x}_0]$ evaluated at a range of values $\mathbf{x}_0$.
- Use $\bar{y}|\mathbf{x} = \mathbf{x}_0$ if there are many observations of y at each value of $\mathbf{x}_0$.
- In practice there are few values of y at each value of $\mathbf{x}_0$, so bring into the average values of y at values of $\mathbf{x}$ near to $\mathbf{x}_0$
  - **local constant** kernel-weighted regression
  - **local linear** kernel-weighted regression
  - **k-nearest neighbors**
  - **lowess**.
- All depend on a **tuning parameter** that trades off bias and variance
  - like choice of bin width for a histogram
  - usually minimize MSE using leave-one-out cross validation.
- Problem: **curse of dimensionality** if many $\mathbf{x}$
- Solutions: semiparametric (old school) or ML methods (new school).

1.1 Nonparametric kernel-weighted regression

- Start with scalar x.
- Local-weighted average

$$\widehat{f}(x_0) = \sum_{i=1}^{N} w(x_i, x_0, h) y_i, \ \sum_{i=1}^{N} w_i = 1.$$

- **Kernel-weighted average** uses kernel weights

$$w(x_i, x_0, h) = K\left(\frac{x_i - x_0}{h}\right) / \left(\sum_{j=1}^{N} K\left(\frac{x_j - x_0}{h}\right)\right)$$

- $K(z)$ is a kernel function with $\int K(z)dz = 1$, $K(-z) = K(z)$, $K(z) \longrightarrow 0$ as $z \longrightarrow \infty$.
  - Most kernels have $K(z) = 0$ for $|z| > 1$ (epanechnikov, uniform, triangular)
  - Others are continuous on $(-\infty, \infty)$ (gaussian).
- Key is the bandwidth h (the tuning parameter)
  - chosen by leave-one-out cross validation (minimizes MSE).

Local constant regression

- Recall: a sample mean of y = OLS of y on an intercept.
- Similarly: a weighted sample mean of y = weighted OLS of y on an intercept.
- Weighted regression on a constant is the estimator that minimizes the weighted sum of squared residuals

$$\sum_{i=1}^{N} w(x_i, x_0, h)(y_i - \alpha_0)^2$$

  - which yields $\widehat{\alpha}_0 = \sum_{i=1}^{N} w(x_i, x_0, h) y_i$ which is the estimator in the previous slide.
  - so called (kernel-weighted) **local constant regression.**

Local linear regression

- The local linear estimator generalizes to linear regression in the neighborhood of x_0.
- Then $\widehat{f}(x_0) = \widehat{\alpha}_0$ where minimize w.r.t. α_0 and β_0

$$\sum_{i=1}^{N} w(x_i, x_0, h)\{y_i - \alpha_0 - \beta_0(x_i - x_0)\}^2.$$

  - so called (kernel-weighted) **local linear regression**.
- Advantages
  - better estimates at endpoints of the data
  - $\widehat{\beta}_0 = \widehat{f}'(x_0)$ provides an estimate of the gradient $\partial E[y|x]/\partial x|_{x_0}$.
- Stata commands
  - `lpoly y x, degree(1)`
  - or `npregress kernel y x, estimator(linear)`

Other nonparametric methods

- **Lowess** (locally weighted scatterplot smoothing)
  - a variation of local linear with variable bandwidth, tricubic kernel and downweighting of outliers
  - `lowess y x`.
- **k-nearest neighbors**
  - use an equal-weighted average of y values for the k observations with x_i closest to x_0
  - $\widehat{f}(\mathbf{x}_0) = \frac{1}{k}\sum_{i=1}^{N} \mathbf{1}[\mathbf{x}_i \in N(\mathbf{x}_0)] \times y_i$
  - Stata user-written command `knnreg`
  - not used much in economics aside from matching estimates of ATE.

1.2 Curse of Dimensionality

- Nonparametric methods do not extend well to multiple regressors.
- Consider p-dimensional $\mathbf{x}$ broken into bins
  - for $p = 1$ we might average y in each of 10 bins of x
  - for $p = 2$ we may need to average over 10^2 bins of (x_1, x_2)
  - and so on.
- On average there may be few to no points with high-dimensional $\mathbf{x}_i$ close to $\mathbf{x}_0$
  - called the **curse of dimensionality**.
- Formally for local constant kernel regression with bandwidth h
  - bias is $O(h^2)$ and variance is $O(nh^p)$
  - optimal bandwidth is $O(n^{-1/(p+4)})$
    - ★ gives asymptotic bias so standard conf. intervals not properly centered
  - convergence rate is then $n^{-2/(p+4)} << n^{-0.5}$

1.3 Kernel Regression in Higher Dimensions

- Kernel regression extends to higher dimensions.
- For $\dim(\mathbf{x}) = k$ local linear regression α_0 and $\boldsymbol{\beta}_0$ minimize

$$\sum_{i=1}^{N} w(\mathbf{x}_i, \mathbf{x}_0, \mathbf{h})\{y_i - \alpha_0 - (\mathbf{x}_i - \mathbf{x}_0)'\boldsymbol{\beta}_0\}^2$$

  where $w(\mathbf{x}_i, \mathbf{x}_0, \mathbf{h}) = \prod_{j=1}^{k} w(x_{ji}, x_{j0}, h_j)$ is a **product kernel**.
- Then $\widehat{\alpha}_0 = \widehat{f}(\mathbf{x}_0)$ can predict $f(\mathbf{x}_0)$ poorly due to curse of dimensionality.
- But when we average we may predict better.
- In particular can compute quantities such as
  - average marginal effect for the j^{th} regressor
  - marginal effect for the j^{th} regressor with other regressors set at their mean values or a prespecified value
- Stata's `npregress kernel` command does this
  - use bootstrap to get confidence intervals.

1.4 Semiparametric Models

- Semiparametric methods place some structure on the problem
  - parametric component $(\boldsymbol{\beta})$ for part of the model
  - nonparametric component (function(s) f) that is often one dimensional
- Ideally $\sqrt{N}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}) \xrightarrow{d} \mathcal{N}[\mathbf{0}, \mathbf{V}]$ despite the nonparametric component.
- Leading examples
  - partial linear (used in economics)
  - single-index (used in economics)
  - generalized additive model (used in statistics)
  - project pursuit (used in statistics).

Leading semiparametric models

- **Partial linear model**: $E[y_i|\mathbf{x}_i, \mathbf{z}_i] = \mathbf{x}_i'\boldsymbol{\beta} + g(\mathbf{z}_i)$ where $g(\cdot)$ not specified.
  - use Robinson differencing estimator.
- **Single index model**: $\mathrm{E}[y_i|\mathbf{x}_i] = g(\mathbf{x}_i'\boldsymbol{\beta})$ where $g(\cdot)$ not specified
  - Ichimura semiparametric least squares
    - ★ $\widehat{\boldsymbol{\beta}}$ and $\widehat{g}$ minimize $\sum_{i=1}^{N} w(\mathbf{x}_i)\{y_i - \widehat{g}(\mathbf{x}_i'\boldsymbol{\beta})\}^2$
    - ★ where $w(\mathbf{x}_i)$ is a trimming function that drops outlying $\mathbf{x}$ values.
  - can only estimate $\boldsymbol{\beta}$ up to scale in this model
    - ★ still useful as ratio of coefficients equals ratio of marginal effects in a single-index models.
- **Generalized additive model**: $E[y_i|\mathbf{x}_i] = g_1(x_{1i}) + \cdots + g_K(x_{Ki})$ where the $g_j(\cdot)$ are unspecified.
  - estimate by backfitting
  - can make more complex by e.g. $x_{3i} = x_{1i} \times x_{2i}$.

1.5 How can ML methods do better?

- Machine learning methods can outperform nonparametric and semiparametric methods
  - so wherever econometricians use nonparametric and semiparametric regression in higher-dimensional models it may be useful to use ML methods.
  - In theory there is scope for improving nonparametric methods.
- k-nearest neighbors usually has a fixed number of neighbors
  - but it may be better to vary the number of neighbors with data sparsity
- Kernel-weighted local regression methods usually use a fixed bandwidth
  - but it may be better to vary the bandwidth with data sparsity.
- There may be advantage to basing neighbors in part on relationship with y.

2. Dimension Reduction

- **Reduce** from p regressors to $M < p$ linear combinations of regressors.
- So form

$$\underset{(N\times m)}{\mathbf{X}^*} = \underset{(N\times p)}{\mathbf{X}} \times \underset{(p\times M)}{\mathbf{A}} \text{ where } M < p.$$

- Then after dimension reduction

$$\begin{aligned} \mathbf{y} &= \beta_0 + \mathbf{X}^*\delta + \mathbf{u} \\ &= \beta_0 + \mathbf{X}\boldsymbol{\beta} + \mathbf{u} \text{ where } \boldsymbol{\beta} = \mathbf{A}\delta. \end{aligned}$$

- Aside: in ML **high-dimensional** simply means p is large relative to n
  - some methods (not PCA) even allow $p > n$
  - n could be large or small.

Dimension Reduction (continued)

- Two methods are covered in ISL
  - 1. Principal components (PCA)
    - ★ use only **X** to form **A** (unsupervised)
  - 2. Partial least squares (PLS)
    - ★ also use relationship between **y** and **X** to form **A** (supervised).
- PCA is widely used in other social sciences
  - and is related to factor analysis.
- PCA using just the first few predict y almost as well as using all the predictors, with less risk of overfitting.
- For PCA standardize regressors as the method is not scale invariant
  - and can use cross-validation to determine the number of component M.
- PLS is really not used in practice.

2.1 Principal Components Analysis (PCA)

- Suppose $\mathbf{X}$ is normalized to have zero means so ij^{th} entry is $x_{ji} - \bar{x}_j$.
- The first principal component has the largest sample variance among all normalized linear combinations of the columns of $n \times p$ matrix $\mathbf{X}$.
- How is this calculated?
  - the first component is $\mathbf{X}\mathbf{h}_1$ where $\mathbf{h}_1$ is $p \times 1$
  - normalize $\mathbf{h}_1$ so that $\mathbf{h}_1'\mathbf{h}_1 = 1$
  - then $\mathbf{h}_1$ max $\text{Var}(\mathbf{X}\mathbf{h}_1) = \mathbf{h}_1'\mathbf{X}'\mathbf{X}\mathbf{h}_1$ subject to $\mathbf{h}_1'\mathbf{h}_1 = 1$
  - the maximum is the largest eigenvalue of $\mathbf{X}'\mathbf{X}$ and $\mathbf{h}_1$ is the corresponding eigenvector.
- The second principal component has the largest variance subject to being orthogonal to the first
  - and so on.

Formulas for PCA

- Eigenvalues and eigenvectors of $\mathbf{X}'\mathbf{X}$
  - Let $\Lambda = \text{Diag}[\lambda_j]$ be $p \times p$ vector of eigenvalues of $\mathbf{X}'\mathbf{X}$
  - Order so $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_1$
  - Let $\mathbf{H} = [\mathbf{h}_1 \ \cdots \ \mathbf{h}_p]$ be $p \times p$ vector of corresponding eigenvectors
  - $\mathbf{X}'\mathbf{X}\mathbf{h}_1 = \lambda_1\mathbf{h}_1$ and $\mathbf{X}'\mathbf{X}\mathbf{H} = \Lambda\mathbf{H}$ and $\mathbf{H}'\mathbf{H}$
- Then
  - the j^{th} principal component is $\mathbf{X}\mathbf{h}_j$
  - $M-$principal components regression uses $\mathbf{X}^* = \mathbf{X}\mathbf{A}$ where $\mathbf{A} = [\mathbf{h}_1 \ \cdots \ \mathbf{h}_M]$.
- ASIDE: PCA is related to, but not exactly the same as factor analysis
  - factor analysis decomposes observed $p-$dimensional $\mathbf{x}$ into a linear combination of M unobserved factor loadings and p i.i.d. errors
  - often $M = 1$ so x_{ji} = a multiple of the one common factor plus i.i.d. noise.

Principal Components Analysis Example

- Stata command pca by default standardizes the data.
- For the d.g.p. for x_1, x_2, x_3 (the d.g.p. in part 1 of these slides) we expect eigenvalues 2, 0.5 and 0.5 as $n \longrightarrow \infty$.

```
. * Principal components with default correlation option that standardizes data
. pca x1 x2 x3

Principal components/correlation                  Number of obs    =         40
                                                  Number of comp.  =          3
                                                  Trace            =          3
    Rotation: (unrotated = principal)             Rho              =     1.0000
```

Component	Eigenvalue	Difference	Proportion	Cumulative
Comp1	1.81668	1.08919	0.6056	0.6056
Comp2	.727486	.27165	0.2425	0.8481
Comp3	.455836	.	0.1519	1.0000

Principal components (eigenvectors)

Variable	Comp1	Comp2	Comp3	Unexplained
x1	0.6306	-0.1063	-0.7688	0
x2	0.5712	-0.6070	0.5525	0
x3	0.5254	0.7876	0.3220	0

Principal Components Analysis Example (continued)

- First principal component is $0.6306zx_1 + 0.5712zx_2 + 0.5254zx_3$
  - where zx_j are the standardized x_j
  - and has variance 1.8618 that explains $1.8618/3 = 62.56\%$ of the variance.
- The principal components have means 0, variances equal to the eigenvalue, and are uncorrelated.

```
. * Generate the 3 principal components and their means, st.devs., correlations
. quietly predict pc1 pc2 pc3

. summarize pc1 pc2 pc3
```

Variable	Obs	Mean	Std. Dev.	Min	Max
pc1	40	-3.35e-09	1.347842	-2.52927	2.925341
pc2	40	-3.63e-09	.8529281	-1.854475	1.98207
pc3	40	2.08e-09	.6751564	-1.504279	1.520466

```
. correlate pc1 pc2 pc3
(obs=40)
```

	pc1	pc2	pc3
pc1	1.0000		
pc2	0.0000	1.0000	
pc3	-0.0000	-0.0000	1.0000

Principal Components Analysis Example (continued)

- In-sample correlation coefficient of y with $\widehat{y}$
  - $r = 0.4871$ from OLS on all three regressors
  - $r = 0.4444$ from OLS on first principal component
  - $r = 0.4740$ on just x1 (the d.g.p. was $y_i = 2 + x_{1i} + u_i$) .

```
. * Compare R from OLS on all three regressors, on pc1, on x1, on x2, on x3
. qui regress y x1 x2 x3

. predict yhat
(option xb assumed; fitted values)

. correlate y yhat pc1 x1 x2 x3
(obs=40)
```

	y	yhat	pc1	x1	x2	x3
y	1.0000					
yhat	0.4871	1.0000				
pc1	0.4444	0.9122	1.0000			
x1	0.4740	0.9732	0.8499	1.0000		
x2	0.3370	0.6919	0.7700	0.5077	1.0000	
x3	0.2046	0.4200	0.7082	0.4281	0.2786	1.0000

Principal Components Analysis (continued)

- PCA is unsupervised so seems unrelated to $\mathbf{y}$ but
  - *Elements of Statistical Learning* says does well in practice.
  - PCA has the smallest variance of any estimator that estimates the model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{u}$ with i.i.d. errors subject to constraint $\mathbf{C}\boldsymbol{\beta} = \mathbf{c}$ where $\dim[\mathbf{C}] \leq \dim[\mathbf{X}]$.
  - PCA discards the $p - M$ smallest eigenvalue components whereas ridge does not, though ridge does shrink towards zero the most for the smallest eigenvalue components (ESL p.79).
- For machine learning the tuning parameter is the number of components and use e.g. K-fold class validation to determine this.
- For completeness next give partial least squares which is supervised.

2.2 Partial Least Squares

- Partial least squares produces a sequence of orthogonal linear combinations of the regressors.
- 1. Standardize each regressor to have mean 0 and variance 1.
- 2. Regress y individually on each $\mathbf{x}_j$ and let $\mathbf{z}_1 = \sum_{j=1}^{p} \widehat{\theta}_{1j}\mathbf{x}_j$
- 3. Regress y on $\mathbf{z}_1$ and let $\widehat{\mathbf{y}}^{(1)}$ be prediction of $\mathbf{y}$.
- 4. Orthogonalize each $\mathbf{x}_j$ by regress on $\mathbf{z}_1$ to give $\mathbf{x}_j^{(1)} = \mathbf{x}_j - \mathbf{z}_1\widehat{\tau}_j$ where $\widehat{\tau}_j = (\mathbf{z}_1'\mathbf{z}_1)^{-1}\mathbf{z}_1'\mathbf{x}_j^{(1)}$.
- 5. Go back to step 1 with $\mathbf{x}_j$ now $\mathbf{x}_j^{(1)}$, etc.
  - When done $\widehat{\mathbf{y}} = \widehat{\mathbf{y}}^{(1)} + \widehat{\mathbf{y}}^{(2)} + \cdots$
- Partial least squares turns out to be similar to PCA
  - especially if R^2 is low
  - in practice PCA is used rather than partial least squares.

3. Flexible Regression using Basis Functions

- Basis function models (or sieves) include
  - global polynomial regression
  - splines: step functions, regression splines, smoothing splines, b-splines
  - polynomial is global while the others break range of x into pieces.
- Can make nonparametric
  - increase order of polynomial or number of knots (split points) in splines
  - select model using leave-one-out cross validation, generalized cross validation, Mallows CP, AIC or BIC
- Stata `npregress series` command implements these methods
  - options `polynomial`, `spline`, `bspline`
  - nonparametric tuning use option `criterion()`
  - or parametric tuning use options `polynomial(#)`, `spline(#)`, `knots(#)`.

3.1 Basis Functions

- Also called series expansions and sieves.
- General approach (scalar x for simplicity)

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \cdots + \beta_K b_K(x_i) + \varepsilon_i$$

  - where $b_1(\cdot), ..., b_K(\cdot)$ are basis functions that are fixed and known.
- **Global polynomial** regression sets $b_j(x_i) = x_i^j$
  - typically $K \leq 3$ or $K \leq 4$.
  - fits globally and can overfit at boundaries.
- **Step functions**: separately fit y in each interval $x \in (c_j, c_{j+1})$
  - could be piecewise constant or piecewise linear.
- **Splines** smooth so that not discontinuous at the cut points.
- **Wavelets** are also basis functions, richer than Fourier series.

Global Polynomials Example

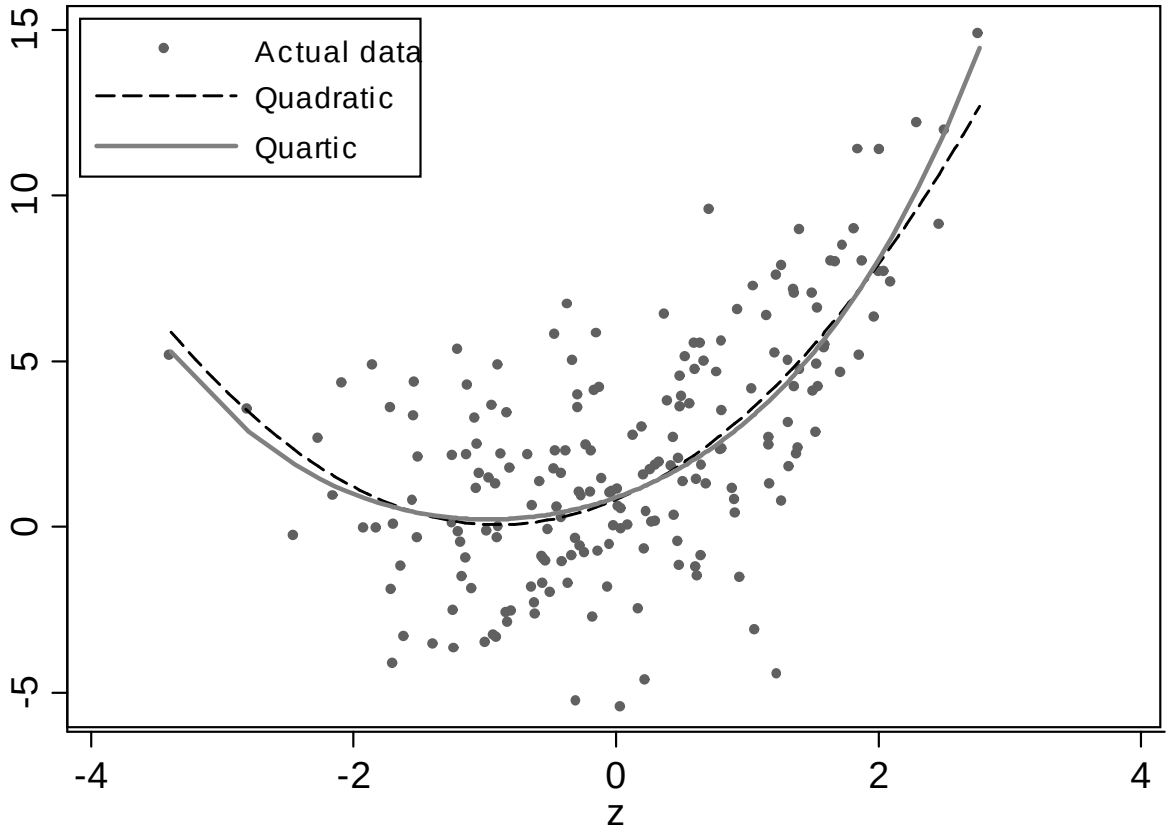
- Generated data:
  $y_i = 1 + x_{1i} + x_{2i} + f(z_i) + u_i$ where $f(z) = z + z^2$.

```
. * Generated data: y = 1 + 1*x1 + 1*x2 + f(z) + u where f(z) = z + z^2
. clear

. set obs 200
number of observations (_N) was 0, now 200

. set seed 10101

. generate x1 = rnormal()

. generate x2 = rnormal() + 0.5*x1

. generate z = rnormal() + 0.5*x1

. generate zsq = z^2

. generate y = 1 + x1 + x2 + z + zsq + 2*rnormal()

. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
x1	200	.0301211	1.014172	-3.170636	3.093716
x2	200	.0226274	1.158216	-4.001105	3.049917
z	200	.0664539	1.146429	-3.386704	2.77135
zsq	200	1.312145	1.658477	.0000183	11.46977
y	200	2.164401	3.604061	-5.468721	14.83116

Global Polynomials Example (continued)

- Fit quartic in z (with x_1 and x_2 omitted) and compare to quadratic in z
  - `regress y c.z##c.z##c.z##c.z, vce(robust)`
  - quartic chases endpoints.



3.2 Regression Splines

- Begin with **step functions**: separate fits in each interval (c_j, c_{j+1})
- Piecewise constant
  - $b_j(x_i) = 1[c_j \leq x_i < c_{j+1}]$
- Piecewise linear
  - intercept is $1[c_j \leq x_i < c_{j+1}]$ and slope is $x_i \times 1[c_j \leq x_i < c_{j+1}]$
- Problem is that discontinuous at the cut points (does not connect)
  - solution is splines.

Piecewise linear spline

- Begin with piecewise linear with two knots at c and d

$$\begin{aligned} f(x) &= \alpha_1 1[x < c] + \alpha_2 1[x < c]x + \alpha_3 1[c \leq x < d] \\ &\quad + \alpha_4 1[c \leq x < d]x + \alpha_5 1[x \geq d] + \alpha_6 1[x \geq d]x. \end{aligned}$$

- To make continuous at c (so $f(c-) = f(c)$) and d (so $f(d-) = f(d)$) we need two constraints

$$\begin{aligned} \text{at } c: &\quad \alpha_1 + \alpha_2 c = \alpha_3 + \alpha_4 c \\ \text{at } d: &\quad \alpha_3 + \alpha_4 d = \alpha_5 + \alpha_6 d. \end{aligned}$$

- Alternatively introduce the Heaviside step function

$$h_+(x) = x_+ = \begin{cases} x & x > 0 \\ 0 & \textit{otherwise}. \end{cases}$$

- Then the following imposes the two constraints (so have $6 - 2 = 4$ regressors)

$$f(x) = \beta_0 + \beta_1 x + \beta_2 (x - c)_+ + \beta_3 (x - d)_+$$

Spline Example

- Piecewise linear spline with two knots done manually.

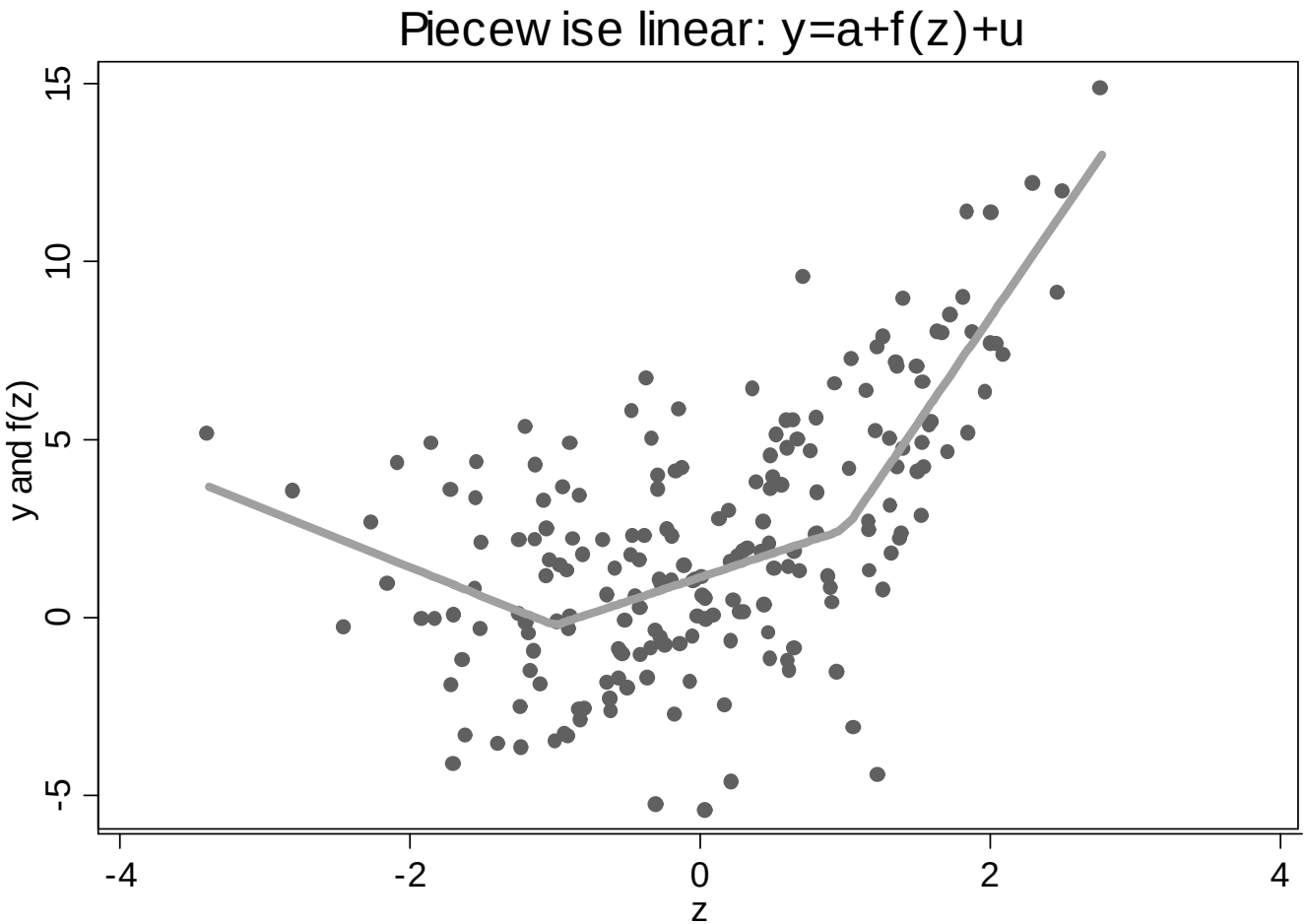
```
. * Create the basis function manually with three segments and knots at -1 and 1
. generate zseg1 = z

. generate zseg2 = 0

. replace zseg2 = z - (-1) if z > -1
(163 real changes made)

. generate zseg3 = 0

. replace zseg3 = z - 1 if z > 1
(47 real changes made)

.
. * Piecewise linear regression with three sections
. regress y zseg1 zseg2 zseg3
```

Source	SS	df	MS		
Model	1253.3658	3	417.7886	Number of obs	= 200
Residual	1331.49624	196	6.79334818	F(3, 196)	= 61.50
				Prob > F	= 0.0000
				R-squared	= 0.4849
				Adj R-squared	= 0.4770
Total	2584.86204	199	12.9892565	Root MSE	= 2.6064

y	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
zseg1	-1.629491	.6630041	-2.46	0.015	-2.937029	-.3219535
zseg2	2.977586	.8530561	3.49	0.001	1.295239	4.659933
zseg3	4.594974	.9164353	5.01	0.000	2.787634	6.402314
_cons	-1.850531	.9204839	-2.01	0.046	-3.665855	-.0352065

Spline Example (continued)

- Plot of fitted values from piecewise linear spline has three connected line segments.



Spline Example (continued)

- The `mkspline` command creates the same spline variables.

```
. * Repeat piecewise linear using command mkspline to create the basis functions
. mkspline zmk1 -1 zmk2 1 zmk3 = z, marginal

. summarize zseg1 zmk1 zseg2 zmk2 zseg3 zmk3, sep (8)
```

Variable	Obs	Mean	Std. Dev.	Min	Max
zseg1	200	.0664539	1.146429	-3.386704	2.77135
zmk1	200	.0664539	1.146429	-3.386704	2.77135
zseg2	200	1.171111	.984493	0	3.77135
zmk2	200	1.171111	.984493	0	3.77135
zseg3	200	.138441	.3169973	0	1.77135
zmk3	200	.138441	.3169973	0	1.77135

- To repeat earlier results: `regress y zmk1 zmk2 zmk3`
- And to add regressors: `regress y x1 x2 zmk1 zmk2 zmk3`

Cubic Regression Splines

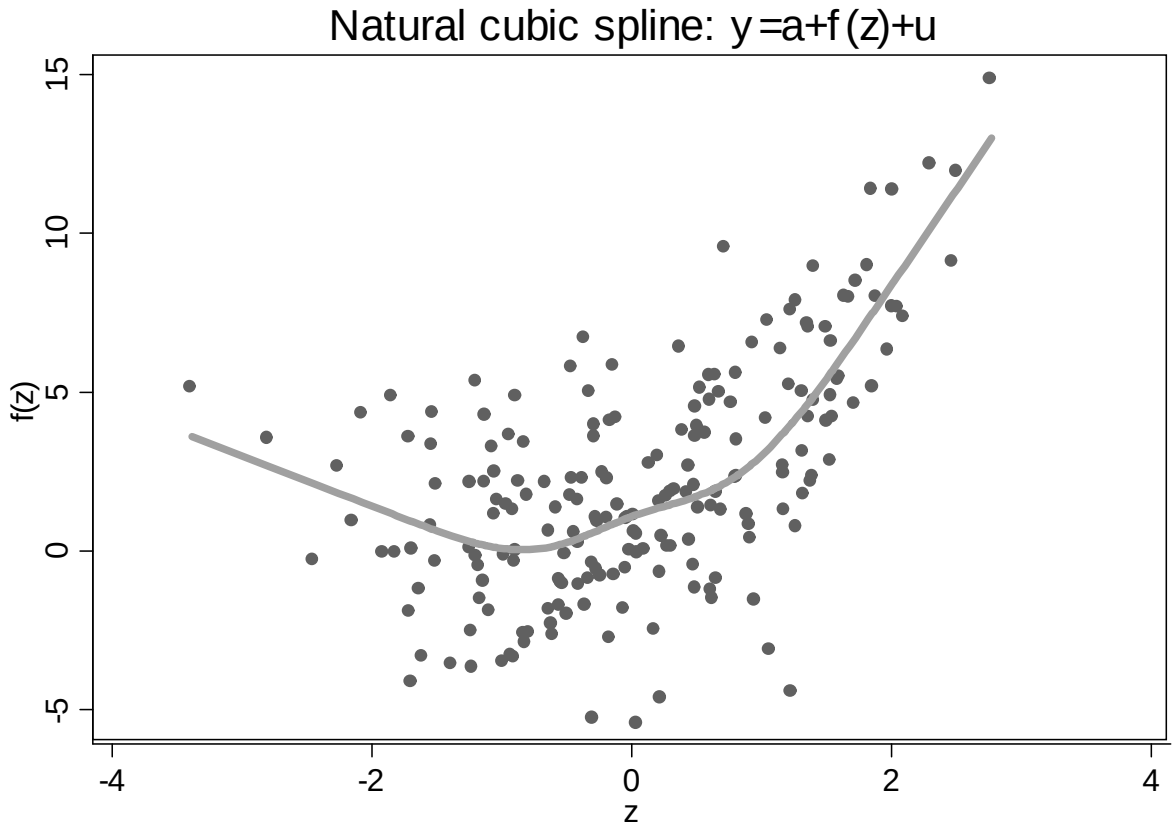
- This is the standard.
- **Piecewise cubic spline** with K knots
  - require $f(x)$, $f'(x)$ and $f''(x)$ to be continuous at the K knots
- Then can do OLS with

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - c_1)_+^3 + \cdots + \beta_{(3+K)} (x - c_K)_+^3$$

  - for proof when $K = 1$ see ISL exercise 7.1.
- This is the lowest degree regression spline where the graph of $\widehat{f}(x)$ on x seems smooth and continuous to the naked eye.
- There is no real benefit to a higher-order spline.
- Regression splines overfit at boundaries.
- A **natural (or restricted) cubic spline** is an adaptation that restricts the relationship to be linear past the lower and upper knots.

Spline Example

- Natural or restricted cubic spline with five knots at the 5, 27.5, 50, 72.5 and 95 percentiles
  - `mkspline zspline = z, cubic nknots(5) displayknots`
  - `regress y zspline*`



Other Splines

- Regression splines and natural splines require choosing the cut points
  - e.g. use quintiles of x.
- **Smoothing splines** avoid this
  - use all distinct values of x as knots
  - but then add a smoothness penalty that penalizes curvature.
- The function $g(\cdot)$ minimizes

$$\sum_{i=1}^{n}(y_i - g(\mathbf{x}_i))^2 + \lambda \int_a^b g''(t)dt \text{ where } a \leq \text{all } x_i \leq b.$$

  - $\lambda = 0$ connects the data points and $\lambda \rightarrow \infty$ gives OLS.
- For multivariate splines use multivariate adaptive regression splines (MARS).

Stata Commands

- The preceding examples were done manually for pedagogical reasons.
- Stata's `npregress series` command has options
  - `polynomial(#)` for a global polynomial of order #
  - `spline(#)` for a natural spline of order #
  - `bspline(#)` for a b-spline of order #
- For splines and B-splines the number of knots can be determined
  - by CV (the default), generalized CV, AIC, BIC or Cp
  - option `knots(#)` where # is the number of knots
  - option `knotsmat(`*matname*`)` specifies the values of the knots.
- If there is more than one regressor then the basis functions for each regressor may be interacted or not interacted.
- Stata user-written add-on commands
  - `gam` (Royston and Ambler) for smoothing splines
  - `bspline` command (Newson 2012) for a range of bases.

3.3 Wavelets

- **Wavelets** are used especially for signal processing and extraction.
  - they are richer than a Fourier series basis.
  - they can handle both smooth sections and bumpy sections of a series.
- Wavelets are not used in cross-section econometrics
  - they may be useful for some time series.
- Start with a mother or father wavelet function $\psi(x)$.
  - example is the Haar function $\psi(x) = \begin{cases} 1 & 0 \leq x < \frac{1}{2} \\ -1 & \frac{1}{2} < x < 1 \\ 0 & \text{otherwise} \end{cases}$
- Then both translate by b and scale by a to give basis functions $\psi^{ab}(x) = |a|^{-1/2}\psi(\frac{x-b}{a})$.

4. Neural Networks (and Deep Learning)

- A **neural network** is a model with one or more hidden layers with multiple units in each layer
- The term **neural** arises as initial models were based on mimicking how the brain works
  - a more complete name is artificial neural network.
- Neural networks have been around for a long time
  - but to work well they need to be complex with many parameters
  - this requires a lot of data and good computational techniques.
- Their use has exploded in the past fifteen years
  - due to more computer power, more data, better algorithms, newer models
  - they work especially well for image recognition and language translation (Google Translate)
  - and are the basis for generative AI.
- **Deep learning** is learning that occurs in a series of levels or layers that goes to considerable **depth**.

4.1 Multilayer Perceptron (MLP) Neural Network

- Like many neural net models originally proposed for classification
  - we consider use for regression (see part 6 for classification).
- Also called a sequential neural net.
- A single hidden layer network explaining y by $\mathbf{x}$ has
  - y depends linearly on $\mathbf{z}'s$ (**hidden units**)
  - $\mathbf{z}'s$ are a nonlinear transformation of linear combinations of $\mathbf{x}'s$ (**input units**).
- History
  - proposed in 1943 to model how the brain processes vision
  - Rosenblatt in 1958 developed a physical machine
  - better computational algorithms developed in the late 1960's
  - but not widely used until 2000's.

One hidden layer neural network (single layer perceptron)

- y depends on p $\mathbf{x}'s$
- Introduce an intermediate hidden layer of M $\mathbf{z}'s$
  - y is a linear combination of M $\mathbf{z}'s$
  - the $\mathbf{z}'s$ are each a nonlinear transformation of a linear combination of the p $\mathbf{x}'s$
  - $y \leftarrow \mathbf{z} \leftarrow \mathbf{x}$.
- Then $y = f(\mathbf{x})$ with hidden units $z_1(\mathbf{x}), ..., z_m(\mathbf{x})$

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \mathbf{z}'\boldsymbol{\beta} \\ &= \beta_0 + \textstyle\sum_{m=1}^{M} \beta_m z_m(\mathbf{x}) \end{aligned}$$

$$\begin{aligned} z_m(\mathbf{x}) &= g(\alpha_{0m} + \mathbf{x}'\boldsymbol{\alpha}_m) && \text{for specified function } g(\cdot) \\ &= g(\alpha_{0m} + \textstyle\sum_{j=1}^{p} \alpha_{mj} x_j) && m = 1, ..., M. \end{aligned}$$

One hidden layer neural network (continued)

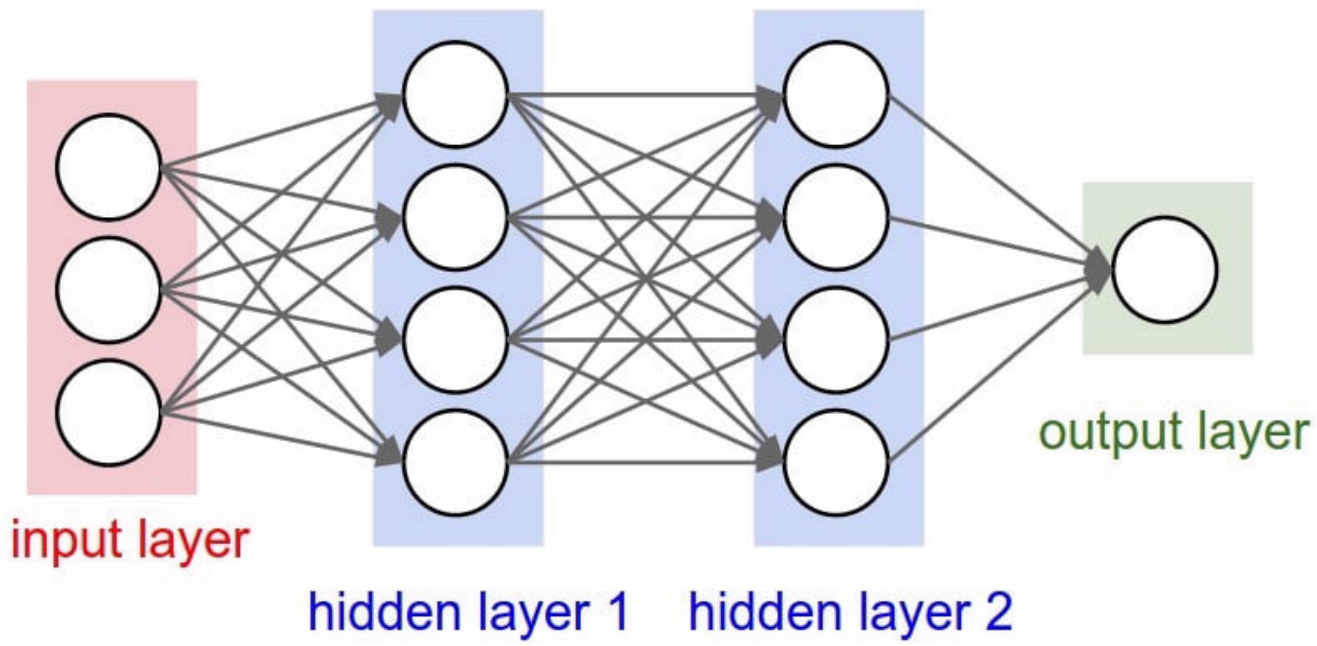
- We have

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \sum_{m=1}^{M} \beta_m z_m(\mathbf{x}) \\ z_m(\mathbf{x}) &= g(\alpha_{0m} + \sum_{j=1}^{p} \alpha_{mj} x_j),\ m = 1, ..., M. \end{aligned}$$

- The specified nonlinear function $g(\cdot)$ is called an **activation function**
  - $g(v) = \max(0, v)$ is the **rectified linear unit** (ReLU) activation
    - ★ computationally quick to compute
  - $g(v) = \frac{1}{1+e^{-v}}$ is the **sigmoid** activation (logit)
  - $g(v) = \exp(v_k) / \sum_{l=1}^{K} \exp(v_l)$ is the **softmax** activation (MNL).
    - ★ softmax is used for classification with outcomes $y_1, ..., y_K$.
- The α_{0m}, $m = 1, ..., M$, are called **biases**.
- The α_{mj}, $j = 1, ..., p$, $m = 1, ..., M$, are called **weights**.

Two hidden layer sequential neural network (perceptron)

- Outcome y depends on $\mathbf{w}'s$ (**hidden units**), $\mathbf{w}'s$ depend on $\mathbf{z}'s$ (**hidden units**) and $\mathbf{z}'s$ depend on $\mathbf{x}'s$ (**input units**).
- Input $\mathbf{x} \longrightarrow \mathbf{z} \longrightarrow \mathbf{w} \longrightarrow y$ output
  - a variation is classification with more than one output y.



4.2 Computation

- A neural network model has a **highly nonlinear and nonconvex objective function**
  - Using squared error loss we minimize $\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \boldsymbol{\theta}))^2$.
  - With one hidden layer $\boldsymbol{\theta} = [\beta_0, ..., \beta_m, \alpha_{10}, ..., \alpha_{1p}, ..., \alpha_{M0}, ..., \alpha_{Mp}]$ minimizes

$$Q(\boldsymbol{\theta}) = \sum_{i=1}^{n}\left\{y_i - \beta_0 - \sum_{m=1}^{M}\beta_m\left[g\left(\alpha_{m0} + \sum_{j=1}^{p}\alpha_{mj}x_j\right)\right]\right\}^2.$$

- Optimization is speeded up by innovations in the 1960's
  - stochastic gradient descent
  - back propagation.
- Note that here we are only wanting to get a good $\widehat{y}$
  - we are not interested in the parameter estimates per se.

Computation: Stochastic Gradient Descent

- (Mini-batch) **stochastic gradient descent** (SGD) uses update rule

$$\widehat{\boldsymbol{\theta}}_{s+1} = \widehat{\boldsymbol{\theta}}_s - \lambda_s \left. \frac{\partial Q_s(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right|_{\widehat{\boldsymbol{\theta}}_s}.$$

- Computation at each step is fast for two reasons.
- (1) $Q_s(\cdot)$ is just a **small randomly-chosen subsample** of the data
  - so the gradient is computed using only a subsample of the data
  - pure SGD uses just one randomly chosen observation
  - **mini-batch** SGD uses a subsample
    - ★ the Keras `model.fit` function has a default of 32 observations
  - an **epoch** is a complete pass through all the data
    - ★ e.g. approximately $N/32$ iterations with mini-batch size 32.
- (2) We simply multiply by a scalar λ_s (rather than e.g. $\mathbf{H}_s^{-1}$).
  - λ_s is called the **learning rate** and is small e.g. $\lambda_s = 0.01$
  - it is best to let λ_s decline with s.
- Aside: In econometrics we instead use $\widehat{\boldsymbol{\theta}}_{s+1} = \widehat{\boldsymbol{\theta}}_s - H_s^{-1} \times \left. \frac{\partial Q(\boldsymbol{\theta})}{\partial(\boldsymbol{\theta})} \right|_{\widehat{\boldsymbol{\theta}}_s}$

Computation: Stochastic Gradient Descent

- SGD requires many iterations e.g. 1,000
  - recall there are many iterations per epoch.
- Due to the sample used for the gradient changing at each iteration there is chatter in the gradient and the loss function
  - so stop when on average there is little change in validation loss.
- The use of different subsamples at each iteration makes it less likely to reach only a local minimum.
- And it helps with regularization (reducing overfitting).
- To further avoid overfitting use a Ridge or Lasso penalty in $Q(\boldsymbol{\theta})$
  - or dropout regularization that randomly drops some of the hidden units.
- Nowadays better variants of SGD are used
  - Especially Adam (Adaptive Moment) which is a variation of Momentum
  - Momentum uses $\Delta\widehat{\boldsymbol{\theta}}_{s+1} = \alpha_s \Delta\widehat{\boldsymbol{\theta}}_s - \lambda_s \left.\frac{\partial Q_s(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\widehat{\boldsymbol{\theta}}_s}$
    - ★ rather than $\Delta\widehat{\boldsymbol{\theta}}_{s+1} = -\lambda_s \left.\frac{\partial Q_s(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\widehat{\boldsymbol{\theta}}_s}$.

Computation: Rescaling Data

- For neural nets(and machine learners in general) it is best to have inputs on a similar scale.
  - Standardization converts the training data inputs to have mean 0 and variance 1: $z_i^* = (z_i - \bar{z})/s_z$.
  - Normalization converts the training data inputs to the 0-1 scale: $z_i^* = (z_i - \min(z_i))/\{\max(z_i) - \min(z_i)\}$
    - ★ not as robust as outlying $\min(z_i)$ or $\max(z_i)$ has a big effect.
- For a single target y there is usually no rescaling
  - e.g. for $Q(\boldsymbol{\theta}) = \sum_{i=1}^n (y_i - \mathbf{x}_i'\boldsymbol{\theta})^2$ if y is 1,000 times larger then $\boldsymbol{\theta}$ is 1,000 times larger and $\partial Q(\boldsymbol{\theta})/\partial\boldsymbol{\theta} = \sum_{i=1}^n -2(y_i - \mathbf{x}_i'\boldsymbol{\theta})\mathbf{x}_i$ is 1,000 times larger
  - so $\widehat{\boldsymbol{\theta}}_{s+1} = \widehat{\boldsymbol{\theta}}_s - \lambda_s \left.\frac{\partial Q_s(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\widehat{\boldsymbol{\theta}}_s}$ leads to 1,000 times larger change in $\widehat{\boldsymbol{\theta}}_{s+1}$ (for unchanged learning rate λ_s).
- For multiple targets $\mathbf{y}$ need to scale appropriately the individual components of $\mathbf{y}$ to ensure a sensible overall loss function.

Back Propagation

- **Back propagation** computes gradients using the chain rule

$$\frac{\partial q_i(\boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \alpha_{mj}} = -2(y_i - f(\mathbf{x}_i)) \times \boldsymbol{\beta}_m \times \frac{\partial g(\mathbf{x}_i' \boldsymbol{\alpha}_m)}{\partial \alpha_{mj}} \times x_{ij}$$

$$\frac{\partial q_i(\boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \beta_m} = -2(y_i - f(\mathbf{x}_i)) \times \mathbf{g}(\mathbf{x}_i' \boldsymbol{\alpha}_m)$$

- This saves computation time as e.g. $-2(y_i - f(\mathbf{x}_i)) \times \boldsymbol{\beta}_m$ is the same for all $\alpha_{m1}, ..., \alpha_{mp}$.
  - "Forward propagation" goes from inputs to output: $x \longrightarrow z \longrightarrow \widehat{y}$
  - Called back propagation as gradient computation order is $\widehat{y} \longrightarrow z \longrightarrow x$.

Computation (continued)

- Neural nets require a lot of fine tuning - not off-the-shelf.
- For a multilayer perceptron (a sequential NN) we need to do a grid search over a range of possible values to choose
  - the number of hidden layers
  - the number M of hidden units within each layer
  - the activation function
  - a penalty to limit overfitting.
- If it is computationally burdensome we need at least to
  - choose the optimization algorithm (SGD, ADAM, ...)
  - choose the mini-batch size for stochastic gradient descent
  - choose step size λ_s (which should decrease with s)
  - possibly starting values for the $\alpha's$, $\beta's$, ...
  - further refinements - see Geron book.
- An example from Geron book using keras and tensorflow is at
  - https://cameron.econ.ucdavis.edu/python/python_neural_net.py

4.3 Shapley Value

- Motivation: how to explain relative importance of each feature.
- For a linear regression model with K regressors
  - rank regressors in terms of reduction in MSE
  - depends on ordering of the regressors
  - so consider marginal contribution of a regressors over all $K!$ unique regressions (including those with some regressors omitted)
  - computationally burdensome if K is large.
- Recently extended to ML methods such as neural nets
- Python package SHAP
  - Shapley Additive exPlanations
  - https://shap.readthedocs.io/en/latest/index.html

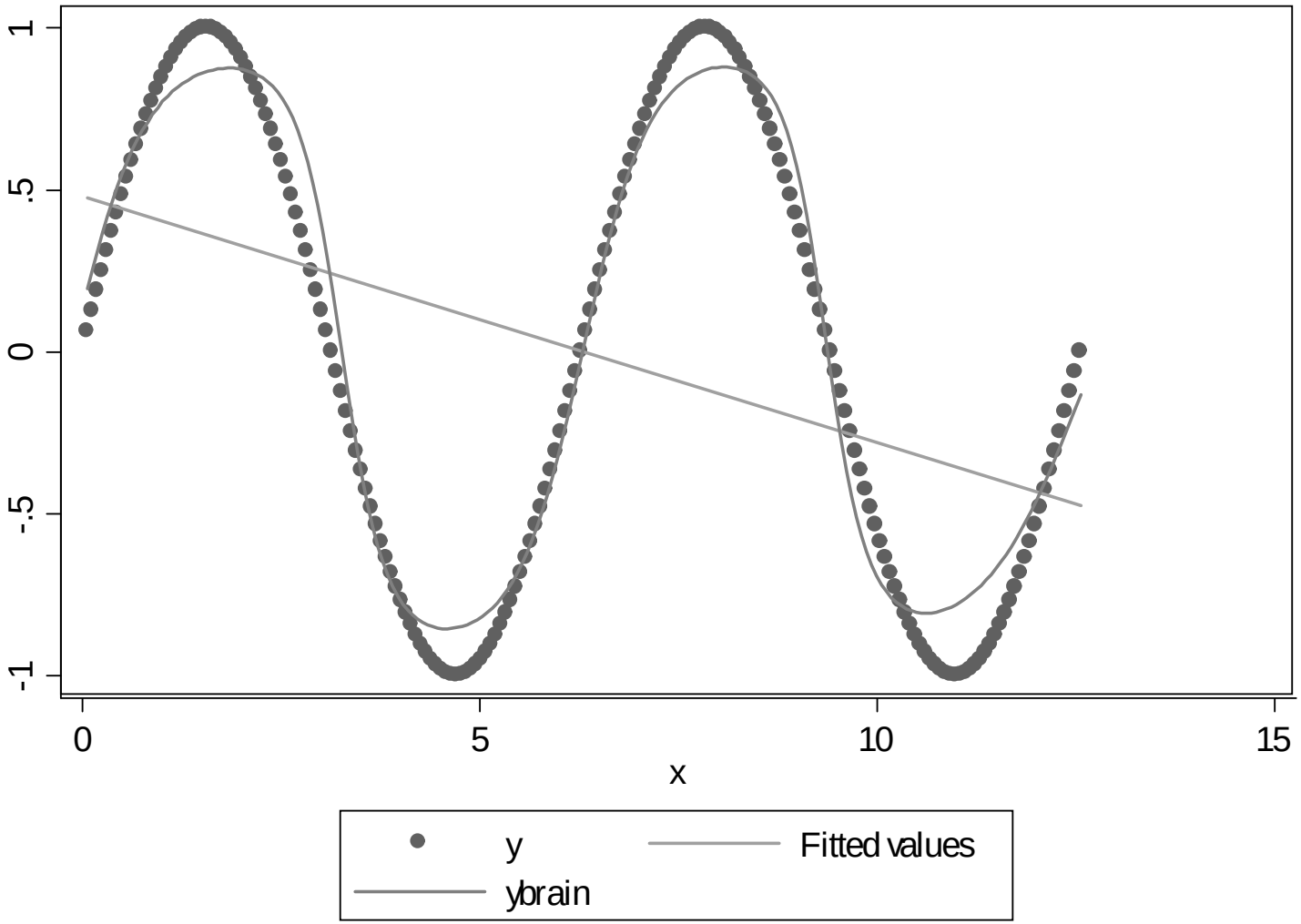
4.4 Neural Networks Example

- This example uses user-written Stata command `brain` (Doherr)
  - regression example with one hidden layer with 20 units

```
. * Example from help file for user-written brain command
. clear

. set obs 200
number of observations (_N) was 0, now 200

. gen x = 4*_pi/200 *_n

. gen y = sin(x)

. brain define, input(x) output(y) hidden(20)
Defined matrices:
   input[4,1]
  output[4,1]
  neuron[1,22]
   layer[1,3]
   brain[1,61]

. quietly brain train, iter(500) eta(2)

. brain think ybrain

. sort x

. twoway (scatter y x) (lfit y x) (line ybrain x)
```

Neural Networks Example (continued)

- We obtain



4.5 Other Neural Networks

- There are many types of neural networks.
- Recurrent neural networks for e.g. autocorrelated time series.
- Convolutional neural networks for images.
- Document classification with e.g. word-pairs as features

Recurrent neural networks

- Example from ISL2 and ISLP chapter 10.
- Consider prediction of a single y_t e.g. daily stock trading volume
- Predict using past values of y and $(p-1)$ other variables up to lag L
- So $\mathbf{x}_{t-l} = (y_{t-l}, x_{1,t-l}, ..., x_{p-1,t-l})$, $l = 1, ..., L$.
- Then in a one hidden layer model
  - the activation $z_{lm}(\cdot)$ for lag l and unit m depends not only on $\mathbf{x}_{t-l}$ but also on the M activations in the previous period
- We have for lags $l = 1, ..., L$ and units $m = 1, ..., M$

$$\begin{aligned} f_l(\mathbf{x}) &= \beta_0 + \sum_{m=1}^{M} \beta_m z_{lm}(\cdot) \\ z_{lm}(\cdot) &= g\left(\alpha_{0m} + \sum_{j=1}^{p} \alpha_{mj} x_{lj} + \sum_{s=1}^{M} \delta_{ms} z_{l-1,s}\right). \end{aligned}$$

- And we use the final outcome $y_L = f_L(\mathbf{x})$.
- Can also use recurrent neural net for word sequences
  - then classification.

Convolutional neural networks

- For classification of images
  - e.g. data are 32×32 pixels with three eight-bit numbers per pixel for red, green and blue.
- Data input features are three dimensional
  - pixel x coordinate x by colors.
- Use convolutional layers to find small patterns in the data such as edges and small shapes
- Use pooling layers that then reduce to a prominent subset.

4.5 References

- Geron book gives a very extensive discussion of neural nets
  - over half the book
  - uses Python modules `keras` and `tensorflow` for neural networks.
- Chapter 10 of ISL2 covers neural nets
  - summarizes various methods and types of neural net
  - uses the `keras` package in Python
  - https://www.statlearning.com/resources-second-edition has the same example done using the `torch` package in R
  - and ISLP uses the `pytorch` package.
- An older text for deep learning is
  - Goodfellow, Yoshua Bengio and Aaron Courville (2016), *Deep Learning*, MIT Press.

5. Regression Trees and Random Forests: Overview

- Regression Trees sequentially split regressors **x** into regions that best predict y
  - e.g., first split is income $<$ or $>$ \$12,000
    second split is on gender if income $>$ \$12,000
    third split is income $<$ or $>$ \$30,000 (if female and income $>$ \$12,000).
- Trees do not predict well
  - due to high variance
  - e.g. split data in two then can get quite different trees
  - e.g. first split determines future splits (a greedy method).
- Better methods are then given
  - bagging (bootstrap averaging)
  - random forests
  - boosting
- Bagging and boosting are general methods (not just for trees).

5.1 Regression Trees

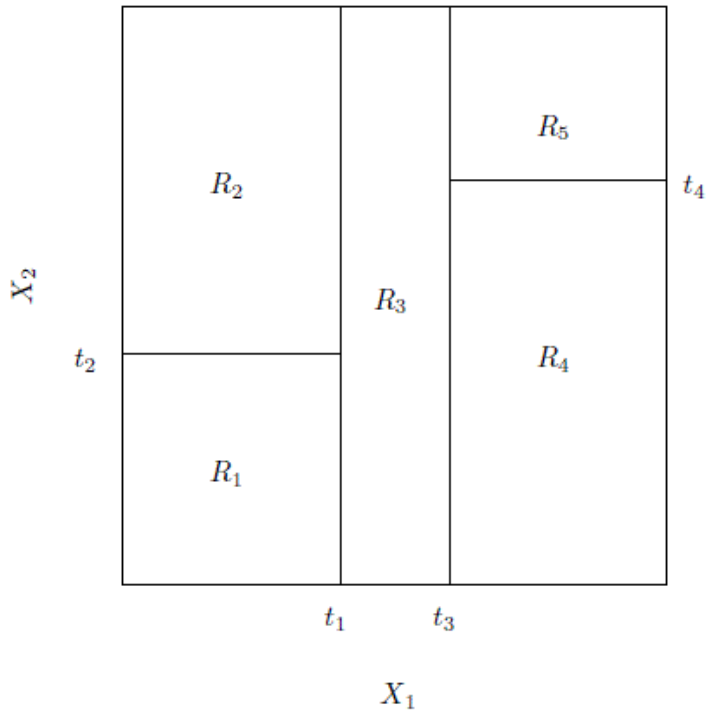
- **Regression trees**
  - sequentially split $\mathbf{x}'s$ into rectangular regions in way that reduces RSS
  - then $\widehat{y}_i$ is the average of $y's$ in the region that $\mathbf{x}_i$ falls in
  - with J blocks RSS$= \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2$.
- Need to determine both the regressor j to split and the split point s.
  - For any regressor j and split point s, define the pair of half-planes $R1(j,s) = \{X | X_j < s\}$ and $R2(j,s) = \{X | X_j \geq s\}$
  - Find the value of j and s that minimize

$$\sum_{i:\mathbf{x}_i \in R1(j,s)} (y_i - \bar{y}_{R1})^2 + \sum_{i:\mathbf{x}_i \in R2(j,s)} (y_i - \bar{y}_{R2})^2$$

    where $\bar{y}_{R1}$ is the mean of y in region $R1$ (and similar for $R2$).
  - Once this first split is found, split both $R1$ and $R2$ and repeat
  - Each split is the one that reduces RSS the most.
  - Stop when e.g. less than five observations in each region.

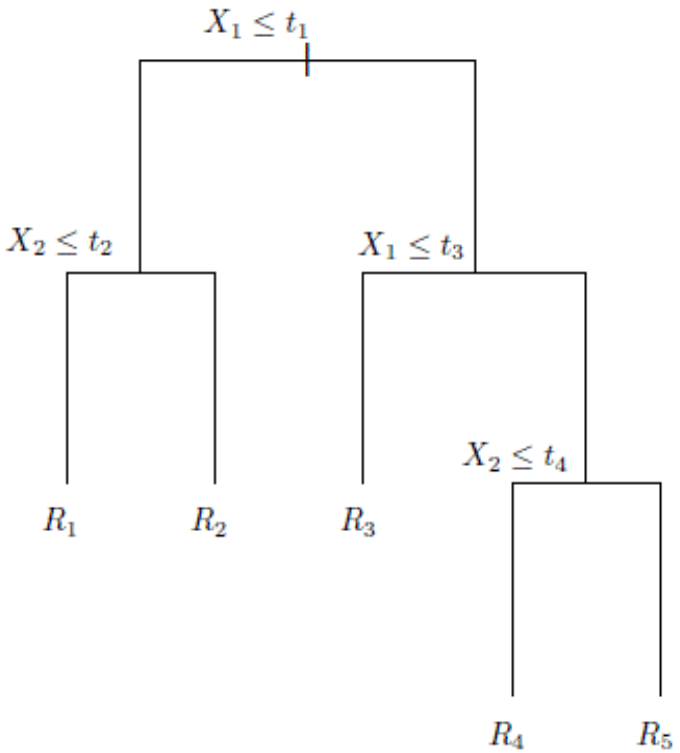
Tree example from ISL2 Figure 8.3 page 332

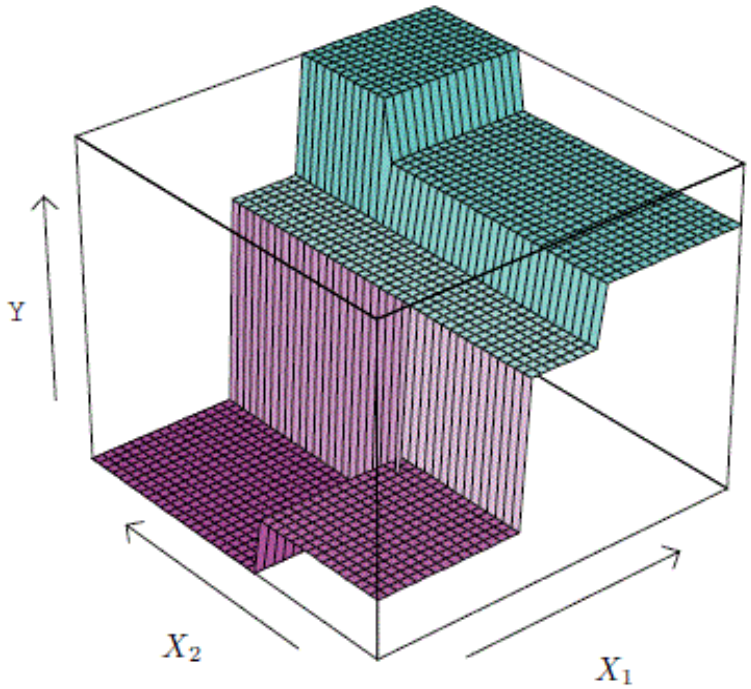
- (1) split X1 in two; (2) split the lowest X1 values on the basis of X2 into R1 and R2; (3) split the highest X1 values into two regions (R3 and R4/R5); (4) split the highest X1 values on the basis of X2 into R4 and R5.



Tree example from ISL (continued)

- The left figure gives the tree.
- The right figure shows the predicted values of y.



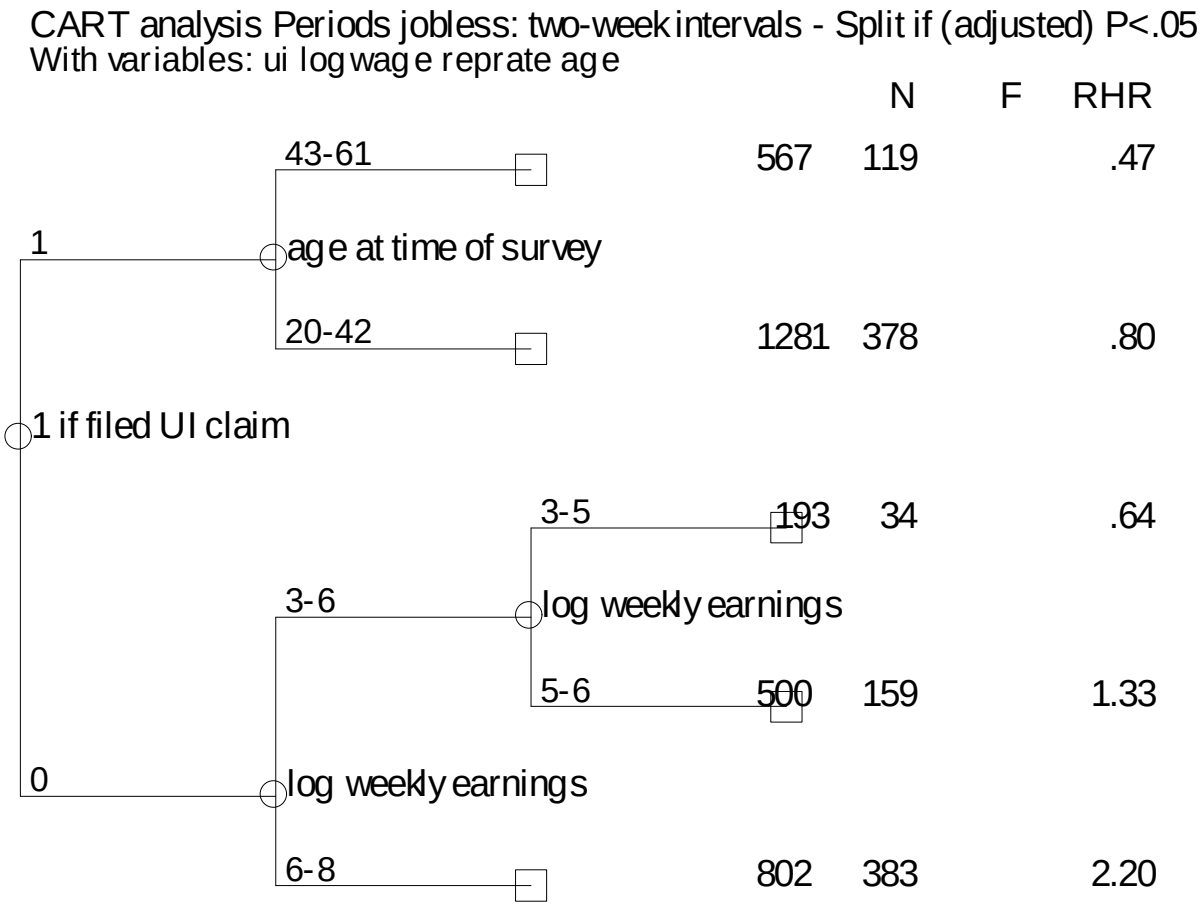


Regression tree (continued)

- The model is of form $f(\mathbf{x}) = \sum_{j=1}^{J} c_j \times \mathbf{1}[\mathbf{x} \in R_j]$
  - essentially OLS on a set of data-determined indicator variables.
- The approach is a topdown greedy approach
  - top down as start with top of the tree
  - **greedy** as at each step the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step
  - so a seemingly worthless split early on in the tree might have been followed by a very good split later on.
- So deliberately **overfit** and then **prune** back
  - use cost complexity pruning (or weakest link pruning)
  - this adds a penalty in the number of terminal nodes and uses CV on this
  - see ISL2 equation (8.4).

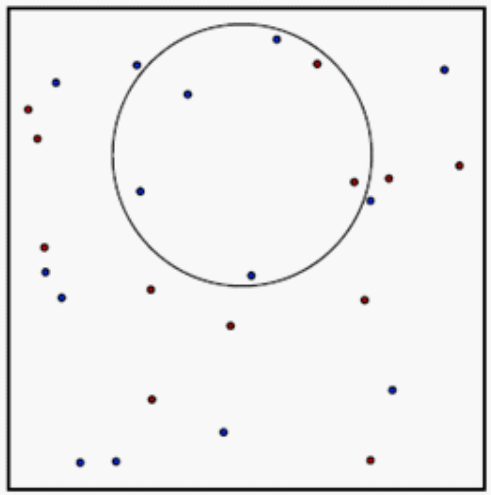
Regression tree example

- This example is for duration data using Stata add-on `cart`
  - I used it merely to illustrate what a tree looks like.

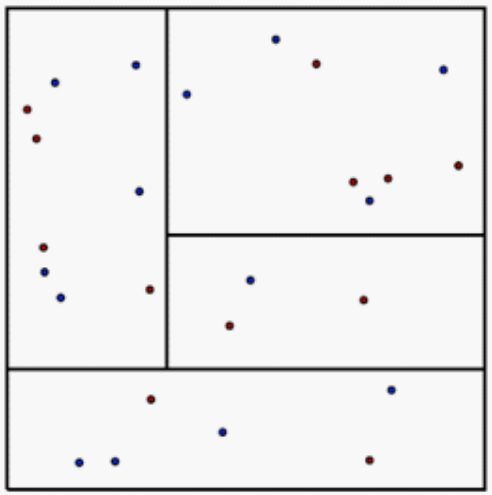


Tree as alternative to k-NN or kernel regression

- Figure from Athey and Imbens (2019), "Machine Learning Methods Economists Should Know About"
  - axes are x_1 and x_2
  - note that tree used explanation of y in determining neighbors
  - tree may not do so well near boundaries of region
    - ★ random forests form many trees so not always at boundary.



Euclidean neighborhood, for KNN matching.



Tree-based neighborhood.

Improvements to regression trees

- Regression trees are easy to understand if there are few regressors.
- But they do not predict as well as methods given so far
  - due to high variance in the predictions
  - e.g. split data in two then can get quite different trees and predictions.
- Better methods are given next
  - bagging (bootstrap aggregating)
    - ★ averages regression trees over many samples
    - ★ benefit: averaging reduces variance $\text{Var}(\overline{Y}) \leq \text{Var}(Y)$.
  - random forests
    - ★ additionally uses only a random subset of regressors at each split
    - ★ benefit: the predictions being averaged are less correlated with each other so $\text{Var}(\overline{Y})$ is less.
  - boosting
    - ★ trees build on the fit from preceding trees.

5.2 Bagging (Bootstrap Aggregating)

- **Bagging** is a general method for improving prediction that works especially well for regression trees.
- Idea is that averaging reduces variance (in nonlinear models).
- So average regression trees over many samples
  - the different samples are obtained by bootstrap resample with replacement (so not completely independent of each other)
  - for each sample obtain a large tree and prediction $\widehat{f}_b(\mathbf{x})$.
  - average all these predictions: $\widehat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\widehat{f}_b(\mathbf{x})$.
- Get test sample error by using out-of-bag (OOB) observations not in the bootstrap sample
  - $\Pr[i^{th} \text{ obs not in resample}] = (1-\frac{1}{n})^n \rightarrow e^{-1} = 0.368 \simeq 1/3$.
  - this replaces using a validation dataset or using cross validation.
- Interpretation of trees is now difficult so
  - record the total amount that RSS is decreased due to splits over a given predictor, averaged over all B trees.
  - a large value indicates an important predictor.

5.3 Random Forests

- The B bagging estimates are correlated
  - e.g. if a regressor is important it will appear near the top of the tree in each bootstrap sample.
  - the trees look similar from one resample to the next.
- Random forests get bootstrap resamples (like bagging)
  - but use only a random sample of $m < p$ predictors in deciding each split (within each bootstrap sample)
  - usually $m \simeq \sqrt{p}$
  - this reduces correlation across bootstrap resamples and reduces overfitting
  - simple bagging is random forest with $m = p$.
- Stata add-on command `rforest` implements random forests (and bagging)
  - Matthias Schonlau and Rosie Zou (2020), "The random forest algorithm for statistical learning," *The Stata Journal*, 3-29.
- Python use SciKit Learn module `RandomForestRegressor`.

Random Forests (continued)

- In practice choose the best predicting model from grid search over a range of values for
  - number of trees: e.g. (50, 75, ..., 200)
  - number of features: e.g. (0.2, 0.4, ..., 1.0)
  - maximum tree depth: e.g. (3, 5, 7, 9)
  - sample_sizes as fraction of n: e.g. (0.3, 0.5, 0.8).
- Random forests are related to kernel and k-nearest neighbors
  - as use a weighted average of nearby observations
  - but with a data-driven way of determining which nearby observations get weight
  - see Lin and Jeon (JASA, 2006).
- Susan Athey and coauthors are big on random forests.

5.4 Boosting

- Boosting is also a general method for improving prediction.
- Regression trees use a greedy algorithm.
- Boosting uses a slower algorithm to generate a sequence of trees
  - each tree is grown using information from previously grown trees
  - and is fit on a modified version of the original data set
  - boosting does not involve bootstrap sampling.
- Specifically (with λ a penalty parameter)
  - given current model b fit a decision tree to model $b's$ residuals (rather than the outcome y)
  - then update $\widehat{f}(\mathbf{x}) =$ previous $\widehat{f}(\mathbf{x}) + \lambda \widehat{f}^b(\mathbf{x})$
  - then update the residuals $r_i =$ previous $r_i - \lambda \widehat{f}^b(\mathbf{x}_i)$
  - the boosted model is $\widehat{f}(\mathbf{x}) = \sum_{b=1}^{B} \lambda \widehat{f}^b(\mathbf{x}_i)$, a weighted sum of trees.
- Stata add-on `boost` includes file `boost64.dll` that needs to be manually copied into `c:\ado\plus`
- `pylearn` does trees, random forests and neural nets directly in Stata and requires installation of Python and the Python scikit-learn library.

5.5 Bayesian Additive Regression Trees

- Like boosting use only the original data
  - whereas random forests draw random samples of the data.
- Start with K trees and in each tree $\widehat{f}^{(1)}(\mathbf{x}) = \bar{y}$.
- Then for each iteration $b = 1, ..., B$
  - for each tree k and each observation i create the partial residual which is y_i minus the predictions from all other trees

$$r_i = y_i - \sum_{k'<k} \widehat{f}_k^{(b)}(\mathbf{x}) - \sum_{k'>k} \widehat{f}_k^{(b)}(\mathbf{x})$$

  - fit a new tree $\widehat{f}_k^{(b)}(\mathbf{x})$ to r_i by randomly perturbing tree k from the previous iteration $\widehat{f}_k^{(b-1)}(\mathbf{x})$ and favor perturbations that improve the fit
  - compute $\widehat{f}_k^{(b)}(\mathbf{x}) = \sum_{k=1}^{K} \widehat{f}_k^{(b-1)}(\mathbf{x})$
- Finally $\widehat{f}_k^{(b)}(x) = \frac{1}{B-L} \sum_{k=1}^{K} \widehat{f}_k^{(b-1)}(\mathbf{x})$ where L is the number of burn-in reps.
- It is an MCMC algorithm.
- ISL2 Section 8.2.4 has details.

5.6 ML Terminology

- ML terminology such as that used by the Python SciKit-Learn package
  - **label** is the dependent variable (y)
  - **features** are the explanatory variables $(\mathbf{X})$
  - **samples** is the number of observations (n)
  - **estimators** is the number of trees in a forest
  - **node** is the split point or partition point
  - **pure node** is a node with all y values the same
    - ★ in classification all training observations belong to the same class
    - ★ if all end nodes were pure the training data is perfectly explained.
  - **impurity measure** is the value of the loss function e.g. MSE
  - **importance measure** is a measure of the relative importance of each feature
    - ★ in each tree collect how on average it decreases the impurity due to splits on that feature and average over all trees.

6. Prediction Example

- Go through MUS2 Section 28.6 Prediction Example in detail
- Use code in `ML_2022_part4.do` with data set `mus203mepsmedexp.dta`
- Predict using 7 methods
  - OLS with no interactions
  - OLS with interactions
  - LASSO with penalized coefficients
  - Post LASSO (OLS with variables selected by LASSO)
  - Neural network (add-on brain)
  - Random forest (add-on randomforest)
  - boost
- Fit on 80% of sample. See how predicts out of sample.

Data

- Same MEPS data for 2013 on 65-90 year-olds. as in part 3.
- y is `ltotexp` = log total annual medical expenditure
- **x is** 5 continuous variables and 14 binary variables and $N = 2955$
  - same as part 3 except include `suppins` with other binary variables.
- With interactions get 188 unique variables.

```
. * Data for prediction example: 5 continuous and 14 binary variables
. qui use mus203mepsmedexp.dta, clear

. keep if ltotexp != .
(109 observations deleted)

. global xlist income educyr age famsze totchr

. global dlist suppins female white hisp marry northe mwest south ///
>     msa phylim actlim injury priolist hvgg

. global rlist c.($xlist)##c.($xlist) i.($dlist) c.($xlist)#i.($dlist)
```

```
. * Summary statistics for full sample
. summarize ltotexp $xlist $dlist
```

Variable	Obs	Mean	Std. Dev.	Min	Max
ltotexp	2,955	8.059866	1.367592	1.098612	11.74094
income	2,955	22.68353	22.60988	-1	312.46
educyr	2,955	11.82809	3.405095	0	17
age	2,955	74.24535	6.375975	65	90
famsze	2,955	1.890694	.9644483	1	13
totchr	2,955	1.808799	1.294613	0	7
suppins	2,955	.5915398	.4916322	0	1
female	2,955	.5840948	.4929608	0	1
white	2,955	.9736041	.1603368	0	1
hisp	2,955	.0812183	.2732163	0	1
marry	2,955	.5583756	.4966646	0	1
northe	2,955	.1536379	.3606623	0	1
mwest	2,955	.2318105	.42206	0	1
south	2,955	.3922166	.4883272	0	1
msa	2,955	.7397631	.438838	0	1
phylim	2,955	.4362098	.4959981	0	1
actlim	2,955	.2879865	.4529014	0	1
injury	2,955	.2020305	.4015828	0	1
priolist	2,955	.8240271	.3808616	0	1
hvgg	2,955	.6013536	.4897026	0	1

```
. * OLS for full sample
. regress ltotexp $xlist $dlist, vce(robust) noheader
```

ltotexp	Coef.	Robust Std. Err.	t	P>|t|	[95% Conf.	Interval]
income	.0007411	.0010967	0.68	0.499	-.0014092	.0028914
educyr	.0415116	.0076743	5.41	0.000	.0264641	.0565591
age	.0042834	.0037527	1.14	0.254	-.0030749	.0116416
famsze	-.0669498	.0261385	-2.56	0.010	-.1182014	-.0156982
totchr	.3238205	.0188741	17.16	0.000	.2868126	.3608283
suppins	.1706101	.0469033	3.64	0.000	.0786434	.2625768
female	-.0508783	.0468787	-1.09	0.278	-.1427968	.0410403
white	.1858472	.1325621	1.40	0.161	-.074077	.4457713
hisp	-.1101501	.0904202	-1.22	0.223	-.2874435	.0671433
marry	.1751016	.0516199	3.39	0.001	.0738868	.2763164
northe	.2736686	.0713944	3.83	0.000	.1336804	.4136567
mwest	.3051208	.0689651	4.42	0.000	.169896	.4403456
south	.1957967	.0593267	3.30	0.001	.0794705	.3121229
msa	.0709307	.0512069	1.39	0.166	-.0294743	.1713357
phylim	.268737	.0567284	4.74	0.000	.1575054	.3799685
actlim	.3661458	.0636335	5.75	0.000	.241375	.4909165
injury	.1664688	.0539137	3.09	0.002	.0607564	.2721813
priolist	.4361775	.0689187	6.33	0.000	.3010436	.5713114
hvgg	-.0959803	.0463345	-2.07	0.038	-.1868316	-.0051289
_cons	5.633868	.3425158	16.45	0.000	4.962272	6.305463

- We will use 80% of sample for training and remaining 20% for out-of-sample evaluation

```
. * Split the sample with 80% in training sample
. splitsample ltotexp, generate(train) split(1 4) values(0 1) rseed(10101)

. tabulate train
```

train	Freq.	Percent	Cum.
0	591	20.00	20.00
1	2,364	80.00	100.00
Total	2,955	100.00	

- Then go through code in ML_2022_part4.do

Predictions from Various Models

```
* OLS with 19 regressors
regress ltotexp $xlist $dlist if train==1, noheader vce(robust)
qui predict y_small
* OLS with 188 potential regressors and 104 estimated
qui regress ltotexp $rlist if train==1
qui predict y_full
* LASSO with 188 potential regressors leads to 32 selected
qui lasso linear ltotexp $rlist if train==1, selection(adaptive) rseed(10101) nolog
qui predict y_laspen // use penalized coefficients
qui predict y_laspost, postselection // use post selection OLS coeffs
```

Predictions from Various Models (continued)

```
* Principal components using the first 5 principal components of 19 variables
qui pca $xlist $dlist if train==1
qui predict pc*
qui regress ltotexp pc1-pc5 if train==1
qui predict y_pca
* Neural network: 19 variables one hidden layers with 10 units
* This did not work on my latest computer
brain define, input($xlist $dlist) output(ltotexp) hidden(10)
qui brain train if train==1, iter(500) eta(2) // eta>1 uses SGD
brain think y_neural
* Random forest with 19 variables
qui rforest ltotexp $xlist $dlist if train==1, ///
      type(reg) iter(200) depth(10) lsize(5)
qui predict y_ranfor
```

Predictions from Various Models (continued)

- Compute training MSE and test MSE

```
foreach var of varlist y_noreg y_small y_full y_laspen ///
    y_laspost y_pca y_neural y_ranfor {
    qui gen `var'errorsq = (`var' - ltotexp)^2
    qui sum `var'errorsq if train == 1
    scalar mse`var'train = r(mean)
    qui sum `var'errorsq if train == 0
    qui scalar mse`var'test = r(mean)
    display "Predictor: " "`var'" _col(21) ///
        " Train MSE = " %5.3f mse`var'train ///
        " Test MSE = " %5.3f mse`var'test
 }
```

Predictions from Various Models (continued)

- Training sample: Flexible models - random forest and neural networks did best.
- Test sample: Simpler models - LASSO and small model OLS did best.

```
Predictor: y_noreg   Train MSE = 1.821  Test MSE = 2.063
Predictor: y_small   Train MSE = 1.339  Test MSE = 1.492
Predictor: y_full    Train MSE = 1.262  Test MSE = 1.509
Predictor: y_laspen  Train MSE = 1.298  Test MSE = 1.491
Predictor: y_laspost Train MSE = 1.297  Test MSE = 1.493
Predictor: y_pca     Train MSE = 1.397  Test MSE = 1.545
Predictor: y_neural  Train MSE = 1.211  Test MSE = 1.808
Predictor: y_ranfor  Train MSE = 1.047  Test MSE = 1.574
```

7. Prediction for Economics

- Hal Varian (2014) has early survey.
- Mullainathan and Spiess (2017)
  - summarizes various
  - has good application to housing prices(already presented)
  - has good summary of recent economics ML applications.

7.1 Hal Varian 2014 Survey

- Hal Varian (2014), "Big Data: New Tricks for Econometrics," JEP, Spring, 3-28.
- Surveys tools for handling big data
  - file system for files split into large blocks across computers
    - ★ Google file system (Google), Hadoop file system
  - database management system to handle large amounts of data across many computers
    - ★ Bigtable (Google), Cassandra
  - accessing and manipulating big data sets across many computers
    - ★ MapReduce (Google), Hadoop.
  - language for MapReduce / Hadoop
    - ★ Sawzall (Google), Pig
  - Computer language for parallel processing
    - ★ Go (Google - open source)
  - simplified structured query language (SQL) for data enquiries
    - ★ Dremel, Big Query (Google), Hive, Drill, Impala.

Hal Varian (continued)

- Surveys methods
  - article discusses k-fold CV, trees, lasso, ...
  - small discussion of causality and prediction
  - (note that a classic fail is Google flu trends)
  - for references mentions ESL and ISL.
- While dated it is still worth reading.

7.2 Summary of Machine Learning Algorithms

- From Mullainathan and Spiess (2017)

Table 2

Some Machine Learning Algorithms

Function class $\mathcal{F}$ (and its parametrization)	*Regularizer $R(f)$*
Global/parametric predictors	
Linear $\beta' x$ (and generalizations)	Subset selection $\|\|\beta\|\|_0 = \sum_{j=1}^{k} 1_{\beta_j \neq 0}$
	LASSO $\|\|\beta\|\|_1 = \sum_{j=1}^{k} \|\beta_j\|$
	Ridge $\|\|\beta\|\|_2^2 = \sum_{j=1}^{k} \beta_j^2$
	Elastic net $\alpha\|\|\beta\|\|_1 + (1-\alpha)\|\|\beta\|\|_2^2$
Local/nonparametric predictors	
Decision/regression trees	Depth, number of nodes/leaves, minimal leaf size, information gain at splits
Random forest (linear combination of trees)	Number of trees, number of variables used in each tree, size of bootstrap sample, complexity of trees (see above)
Nearest neighbors	Number of neighbors
Kernel regression	Kernel bandwidth

Table 2 (continued)

Mixed predictors	
Deep learning, neural nets, convolutional neural networks	Number of levels, number of neurons per level, connectivity between neurons
Splines	Number of knots, order
Combined predictors	
Bagging: unweighted average of predictors from bootstrap draws	Number of draws, size of bootstrap samples (and individual regularization parameters)
Boosting: linear combination of predictions of residual	Learning rate, number of iterations (and individual regularization parameters)
Ensemble: weighted combination of different predictors	Ensemble weights (and individual regularization parameters)

7.3 Some Thoughts on ML Prediction

- Clearly there are many decisions to make in implementation
  - how are features converted into x's
  - tuning parameter values
  - which ML method to use
  - even more with an ensemble forecast.
- For commercial use this may not matter
  - all that matters is that predict well enough.
- But for published research we want reproducibility
  - At the very least document exactly what you did
  - provide all code (and data if it is publicly available)
  - keep this in mind at the time you are doing the project.
- For public policy we prefer some understanding of the black box
  - this may be impossible.

8. Software for Machine Learning

- This list will change over time and is not necessarily the best
  - There are over 20,000 R packages in CRAN
    and over 350,000 Python packages (most are not data-oriented).
- Python libraries presented in Geron book and ISLPython
  - `scikit-learn` for most ML methods including trees and random forests
  - `keras` and `tensorflow` and `pytorch` for neural networks
  - code at `https://github.com/ageron/handson-ml3`.
- R packages used in ISLR2 include
  - `spline` library for splines
  - `torch` package for neural networks
  - `tree` library and `randomForest` package for tree-based methods.
- Stata commands include
  - `npregress` for local regression, series regression and splines
  - `pca` for principal components

Software (continued)

- Stata add-ons include
  - `brain` for neural networks (very basic)
  - `rforest` for trees and random forests
  - `crtrees` for trees and random forests
  - `srtrees` wrapper for R commands for trees and random forests
  - `r_ml_stata.ado` and `r_ml_stata.ado` wrappers for ML methods including neural networks and trees in the python `scikit-learn` library
    - ★ https://sites.google.com/view/giovannicerulli/machine-learning-in-stata
  - `pylearn` is a wrapper for trees and random forests in python `scikit-learn` library

9. References

- Geron: Aurelien Geron (2022), Hands-On Machine Learning with Scikit-Learn, Keras and Tensor Flow, Third edition, O'Reilly
  - excellent book using Python for ML writtten by a computer scientist.
- ISLR2: Gareth James, Daniela Witten, Trevor Hastie and Robert Tibsharani (2021), *An Introduction to Statistical Learning: with Applications in R*, 2nd Ed., Springer.
- ISLP: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibsharani and Jonathan Taylor (2023), *An Introduction to Statistical Learning: with Applications in Python*, Springer.
  - PDF and $40 softcover book at Springer Mycopy
- ESL: Trevor Hastie, Robert Tibsharani and Jerome Friedman (2009), *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, Springer.
  - PDF and $40 softcover book at https://link.springer.com/book/10.1007/978-0-387-84858-7
- MUS2: Chapter 28 "Machine Learning for prediction and inference" and Chapter 27 "Semiparametric Regression" in A. Colin Cameron and Pravin K. Trivedi

References (continued)

- EH: Bradley Efron and Trevor Hastie (2016), *Computer Age Statistical Inference: Algorithms, Evidence and Data Science*, Cambridge University Press.
- Ian Goodfellow, Yoshua Bengio, A. Courville (2016), Deep Learning, MIT Press.
- Yi Lin and Yongho Jeon (2006), "Random Forests and Adaptive Nearest Neighbors," JASA, 578-590.
- Sendhil Mullainathan and J. Spiess (2017): "Machine Learning: An Applied Econometric Approach", Journal of Economic Perspectives, Spring, 87-106.
- Jon Kleinberg, H. Lakkaraju, Jure Leskovec, Jens Ludwig, Sendhil Mullainathan (2018), "Human Decisions and Machine Predictions", Quarterly Journal of Economics, 237-293.
- Hal Varian (2014), "Big Data: New Tricks for Econometrics", Journal of Economic Perspectives, Spring, 3-28.
- Athey and Imbens (2019), "Machine Learning Methods Economists Should Know About", Annual Review of Economics, 685-725.
- A great lengthy article on neural nets leading to Google Translate https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html