Machine Learning Principles: A Helicopter Tour

Christian Hansen - University of Chicago

Introduction

Data

The world's most valuable resource is no longer oil, but data.

Figure 1: From: The Economist, May 6th, 2017.

But, data has no value, per se.

Crucial question: What can be learned from the data?

Main tasks: **Prediction** and (causal) **inference**

Taxonomy of Data Sets

Notation: n = number of observations; p = number of variables

1. "Tall data"
   - $n \gg p$ or big n, small p
   - computationally demanding
   - **"traditional" statistical theory applies**

2. "High-dimensional data" or "wide data"
   - $p/n \not\to 0$ or small n, big p
     - alternatively, "flexible" models - nonparametrics, semiparametrics
   - potentially computationally demanding
   - **non-standard theory**
   - **honest inference usually belongs here**

3. "Big Data"
   - large np
   - computationally demanding
   - encompasses 1. and 2.

Focus in these notes is High Dimensional Models for Supervised Learning

- potentially computationally intensive
- BUT our emphasis is NOT on computational issues
  - though we will exclusively talk about computationally tractable methods
- Supervised Learning: Predict target Y with input X
  - When we talk about prediction, we are talking about making use of a stable association to make a forecast.
  - Stable associations $\Rightarrow$ learning about prediction rules directly from data
  - (Aside) Unsupervised Learning: Find structure within X with no specific target

Lecture Goal:

- Introduce basic concepts and methods popular in statistical learning/machine learning/data mining
  - ML/AI: use computer to find patterns rather than rely on researcher specification

Popular ML Methods

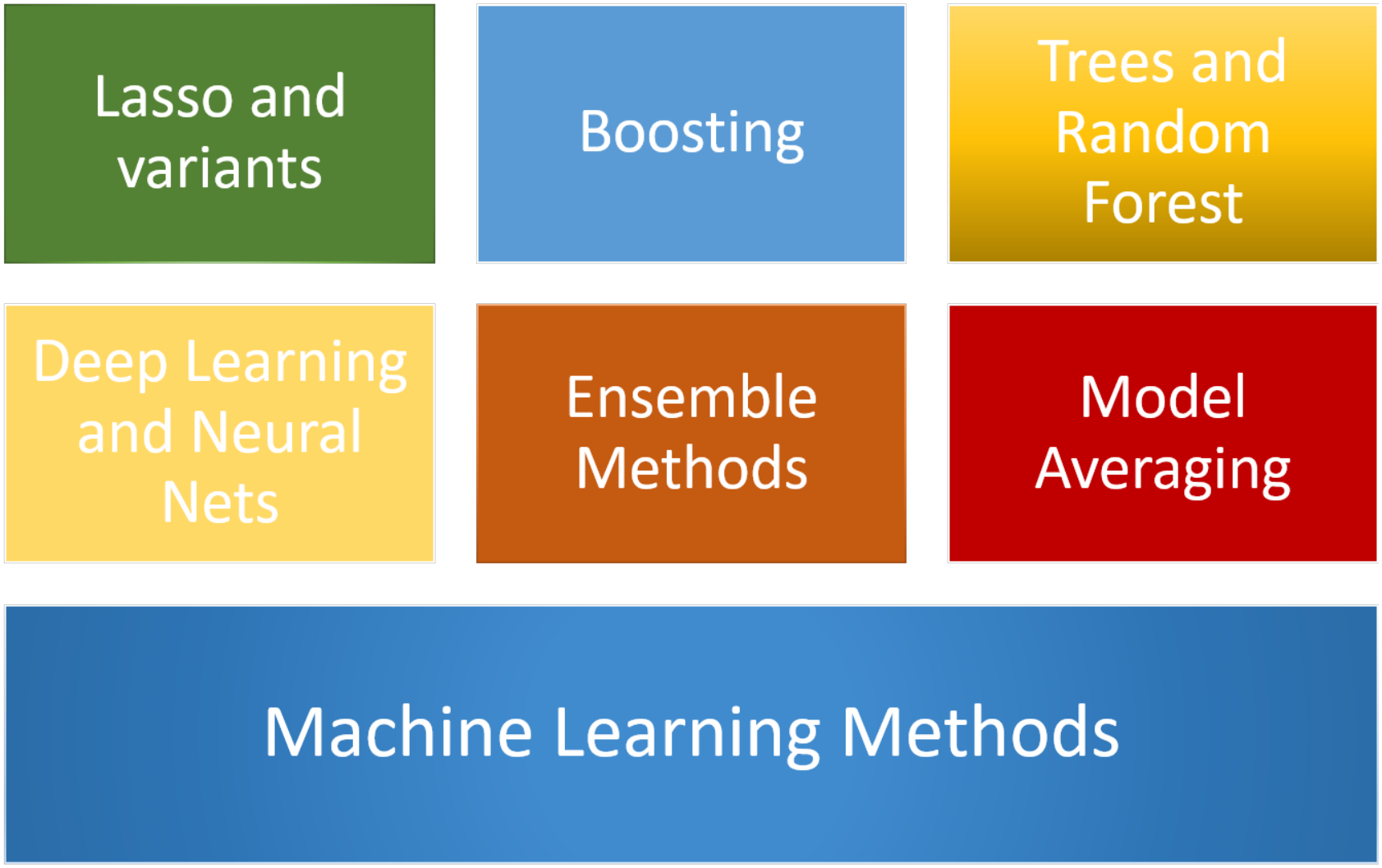


Figure 2: We'll provide a high-level introduction to these approaches [Mullainathan and Spiess (2017), Athey and Imbens (2019)]

The Prediction Problem

Predictive Models

Supervised Learning

A core statistical learning task is creating a rule for predicting the (as yet unseen) value of a target (Y) given observed characteristics X (i.e. prediction/forecasting).

Useful to think about relation between

- target: Y and
- p-dimensional predictor variable: $X = (X_1, ..., X_p)$

$$\underbrace{Y}_{\text{target}} = \underbrace{f(X)}_{\text{signal}} + \underbrace{\varepsilon}_{\text{noise}}$$

Goal: Learn $f(\cdot)$ from the data in a way that yields "generalizable" forecasts

- Produce a forecast rule that minimizes expected forecast loss
- Will focus on minimizing squared error loss: MSE = $\mathrm{E}[(Y - f(X))^2]$
  - Recall $\mathrm{E}[Y|X]$ provides the squared error loss minimizing prediction rule for Y
  - Nothing fundamentally different in classification problems (problems with categorical Y)

Setting

Assume target prediction rule is $f(X) = \mathrm{E}[Y|X]$, so "model" is

$$Y = f(X) + \varepsilon; \qquad \mathrm{E}[\varepsilon|X] = 0$$

Problem: Given a data set of past observations (y_i, x_i), $i = 1, ..., n$

- y_i: observed outcome
- $x_i = (x_{i1}, ..., x_{ip})'$: $p \times 1$ observed predictor vector

produce an estimator $\widehat{f}$ of f to form predictions $\widehat{y} = \widehat{f}(x)$.

So, how should we specify and estimate f?

- Estimation seems "easy": minimize sample analog of forecast loss. E.g.

$$\widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_i (y_i - f(x_i))^2$$

- Specifying $\mathcal{F}$ seems more difficult.

Financial Asset Prediction

Simple illustration to fix concepts: Suppose we want to predict household level net financial assets using income with the conditional mean. (Aside: A conditional median might be preferable.)

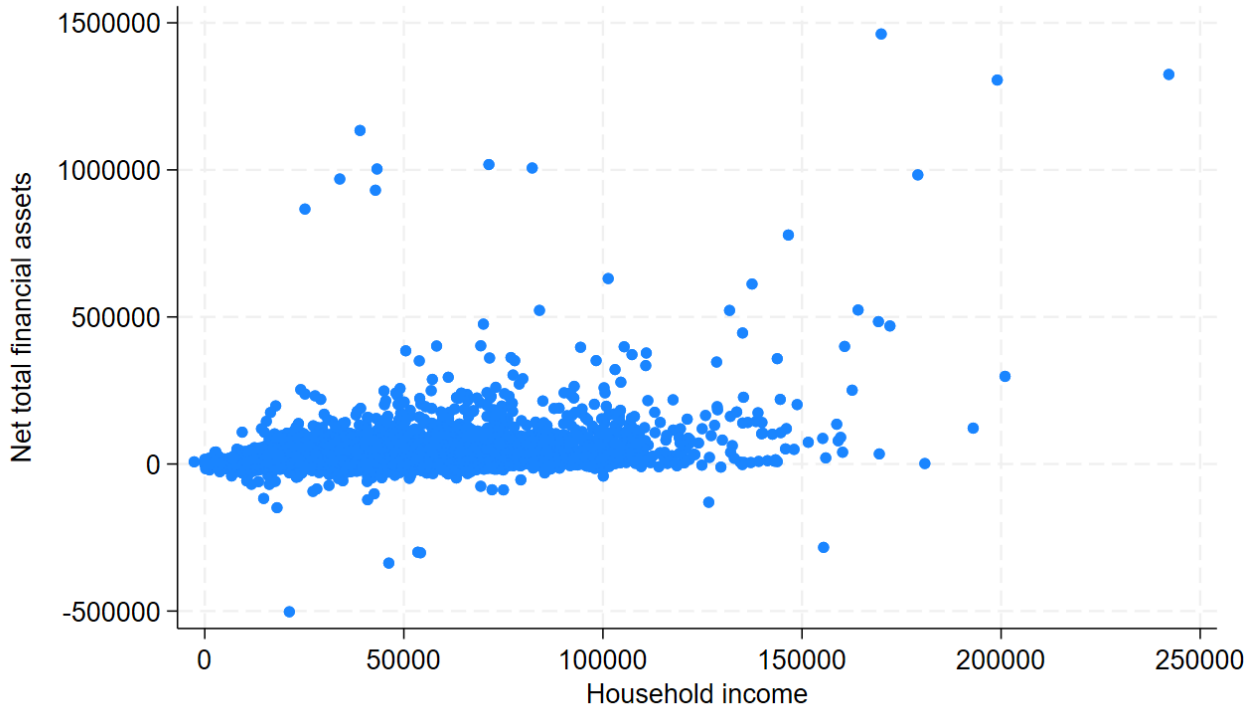


Figure 3: Data from (subset of) SIPP sample for 401(k) example.

Financial Asset Prediction: Simple Model

A simple prediction rule:

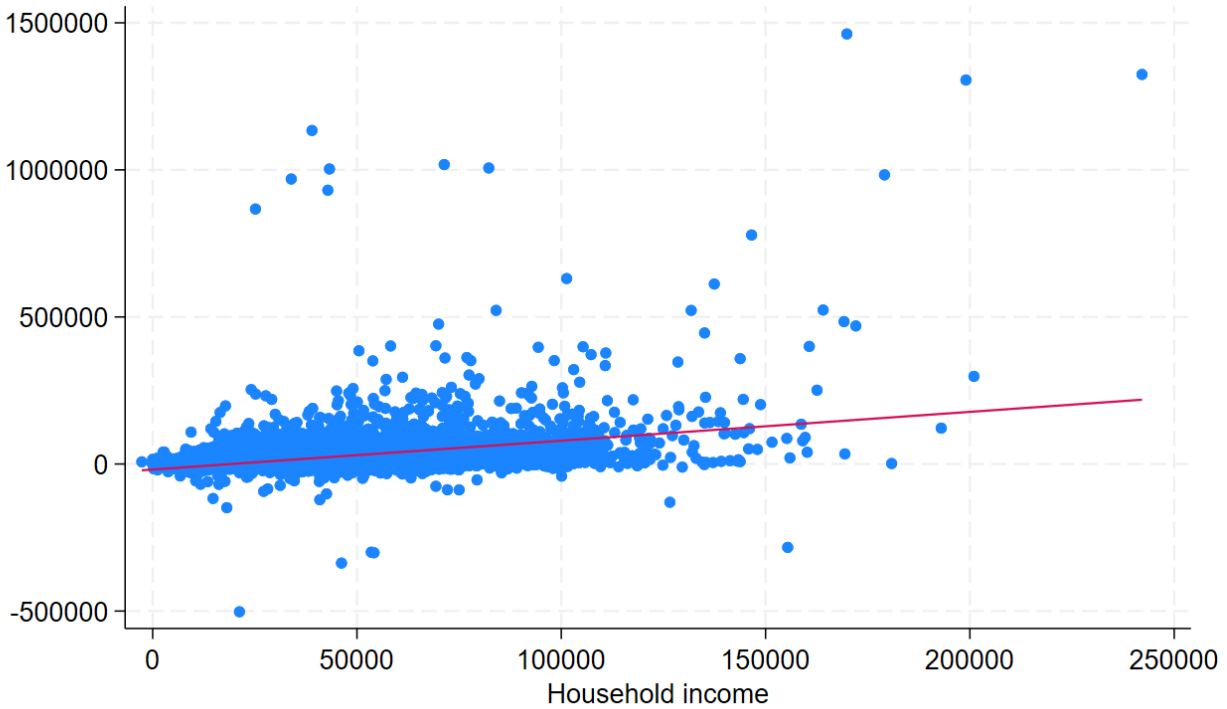


Figure 4: Linear prediction rule. $R^2 = 0.161$

Is this too simple? Are we missing part of the pattern?

Financial Asset Prediction: Complex Model

A complex prediction rule:

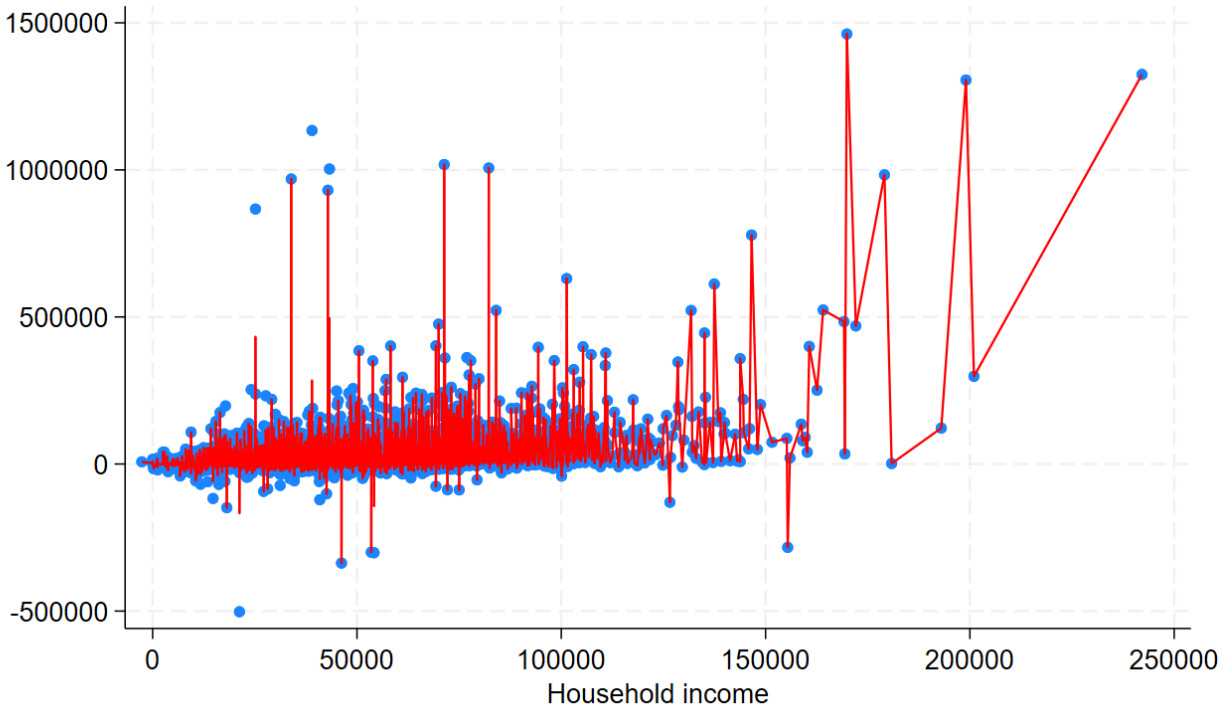


Figure 5: Complex prediction rule. $R^2 = 0.886$

Is this too complex? Will this pattern generalize?

Financial Asset Prediction: Intermediate Model

An “intermediate” prediction rule:

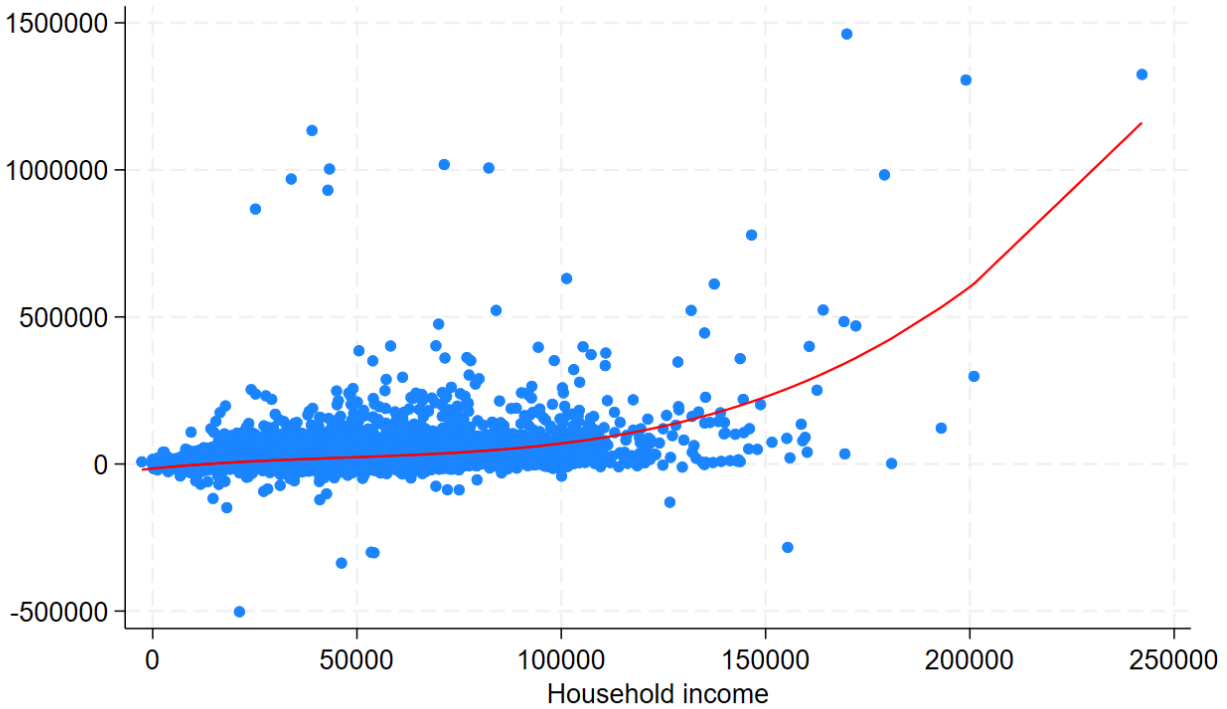


Figure 6: Intermediate prediction rule. $R^2 = 0.244$

Will this pattern generalize? Can we do better?

Overfitting/Underfitting

Fundamental problem:

- Find model which is not so simple it misses patterns in data - I.e. avoid underfitting
- Find model which is not so flexible it captures non-generalizable patterns in data - I.e. avoid overfitting

How do we decide?

With one input and moderate numbers of observations, we can use our eyes and intuition. We want something more general.

Assessing Model Accuracy

Prediction Loss

Remember: Goal is to find a rule that does a good job predicting outcome for new observation.

Consider a new/unseen observation: (x_0, y_0) and a prediction rule learned on previous observations $\widehat{f}(\cdot)$

Forecast loss for new observation:

$$MSE = \mathrm{E}[(y_0 - \widehat{f}(x_0))^2]$$

We would like to assess prediction rules on their MSE.

- Could use different losses

The Bias-Variance Trade-Off

Decomposition of the MSE:

$$\begin{aligned} MSE(x_0) &= E(y_0 - \widehat{f}(x_0))^2 \\ &= Var(\widehat{f}(x_0)) + Bias^2(\widehat{f}(x_0)) + Var(\varepsilon_0) \end{aligned}$$

"MSE = Variance + Bias squared + non-reducible variance of error"

The tension between overfitting and underfitting is often referred to as the "bias/variance tradeoff"

- Bias and variance here refer to bias of the prediction rule under repeated sampling.
- Practical Implication: Prediction rule from your sample is likely to produce systematic mistakes out-of-sample if it has high bias OR variance

The Bias-Variance Trade-Off

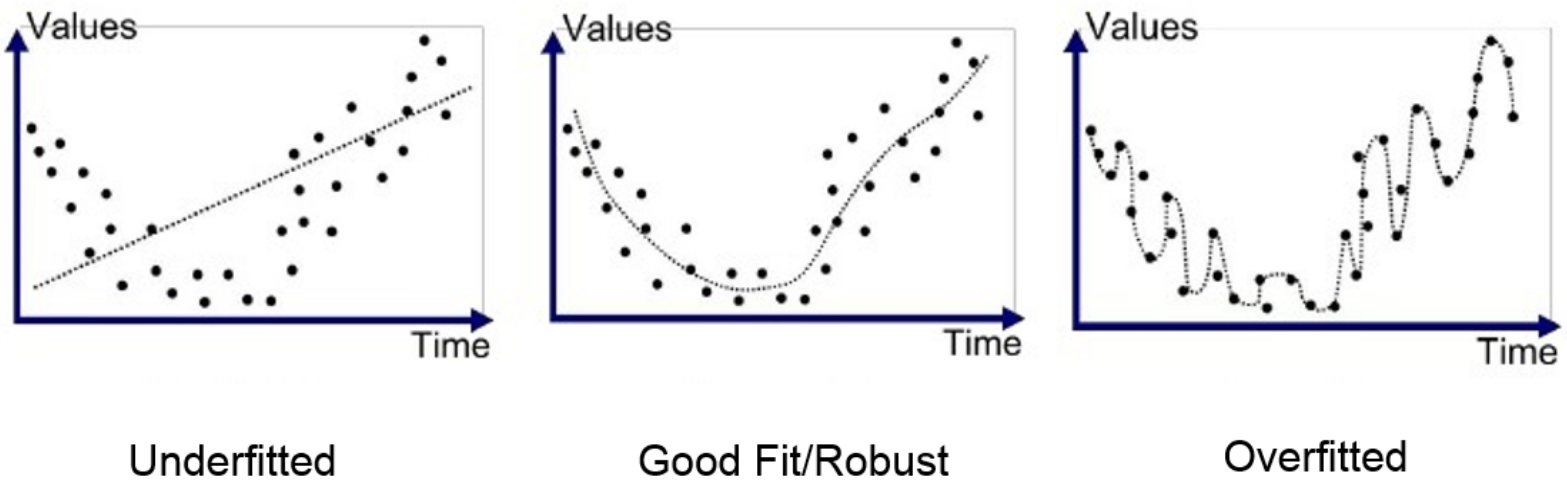


Figure 7: The fundamental tension in predictive modeling

Bias/Variance Tradeoff

As model complexity increases, the bias and variance of learned prediction rules tend to move in opposite directions - with bias decreasing and variance increasing. A good prediction rule tries to balance these forces to create a low bias, stable predictor.

In-Sample Fit

Of course, knowing $E(y_0 - \widehat{f}(x_0))^2$ is infeasible.

Want a feasible way to estimate quality of fit.

- In-sample analogs offer an obvious solution:

$$\text{(Within-sample) MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{f}(x_i))^2$$

$$\text{(Within-sample) } R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \widehat{f}(x_i))^2}{\sum_{i=1}^{n}(y_i)^2}$$

- Conventional to define R^2 with $\sum_{i=1}^{n}(y_i - \bar{y})^2$ in the denominator
- Can also use any other "baseline" model in denominator for "R^2"-like measure of improvement relative to baseline

REMEMBER: Goal is "out-of-sample" fit - not "in-sample" fit

- "in-sample fit" $\approx$ "out-of-sample" fit if n is large relative to "model complexity" but in-sample fit not a good guide with complex models
  - standard corrections (e.g. adjusted R^2) are better but also fail with complex models

Out-of-sample fit

Do we really care about explaining what we've already seen?

What matters is accuracy out-of-sample

If we got a bunch of new data NOT USED TO ESTIMATE MODEL (say $\{x_i^o, y_i^o\}_{i=1}^m$), could use this data to evaluate models:

$$\text{(Out-of-Sample) MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i^o - \widehat{y}_i^o)^2 = \frac{1}{m}\sum_{i=1}^{m}(y_i^o - \widehat{f}(x_i^o))^2$$

$$\text{(Out-of-Sample) } R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i^o - \widehat{f}(x_i^o))^2}{\sum_{i=1}^{n}(y_i^o)^2}$$

Model Validation

Don't really have data we haven't seen...

Key Idea: Use sample to replicate forecasting environment by splitting data to estimate out-of-sample predictive ability

Split the data:

- Training Sample - data used to estimate prediction rule(s)
- Testing Sample - data used to test estimated rule(s) on NEW data
    - AKA "validation" or "hold-out" sample

Model Validation

Training data: $\{y_i^T, x_i^T\}_{i=1}^{n_T} \rightarrow \widehat{f}_T(\cdot)$

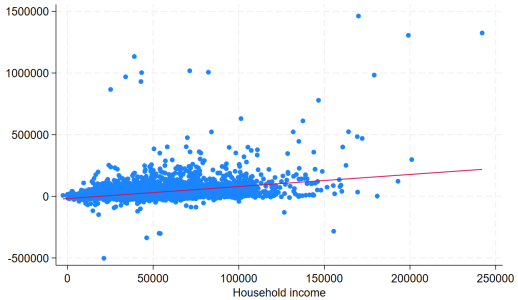
Testing data: $\{y_i^o, x_i^o\}_{i=1}^{m}$, $m + n_T = n$

Estimate of out-of-sample MSE (or R^2) using testing data:

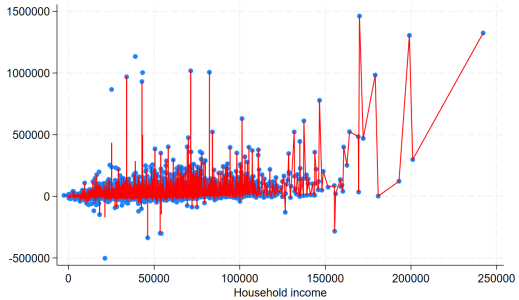
$$\widehat{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i^o - \widehat{f}_T(x_i^o))^2$$

$$\widehat{R^2} = 1 - \frac{\widehat{MSE}}{\frac{1}{m}\sum_{i=1}^{m}(y_i^o)^2}$$

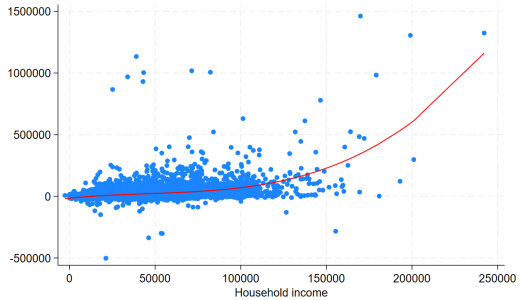
Financial Asset Prediction Revisited



(a) Linear Model Fit



(b) Overfit (Kernel)



(c) Polynomial Fit

Models were fit using $n_T = 7854$ and we kept $m = 2061$ test observations.

In-sample vs out-of-sample fit:

	Linear	Overfit	Polynomial
RMSE (in-sample)	55576	20516	52745
RMSE (out-of-sample)	68706	83167	68959
R2 (in-sample)	0.161	0.886	0.244
R2 (out-of-sample)	0.123	-0.285	0.116

Cross-Validation

Idea of splitting sample and using a training and validation sample good but

1. Depends on choice of training and validation samples
   - Which observations in which sample?
   - How many observations in each?
2. Not using all observations for estimation AND testing

Cross-validation (CV) tries to address these concerns

- Theory for VALIDATION clear and simple
- Theory for cross-validation less transparent

K-Fold CV

1. Divide sample into K approximately equal sized groups
   - n-fold = leave-one-out CV
2. For $k = 1$ to K:
   - Use subsample k as validation sample and use remaining groups as training sample
   - Estimate model using $(Y_{-k}, X_{-k}) \rightarrow \widehat{f}_k(\cdot)$
   - Forecast $\widehat{y}_i = \widehat{f}_k(x_i)$ for all $i \in \mathcal{I}_k$ where $\mathcal{I}_k$ is the set of all indices belonging to group k
3. Estimate generalization error as $\widehat{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$
   - Or use any other prediction criterion as appropriate

Do this for each model/tuning parameter under consideration

Choose the procedure that does best according to forecasting criterion

- Re-estimate selected model using all data
  - theory still being developed: e.g. Chetverikov et al. (2021), Chetverikov and Sørensen (2022)
- Pool predictor $\widehat{f}(X) = \frac{1}{K}\sum_k \widehat{f}_k(X)$
  - general theory providing optimality or near optimality: e.g. Wegkamp (2003), Lecué and Mitchell (2012)

Financial Asset Prediction Revisited

Let's look at 5-Fold CV in our financial asset prediction example:

RMSE by fold:

	Linear	Overfit	Polynomial
Fold 1	63041.474	76649.959	55578.143
Fold 2	44010.977	69666.27	44121.159
Fold 3	41543.506	79414.821	41834.739
Fold 4	60649.763	81475.432	60120.927
Fold 5	64671.224	91462.231	64827.367

Overall CV (R)MSE:

	Linear	Overfit	Polynomial
RMSE	55672.723	80049.04	54042.128

Performance on validation data:

	Linear	KNN	Polynomial
CV Average	0.123	-0.150	0.119
Refit	0.123	-0.285	0.116

Method 1: Penalized Estimation

Introduction

Target: Predict scalar outcome variable Y

- $p \times 1$ vector of regressor variables or features $X = (X_1, ..., X_p)'$
- $p > n$ allowed

Allowing $p > n$ poses obvious problems. Consider linear model and least squares:

Want to find linear prediction rule $x_i' \widehat{\beta}$ as solution to

$$\arg\min_b \sum_i (y_i - x_i' b)^2$$

Usual derivation gives that $\widehat{\beta}$ solves $X'X\widehat{\beta} = X'Y$

- Here X refers to $n \times p$ matrix of observed covariates
- Y refers to $n \times 1$ vector of observed outcomes

High-Dimensional Linear Model: OLS Estimator

Suppose $p > n$ and observed data matrix X is full (row) rank

There are many solutions to $X'X\widehat{\beta} = X'Y$. Specifically, we can take $\widehat{\beta}$ to be any element of

$$\widehat{\beta}^w = (X'X)^- X'Y + [I - (X'X)^- X'X]w$$

for A^- the Moore-Penrose generalized inverse and w an arbitrary conformable vector

No unique coefficient estimator!

High-Dimensional Linear Model: OLS Prediction

Least squares fitted values in HDLM satisfy

$$\begin{aligned}\widehat{Y} &= X\widehat{\beta}^{w} \\ &= X(X'X)^{-}X'Y + [X - X(X'X)^{-}X'X]w \\ &= XX'(XX')^{-1}(XX')^{-1}XX'Y + [X - XX'(XX')^{-1}(XX')^{-1}XX'X]w \\ &= Y\end{aligned}$$

using $(X'X)^{-} = X'(XX')^{-1}(XX')^{-1}X$ (property of Moore-Penrose inverse)

I.e. any solution perfectly fits Y within sample.

Important general takeaways when $p/n \not\approx 0$:

- In-sample measures of fit are inaccurate
- Parameter estimates unstable/arbitrary
- Linearity insufficient structure to allow informative estimation and inference
  - Need further **regularization**/dimension reduction
  - Regularization: intuitively means imposing additional structure to make learning from finite data possible
- carries over to other GLMs/parameterized models

Motivation for High p: Rich Data

Two key sources of high p:

1. Rich data
2. Flexible estimation

1. Rich data:

- Data sets increasingly contain many features. E.g.
  - aggregate characteristics (e.g. country, state, city, ...) for macro questions, state level policy analysis, ...
  - price and product characteristics for demand analysis
  - individual health history for health economics/policy analysis
  - individual tax records for wage/employment/labor econ questions
  - anything involving text or image data

Motivation for High p: Flexible Estimation

2. Flexible Estimation:

Recall the solution to best prediction problem

$$g(W) = \arg\min_{m(W)} \mathrm{E}[(Y - m(W))^2]$$

is the conditional expectation function $g(W) = \mathrm{E}[Y|W]$

- need not be linear
- will be better (at least in population) than linear prediction rule unless $g(W) = \beta' W$
- can be approximated by a basis expansion

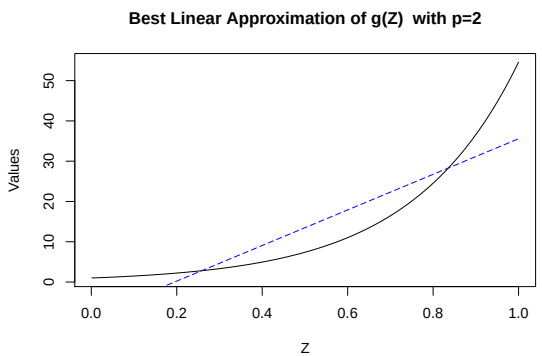
$$g(W) = \sum_{j=1}^{p} \beta_j \varphi_j(W) + r_p(W)$$

  - have $X = (\varphi_1(W), ..., \varphi_p(W))'$
  - $r_p(W)$ represents approximation error which generally $\searrow 0$ as $p \to \infty$
  - i.e. using approximating functions allows linear prediction to approximate best predictor $\mathrm{E}[Y|W]$

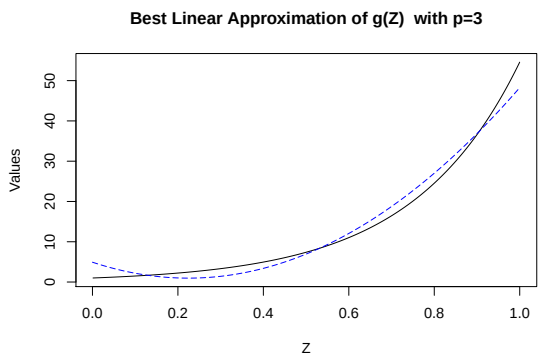
Flexible Estimation: Illustration

Let's look at approximating $Y = \exp\{W\}$ for $W \in [0, 1]$ in the noiseless case

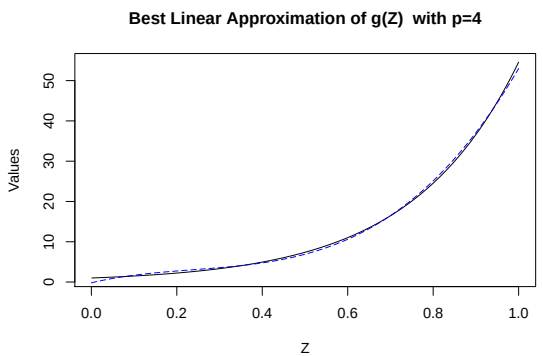
- using $X = (1, W, W^2, ..., W^{p-1})$



(a) Linear Approximation



(b) Quadratic Approximation



(c) Cubic Approximation

When raw input W is moderate dimensional, p increases very quickly as flexibility is added.

Penalized Estimation

Goal: Learn a prediction rule for Y given X that generalizes out-of-sample in a setting allowing $p \gg n$:

Penalized Estimation

Define penalized estimator

$$\widehat{\beta} = \arg\min_{b \in \mathbb{R}^p} \sum_i \ell(y_i, x_i; b) + \lambda p(b)$$

where

- $\ell(y_i, x_i; b)$ is a measure of forecast loss
- $\lambda p(b)$ is a penalty function that increases in "model complexity"
  - complexity defined in terms coefficients
  - $\lambda p(b)$ key input - form of $p(b)$ determines structure of $\widehat{\beta}$

- Penalized estimator trades off in-sample fit with complexity of the prediction rule
- Penalized GLM's readily available with appropriate choice of loss
  - Can understand estimation results within familiar context of GLM
  - Will only consider $\ell(y_i, x_i; b) = (y_i - x_i' b)^2$ (penalized linear regression)

Canonical Examples

MANY penalized estimators in the literature

- consider two fundamental examples:
  - **Ridge**: $\widehat{\beta} = \arg\min_b \sum_i (y_i - x_i' b)^2 + \lambda \sum_j b_j^2$
  - **Lasso**: $\widehat{\beta} = \arg\min_b \sum_i (y_i - x_i' b)^2 + \lambda \sum_j |b_j|$

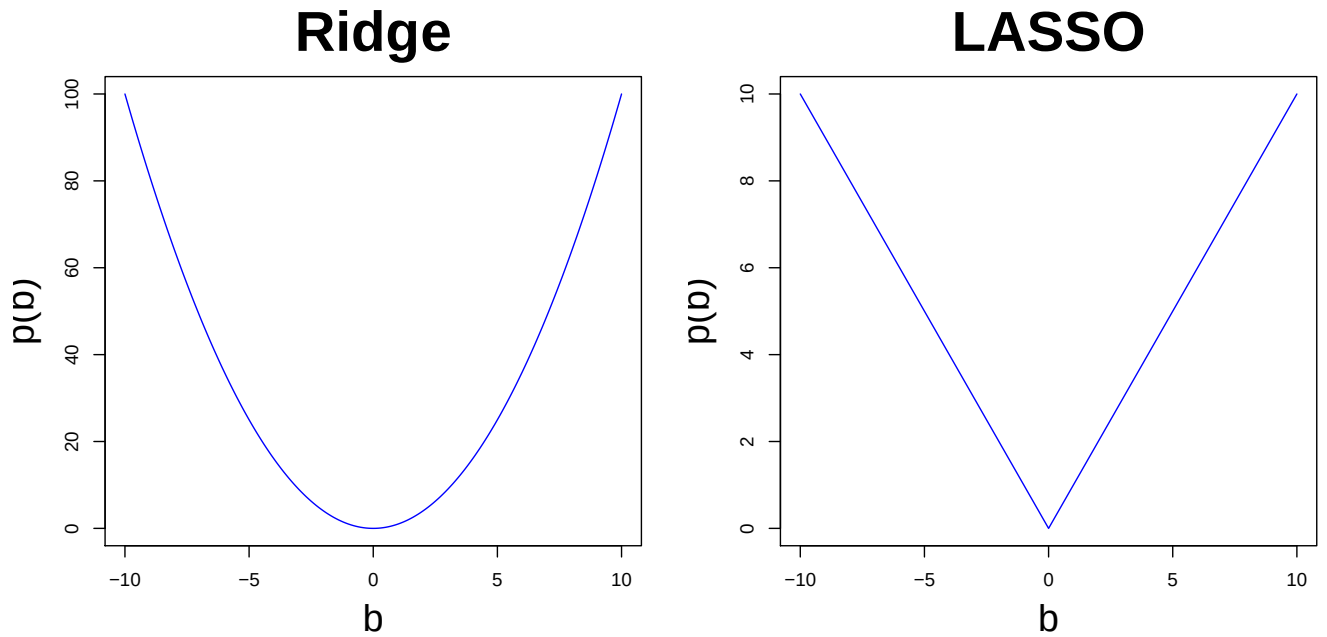


Figure 10: Lasso and Ridge penalty functions in one variable

Ridge

Ridge:

$$p(b) = \sum_{j=1}^{p} b_j^2$$

- Closed form solution.
- Estimated model will have all coefficients non-zero and mostly small - a dense model
  - Penalty essentially 0 for small b_j
  - Penalty increases quickly as b_j increases and is very aggressive for large b_j
  - **Does not do variable selection!**
- See Hsu et al. (2012) for theoretical properties but intuitively dense models are very hard to learn
  - need lots of observations per parameter to learn lots of small parameters

Lasso

A popular penalized estimator that produces a sparse estimate is Lasso.

Lasso: $p(b) = \sum_{j=1}^{p} |b_j|$

- Estimated model will have some coefficients set exactly to 0 and favor small coefficients - a sparse model
  - Cost to increasing b_j constant
  - Kink at 0 means benefit to fit must be sufficiently large to be worth having $b_j \neq 0$
  - **Does variable/model selection!**
- Generalizes "usual" practice which relies on a pre-specified (small) set of variables by trying to learn identities of relevant predictors
- Can have good theoretical properties even when $p \gg n$
  - E.g., Bickel et al. (2009); Belloni and Chernozhukov (2013); Belloni et al. (2012); Farrell (2015); Belloni et al. (2016); Chernozhukov et al. (2021); Chetverikov et al. (2021); Chetverikov and Sørensen (2022)
  - "Bet on sparsity" - Friedman et al. (2001)

Choice of Penalty Parameter

Parameter λ important for actual performance of estimator

- λ is tuning parameter that controls "shrinkage"
- referred to as penalty parameter or penalty level

Choosing λ in practice:

- CV is probably most common method
  - See Chetverikov et al. (2021), Chetverikov and Sørensen (2022) for some theory for Lasso
- Can provide theoretically valid plug-in choices for Lasso in many settings
  - heteroscedasticity – Belloni et al. (2012); clustering – Belloni et al. (2016); time series dependence – Chernozhukov et al. (2021)
  - Probably most useful in settings with dependence where CV may not be obvious

Generic Comments

Penalty defined in terms of coefficients means scale of variables very important:

- Usual practice is to define penalty in terms of coefficients on standardized variables
- Standardization done by default in canned software
  - Results returned on original scale

Intercept:

- Generally don't want to penalize intercept
- Intercept unpenalized by default in canned software

Loadings:

- Easy to generalize to allow different penalization of different coefficients
  - E.g. $p(b) = \sum_j \psi_j |b_j|$
- Useful when there are variables you wish to treat differently (e.g. make sure they're in the model)
- Potentially important in settings with non-iid data (references above)

401(k) Example: Data

Pick up with the financial asset prediction introduced earlier.

Data:

- 9915 observations from the 1991 SIPP (divided with 7854 training, 2061 validation)
- $y_i =$ net financial assets
- $x_i =$ individual characteristics
  - income, age, family size, education (high school, some college, college), married, two-earner, defined benefit, ira, home-owner, eligibility for a 401(k)
- same data as in Poterba et al. (1995)

401(k) Example: HDLM Predictive Models

We'll consider three different variable structures (number of regressors is number of terms not dropped by `Stata: regress`)

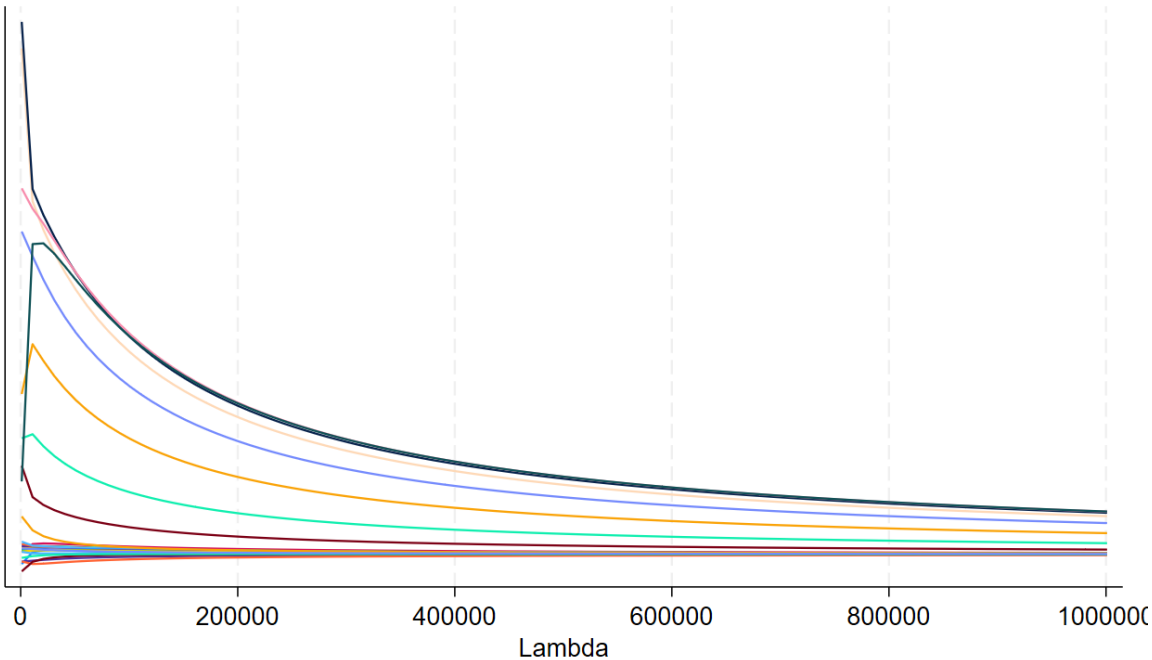
- Basic Model: Raw regressors only
  - p = 10
- Polynomial Model: 6th order polynomial in age, 8th order polynomial in income, fourth order polynomial in education, 2nd order polynomial in family size, + all other variables
  - p = 25
- Interactive Model: Polynomial model + all non-redundant interactions
  - p = 257

We'll estimate each model using OLS, Lasso, Ridge

- We'll use cross-validation to choose tuning parameters for the penalized estimators
- We'll evaluate performance on a validation sample

401(k) Example: Ridge Shrinkage

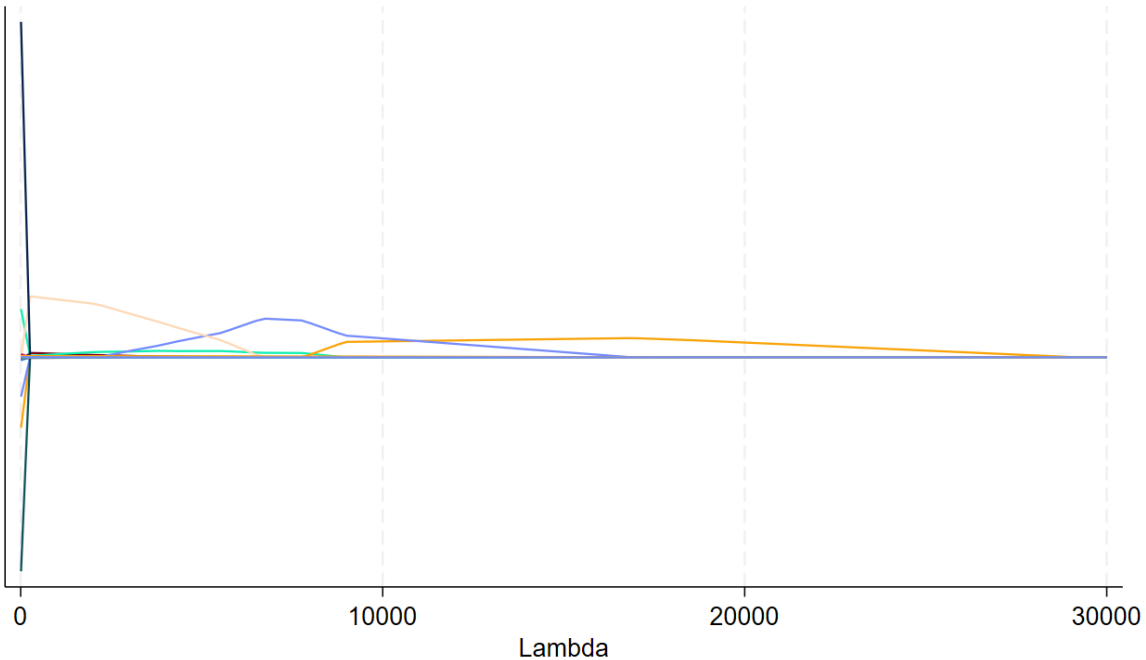
Before turning to actual prediction, let's look at impact of penalty parameter on estimated coefficients in polynomial model



- Ridge is very smooth in λ
- No zeroes
- Shrinkage does not mean monotonic decrease to 0 as $\lambda \uparrow \infty$

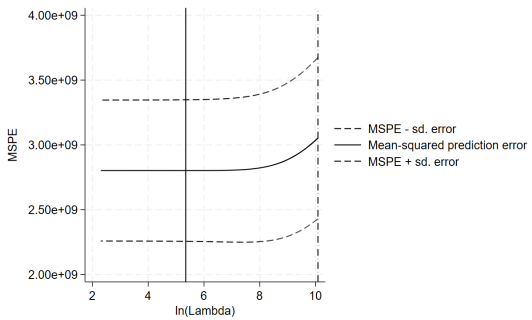
401(k) Example: Lasso Shrinkage

Before turning to actual prediction, let's look at impact of penalty parameter on estimated coefficients in polynomial model

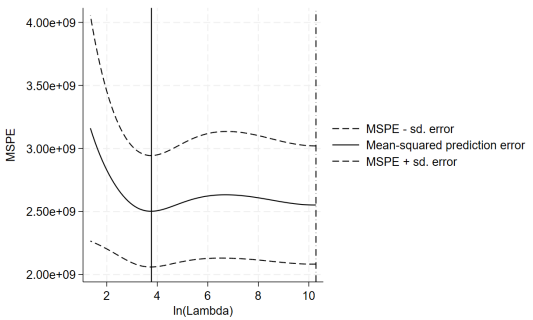


- Lasso is continuous, not smooth in λ
- Zeroes variables out for large enough λ
- Shrinkage does not mean monotonic decrease to 0 as $\lambda \uparrow \infty$

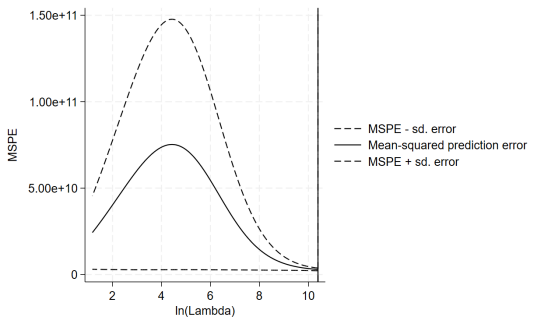
401(k) Example: Cross Validation - Ridge



(a) Ridge CV function for basic design



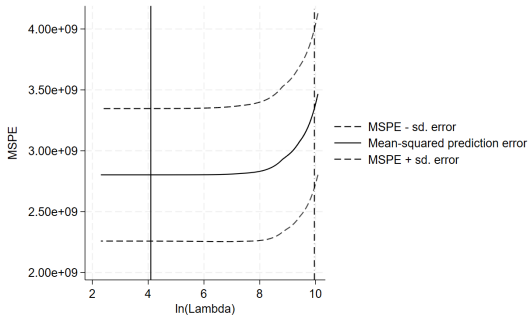
(b) Ridge CV function for polynomial design



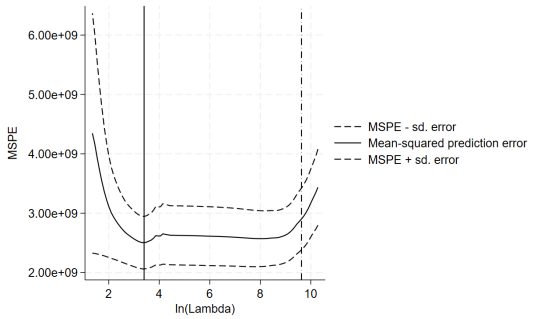
(c) Ridge CV function for interaction design

For comparison, blue line is CV SSE for OLS with basic model.

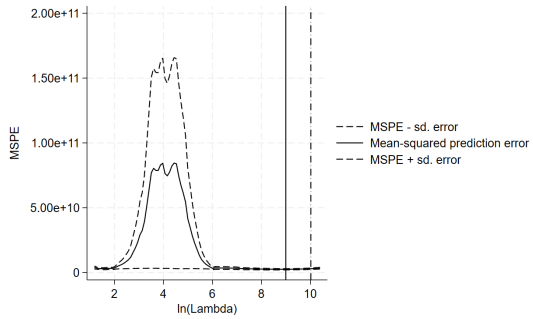
401(k) Example: Cross Validation - Lasso



(a) Lasso CV function for basic design



(b) Lasso CV function for polynomial design



(c) Lasso CV function for interaction design

For comparison, blue line is CV SSE for OLS with basic model.

401(k) Example: Cross Validated Prediction Errors

Minimum cross validated RMSE for each procedure:

- I.e. CV RMSE using tuning parameter values that minimize CV SSE

	Base	Polynomial	Interactions
OLS	52937	56429	68037
Lasso	52937	50026	50277
Ridge	52935	50018	54382

Outside of OLS with flexible controls, they're all relatively close.

401(k) Example: Validation R^2

Here we provide R^2 from the set aside validation data set

- Model refit using all training data at CV-min tuning parameter value

	Base	Polynomial	Interactions
OLS	0.195	0.181	0.013
Lasso	0.195	0.192	0.174
Ridge	0.195	0.192	0.191

Method 2: Trees

Modern Nonlinear Regression

- Target: Predict scalar outcome variable Y
- $p \times 1$ vector of regressor variables or features $X = (X_1, ..., X_p)'$
  - High Dimensional: $p \gg n$ is allowed
- Best (MSE) predictor: $\mathrm{E}[Y|X] := g(X)$
  - In HDLM's, approximate $g(X) \approx \sum_j \beta_j p_j(X)$ for a set of *pre-specified* functions $\{p_j(\cdot)\}$
  - Specifying $\{p_j(\cdot)\}$ is challenging in even moderate dimensional settings.

Modern nonlinear regression: Aims to obtain good prediction rule $\widehat{g}(X)$ without pre-specifying an approximation for $g(X)$

- Tries to learn how to represent $g(X)$ from the data
- Useful and interesting even when X is low-dimensional
- Examples: trees, deep neural nets

Tree-Based Methods

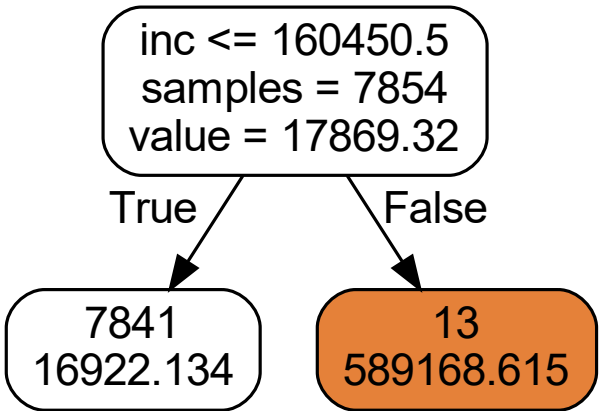
Basic idea of tree-based methods:

- Adaptively partition the regressor space into regions
  - Usually rectangles
- Fit a (simple) prediction rule within each region
  - Usually a constant prediction rule
- Partition and prediction rule jointly chosen to get good in-sample fit
- With rectangles and a constant rule within rectangles - equivalent to approximating $g(X)$ with a step function where steps are not pre-specified

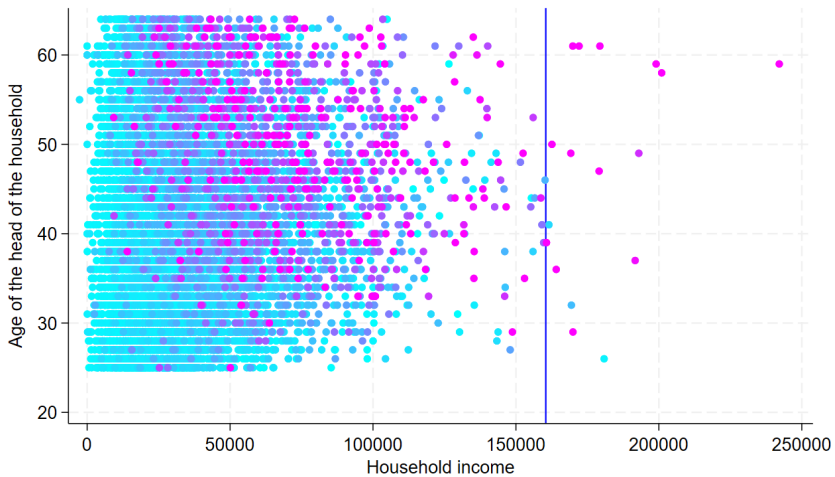
Heuristic Regression Tree Algorithm

- Basic idea: Use the data to form a decision tree using recursive binary partitions
- Step 1:
  - Let $R_1(j,s) = \{X|X_j < s\}$ and $R_2(j,s) = \{X|X_j \geq s\}$
  - Choose j and s as
    $\arg\min \sum_{i:x_i \in R_1(j,s)} (y_i - \widehat{y}_{i,R_1})^2 + \sum_{i:x_i \in R_{i,2}(j,s)} (y_i - \widehat{y}_{i,R_2})^2$
    - $\widehat{y}_{i,R_k} = \bar{y}_{i,R_k}$ is the forecast of y_i using data in region R_k - just the sample mean of Y in that region
- Step 2:
  - Split the data into the two subregions from Step 1.
  - Repeat the splitting process on each subregion, taking the split that minimizes loss
- etc...
- Stop splitting when some stopping criterion reached (often a minimum node size - e.g. no less than 5 observations - or a maximum number of splits - e.g. no more than 4 terminal nodes)

401(k) Example: Trees



(a) Tree

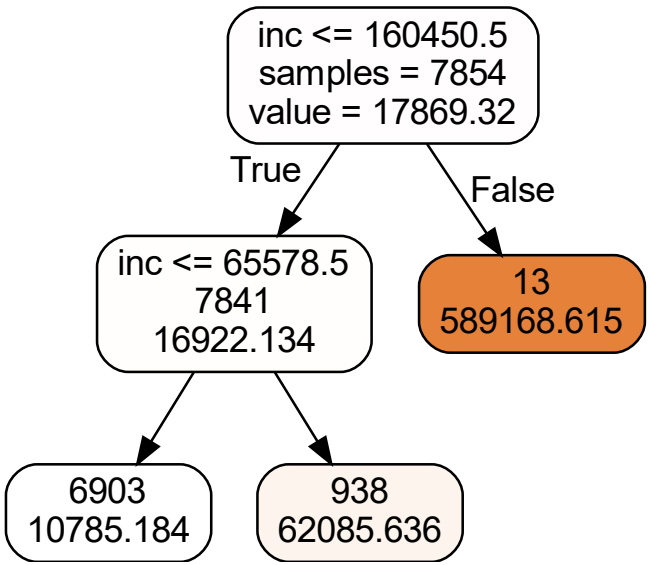


(b) Tree Partition

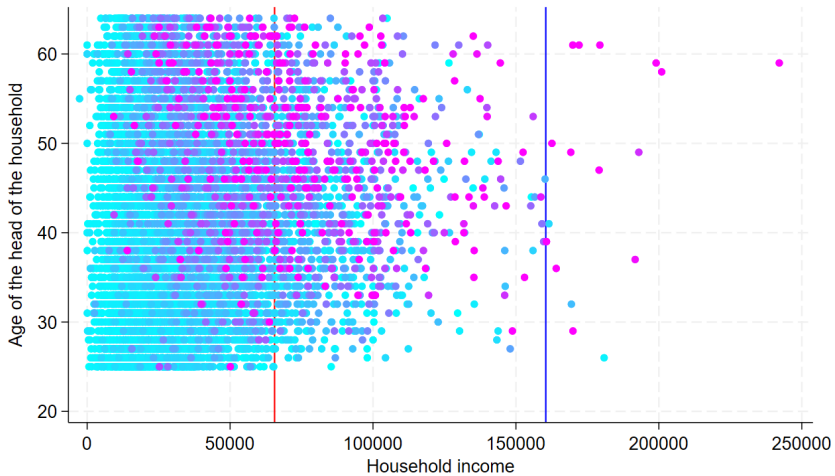
Trees fit in financial asset data from SIPP

- First split (using only age and income)

401(k) Example: Trees



(a) Tree

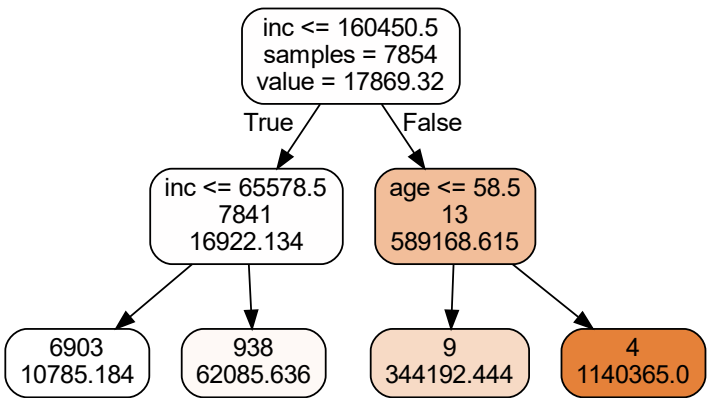


(b) Tree Partition

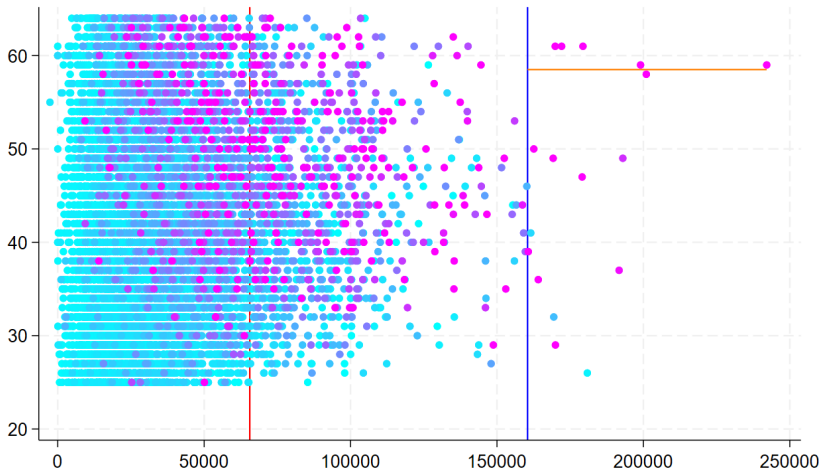
Trees fit in financial asset data from SIPP

- Second split (using only age and income)
- Start to see nonlinearity

401(k) Example: Trees



(a) Tree



(b) Tree Partition

Trees fit in financial asset data from SIPP

- Third split (using only age and income)
- Start to see interaction

401(k) Example: Prediction Performance

In-sample performance increases as tree becomes more complex.

Agreement between in sample and out of sample performance tends to decrease as tree becomes more complex.

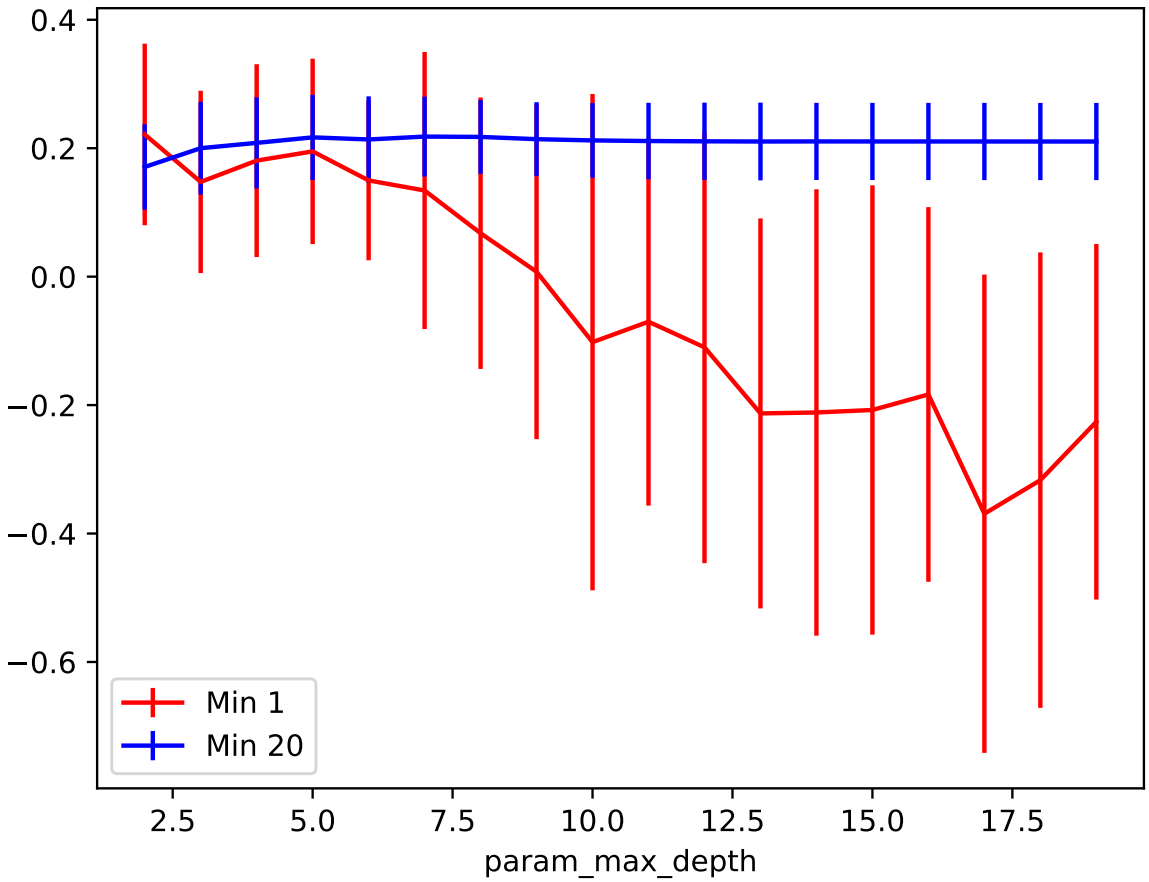
- typically choose tree via cross-validation
- could also look at "cost-complexity pruning" - not today

R^2 in-sample and in validation sample:

	No Min		Min = 20	
	in-sample	Validation	in-sample	Validation
Depth 1	0.147	-0.032	0.122	0.113
Depth 2	0.315	0.061	0.215	0.158
Depth 3	0.374	0.106	0.262	0.199
Depth 12	0.880	0.048	0.350	0.210

Default has no restriction on minimum observations per leaf – also consider setting minimum to 20

401(k) Example: Tree CV Function



- cross-validating over depth (could try leaves, min leaf size, ...)
- y-axis is cross-validation R^2

401(k) Example: CV min and Validation R^2

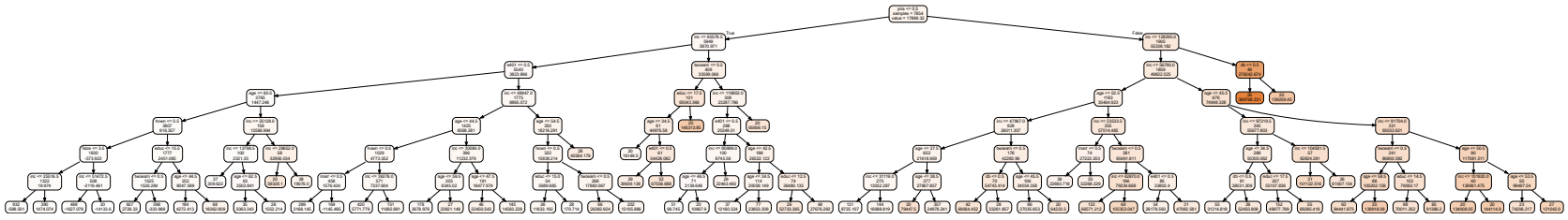
R^2 from CV and in validation sample:

	CV	Validation
Min 1	0.221	0.061
Min 20	0.218	0.223

401(k) Example: CV Tree

Depth 7 tree is pretty hard to read

- we could blow it up ...



A Quick Summary

Good:

- Don't have to think about the scale of x's
- Computationally fast ("scales").
- Small trees are interpretable.
- Does automatic interaction detection
- Does variable selection.

Bad:

- Step function is crude, may not give the best predictive performance.
  - E.g. Doesn't do well with linearity
- Big trees are not interpretable.

Method 3: Random Forests

Bagging

Bootstrap aggregation (bagging) generic idea in ML (Breiman (1996)):

- Idea:
  - fit lots of noisy, low bias models
  - average them together to reduce variability
- Also "subagging" (subsample bootstrap aggregation)
  - computationally convenient and "clean"

Basic algorithm:

- For $b = 1$ to B
  - Draw a bootstrap sample $\{y_i^b, x_i^b\}_{i=1}^n$
  - Estimate model using bootstrap sample $\rightarrow \widehat{f}_b(\cdot)$
- Bagging estimator: $\widehat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_b(x)$
  - Intuition: Nice convergence if $\widehat{f}_b(\cdot)$ were independent across b and unbiased
- B is an additional tuning parameter
  - results seem relatively insensitive as long as B is "big"

Basic procedure can be applied on top of any learner, but only interesting if learner adapts to data at hand.

- E.g. bagging applied to OLS with a fixed set of covariates is a complex way to calculate full sample OLS

Out-of-Bag Error Estimation

Useful byproduct of bagging:

With careful accounting, get an estimate of out-of-sample performance for free!

Out-of-Bag (OOB) Error Estimation:

- Sampling with replacement $\Rightarrow$ Not all observations used at each replication b
- Can form an out-of-sample prediction for each observation not used at replication b (called "out-of-bag" observations) along with estimating $\widehat{f}_b$
- Average predicted OOB responses to get out-of-sample prediction for i^{th} observation
- Use these predictions to form OOB loss (e.g. OOB MSE)
- OOB MSE $\rightarrow$ leave-one-out CV as $B \rightarrow \infty$

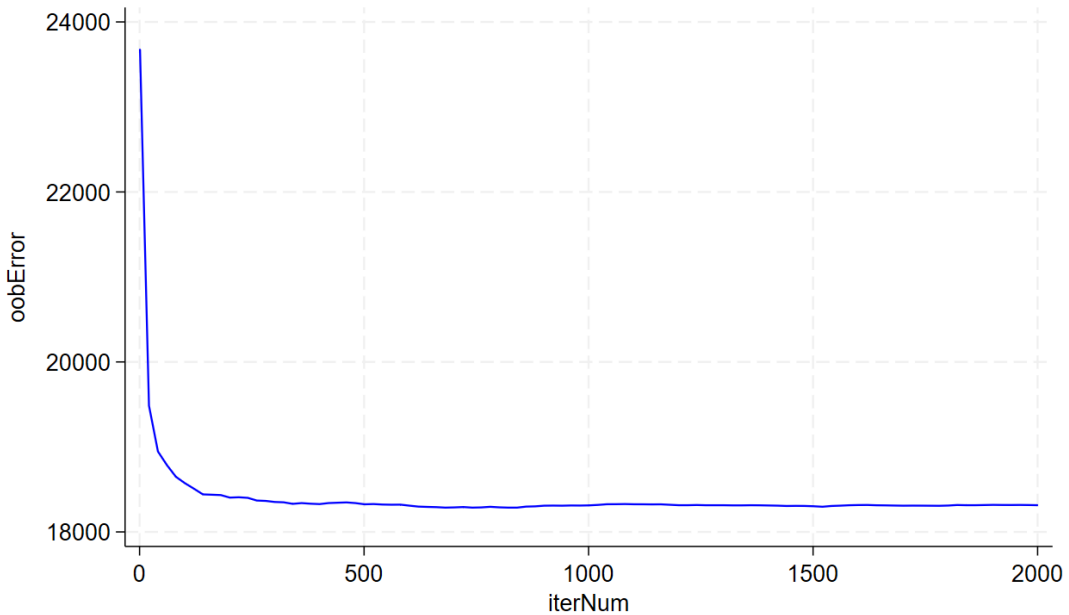
Random Forests

Applying bagging on top of a tree produces a "random forest" (Breiman (2001)):

- For $b = 1$ to B
  - Draw a bootstrap sample $\{y_i^b, x_i^b\}_{i=1}^n$
    - Typical random forest implementation also randomly samples variables. I.e. each x_i^b consists of a (potentially different) random subset of the columns of X
    - Idea is that this "decorrelates" fits across bootstrap samples
  - Estimate model using bootstrap sample $\rightarrow \widehat{f}_b(\cdot)$
- Bagging estimator: $\widehat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_b(x)$
  - Averaging across step functions with steps chosen at different points
  - Will produce a smooth fit (with continuous X and sufficiently large B)
- Sales pitch is much less sensitive to tuning choices as long as "low bias" in learning $\widehat{f}_b(\cdot)$
  - use trees with lots of leaves (fast to learn and low bias)
  - tuning still seems to help though (in Hansen's experience)

401(k) Example: Random Forests

Let's look at out-of-bag error in a simple example



- Figure gives OOB RMSE estimate against B
- Performance seems "stable" for $B > 750$
- Validation sample $R^2 \approx .218$. (Note this can change across reruns - it is a random forest)
- Comparable to best (HD)LM's with no prespecification

401(k) Example: Other Tuning Choices

	OOB RMSE	Validation R^2
Default	18314	0.218
No X Randomization	19226	0.159
Min Size(20)	17902	0.214
Min Size(40)	17903	0.211
Min Size(80)	18056	0.206
Min Size(160)	18392	0.195

- Performance relatively stable across tuning choices (except not randomizing over X)
- OOB error estimates not comparable to CV error estimates from other procedures
    - Make sure to use comparable metrics if comparing across methods

Method 4: Boosted Trees

Boosting

Boosting (e.g. Schapire (1990), Friedman et al. (2000)) another generic ML idea

- Basic idea is to improve model fit in small increments via recursive fitting
  - Fit a simple model to outcome and construct residuals
  - Fit a new simple model to residuals and construct new residuals
  - Continue iterating
- Each step improves fit relative to previous step
- The sum of all the prediction rules is the prediction rule for the outcome
- Strong potential to overfit if enough steps are taken
  - Choose number of steps on basis of (cross-)validation

Boosted Trees

Basic boosting algorithm for trees with data $\{y_i, x_i\}_{i=1}^n$ proceeds as follows:

Step 1. Initialize residuals $r_i = y_i$ for $i = 1, ..., n$

Step 2. For b = 1,...B

1. Fit a tree-based prediction rule $\widehat{g}_b(X)$ using data $\{r_i, x_i\}_{i=1}^n$
   - Should be a simple tree - e.g. a tree of depth d for d very small
   - Nice interpretation: $d = 1$ additive nonparametric model, $d = 2$ nonparametric model with maximum interaction depth of 1, ...
2. Update the residuals $r_i \leftarrow r_i - \lambda \widehat{g}_b(x_i)$
   - λ is the "learning rate" and plays an important role. Choose $\lambda \ll 1$

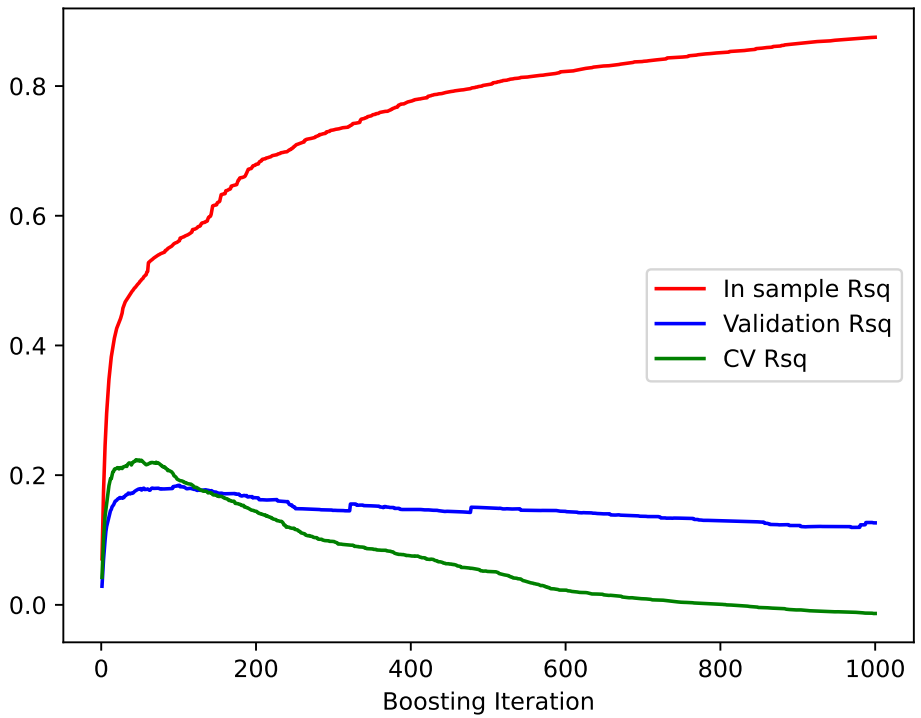
Step 3. Construct the final prediction rule for Y:

$$\widehat{g}(X) = \sum_{b=1}^{B} \lambda \widehat{g}_b(X)$$

Tuning parameters, (B, λ, d) can be chosen by cross-validation

401(k) Example: Boosted Trees

Let's look at how in-sample and validation sample R^2 change as we increase boosting iterations (with default tuning choices)

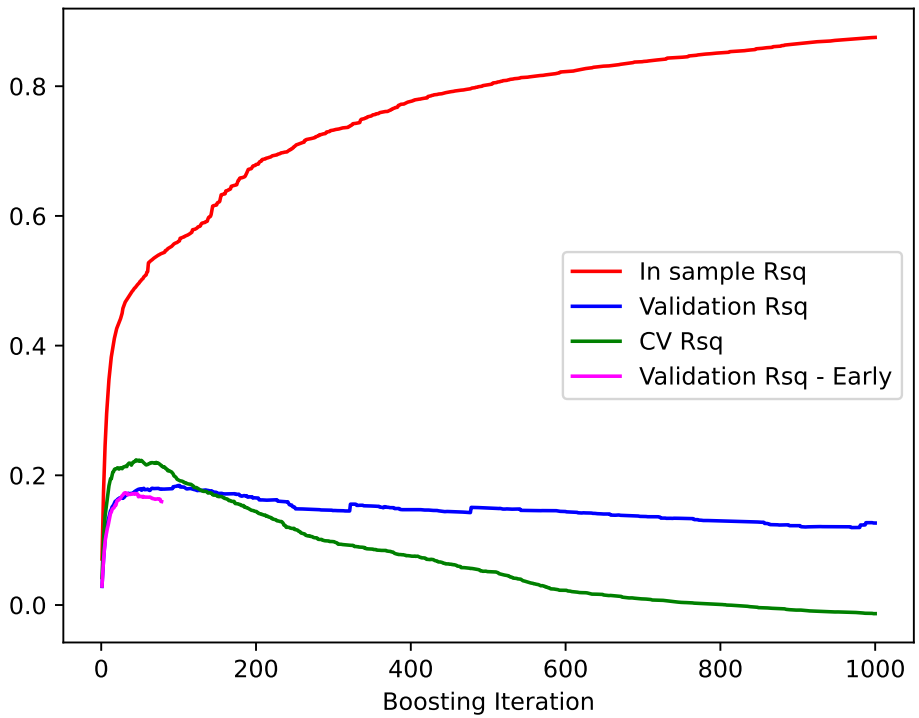


- Figure gives Validation and Training sample R^2 against B
- In sample fit keeps increasing as $B \nearrow$ while validation starts to decrease
- Validation sample $R^2 \approx .187$ at best B (in validation sample).
- Validation sample $R^2 \approx .175$ at CV B

401(k) Example: Early Stopping

Another way to choose B is via *early stopping*

- Monitor fit on a validation sample during training and stop once fit in validation data stops improving
- Choice of how far to go after fit stops improving
- Choice of tolerance for judging improvement



- Early stopping (with 10 steps of no improvement and a tolerance of .0001) stops at 78 iterations and gives validation sample $R^2 \approx .159$

401(k) Example: Other Tuning Choices

All impose minimum leaf size of 20

Depth	Rate	Iter	CV RMSE	in-sample R^2	Validation R^2
6	0.3	500	59020.036	0.823	0.058
6	0.3	e.s.	52541.337	0.405	0.205
6	0.1	500	54218.371	0.638	0.125
6	0.1	e.s.	52552.372	0.397	0.205
5	0.1	e.s.	52344.737	0.374	0.205
4	0.1	e.s.	52369.538	0.365	0.202
3	0.1	e.s.	52163.745	0.322	0.187
2	0.1	e.s.	52708.711	0.331	0.179
6	0.01	e.s.	53467.074	0.276	0.173
5	0.01	e.s.	53631.916	0.263	0.167
4	0.01	e.s.	53819.876	0.246	0.159
3	0.01	e.s.	54197.389	0.228	0.148
2	0.01	e.s.	55020.508	0.199	0.132

Method 5: Deep Neural Networks

Single Layer Neural Networks

A single layer neural network (NN) produces a prediction rule of the form:

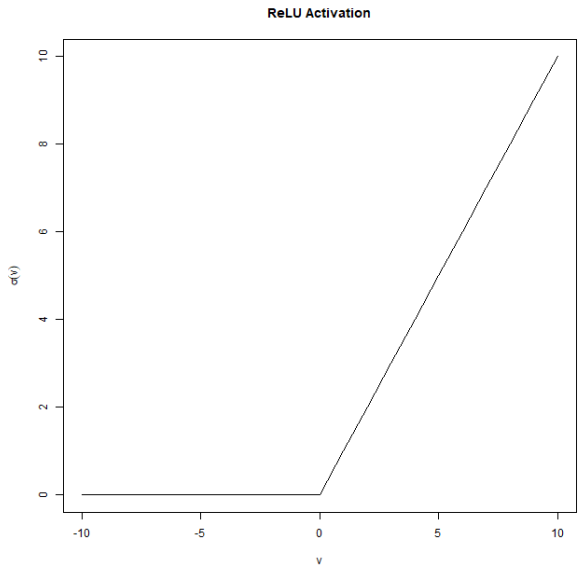
$$\widehat{g}(x) = \sum_{m=1}^{M} \widehat{\beta}_m Z_m(\widehat{\alpha}_m)$$

- $Z_m(\widehat{\alpha}_m)$ are constructed regressors called **neurons**
  - Always set $Z_1(\alpha_1) = 1$
  - For $m = 2, ..., M$, $Z_m(\alpha_m) = \sigma(\alpha_m' X)$ for **activation function** $\sigma(\cdot)$
- a NN is a (generalized) linear combination of "learned" predictor variables $\{Z_m(\widehat{\alpha}_m)\}_{m=1}^{M}$
  - to be interesting, $\sigma(\cdot)$ should be nonlinear
  - **Feature Engineering**: rather than specifying a representation, we are trying to learn **features** or "factors" that combine the raw inputs to provide the best prediction of Y
- estimated by empirical loss minimization with a variety of regularization schemes (more on this later)

Activation functions

Most popular activation function (currently) is the **rectified linear unit** function (ReLU):

$$\sigma(v) = \max(0, v)$$



Other common activation functions:

- smoothed rectified linear unit function (SReLU): $\sigma(v) = \log(1 + \exp(v))$
- sigmoid: $\sigma(v) = \frac{1}{1+\exp(-v)}$

Deep Neural Networks

Deep Neural Networks (DNN; aka Deep Nets, Deep Learning) adds more **layers** to the NN structure

- The DNN structure is repeated composition of nonlinear mappings

$$X \overset{f_1}{\longmapsto} H^{(1)} \overset{f_2}{\longmapsto} \quad ... \overset{f_m}{\longmapsto} H^{(m)} \overset{f_{m+1}}{\longmapsto} Z$$

- **neurons**: $H^{(\ell)} = \{H_k^{(\ell)}\}_{k=1}^{K_\ell}$
- Mapping from one layer to next:

$$f_\ell : v \longmapsto \{H_k^{(\ell)}(v)\}_{k=1}^{K_\ell} := (1, \{\sigma_{k,\ell}(v'\alpha_{k,\ell})\}_{k=2}^{K_\ell})$$

- **Input Layer**: X (raw predictor variables)
- **Hidden Layers**: Layers between input layer and output
- Final prediction model uses Z as inputs (e.g. $\widehat{g}(X) = \beta' Z$)
- Let η collect all the $\alpha_{k,\ell}$'s and parameters of the final prediction model

DNN Problem for MSE

"Train" DNN by attempting to solve

$$\min_{\eta} \sum_{i} (y_i - \widehat{g}(X;\eta))^2 + \text{penalty}(\eta, \lambda) \tag{8.1}$$

- Could use other loss
- penalty(η, λ) is a penalty function (e.g. Ridge style, Lasso style) with penalty parameter(s) λ
- Generally non-convex and highly over-parameterized
- Lots of tuning parameters
  - Depth of network
  - How many neurons per layer
  - Connection structure between neurons (won't talk about this in this class)
  - Regularization?
    - Explicit penalization and penalty parameters? Dropout? Early stopping? Optimization parameters (e.g. learning rate, starting values)?
  - No great theoretical guidance - *ad hoc* choice and (cross-)validation

playground.tensorflow.org

Before turning to some details, which play with some simple NN's in a nice visual structure available at playground.tensorflow.org

Ingredients to DNN Training

Explosion of DNNs driven by advances in computation, optimization, and structure

- structures that allow efficient computation of gradients
- structures that allow efficient parallelization
- stochastic gradient descent (SGD)
- implicit regularization
- details well outside scope of these lectures (and can fill their own courses); see, e.g. Goodfellow et al. (2016) (textbook)

Here, we'll briefly talk about

- SGD
- Dropout
- Early stopping
- Network width and depth

Stochastic Gradient Descent (SGD)

Basic gradient descent:

- Inputs: Initial condition - η_0, learning rate - ν, gradient $\nabla f(\eta)$
- Iterate until some convergence criterion is met updating estimated parameter at t^{th} iteration as

$$\eta_t = \eta_{t-1} - \nu \nabla f(\eta_{t-1})$$

- Lots of variants out there

SGD is just gradient descent where the gradient is computed using subsets of observations

- **batch**: subsample of data (often single observations)
- **epoch**: a single cycle through all observations
- Lots of variants out there

Advantages of SGD:

- fast and scalable (only need little bits of data at a time)
- noise in gradient computation seems to have some advantages - e.g. help avoid saddle points
- SGD tuning parameters (e.g. epochs, learning rate, batch size) offer another lever for regularization

Dropout

Dropout regularization:

- randomly drop neurons during parameter update sets of the training process
- tuning parameter p - probability of dropping a neuron
  - can use different probabilities for different layers
  - common recommendation is $p = .5$
- akin to randomly dropping variables in random forests
- regularizes by reducing "co-adaptation" of neurons
  - E.g. pretend one neuron captured the generalizable prediction pattern. Given this other neurons can specialize (co-adapt) to fit idiosyncracies in the training data.
  - Dropout encourages more robust (i.e. regularized) networks by not "allowing" this type of overspecialization

Early Stopping

Early Stopping:

- optimization stopped before minimum is achieved
- measure of out-of-sample prediction accuracy is monitored along with value of objective function during parameter update steps
    - optimization stops when out-of-sample prediction accuracy levels off or begins to degrade
- updating parameters to achieve in-sample accuracy but stopping based on out-of-sample accuracy guards against overfitting
- can be used with other methods

Network Depth and Width

Good approximation of a target function via a DNN can be achieved with sufficient depth/width (not surprising)

- e.g. Yarotsky (2017); Schmidt-Hieber (2020); Farrell et al. (2021); Kidger and Lyons (2020)

- Depth requires more sequential computations - less easy to parallelize
- Lu et al. (2017) suggest depth may be more important than width for ReLU networks
- Mhaskar et al. (2017): compositional functions approximated by shallow or deep networks, but deep networks require (exponentially) fewer parameters

401(k) Example: Neural Nets

Let's revisit 401(k) data and look at how neural nets do under some arbitrary choices

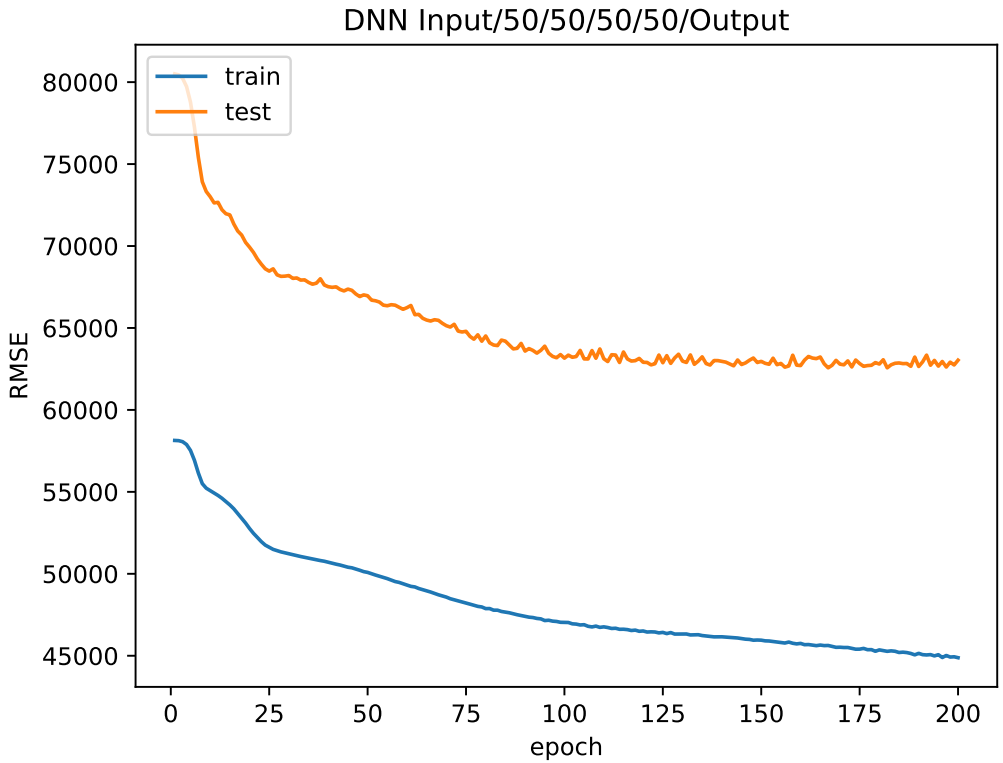
Basic structure

- Input layer $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ output layer
- Layers fully connected
- Look at some different regularization choices

401(k) Example: DNN with No Explicit Regularization

Model A:

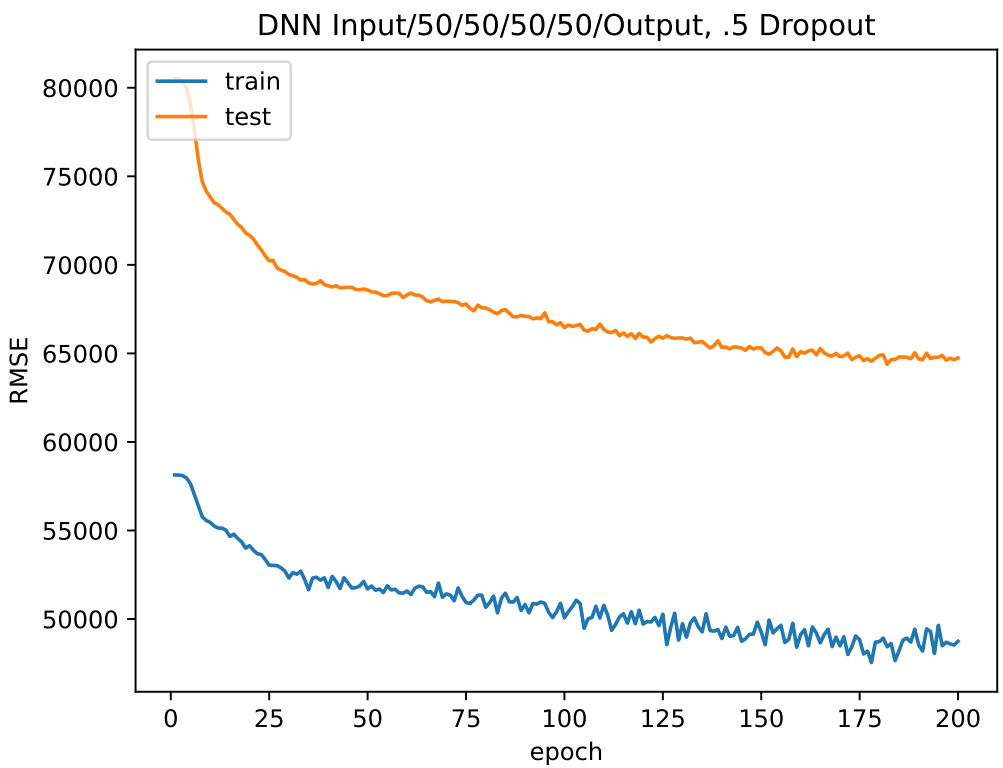
- Input layer $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ output layer
  - trained for 200 epochs, batch size of 200
  - trained w/ 20% hold-out sample
- R^2 on validation data: 0.220



401(k) Example: DNN with Dropout

Model B:

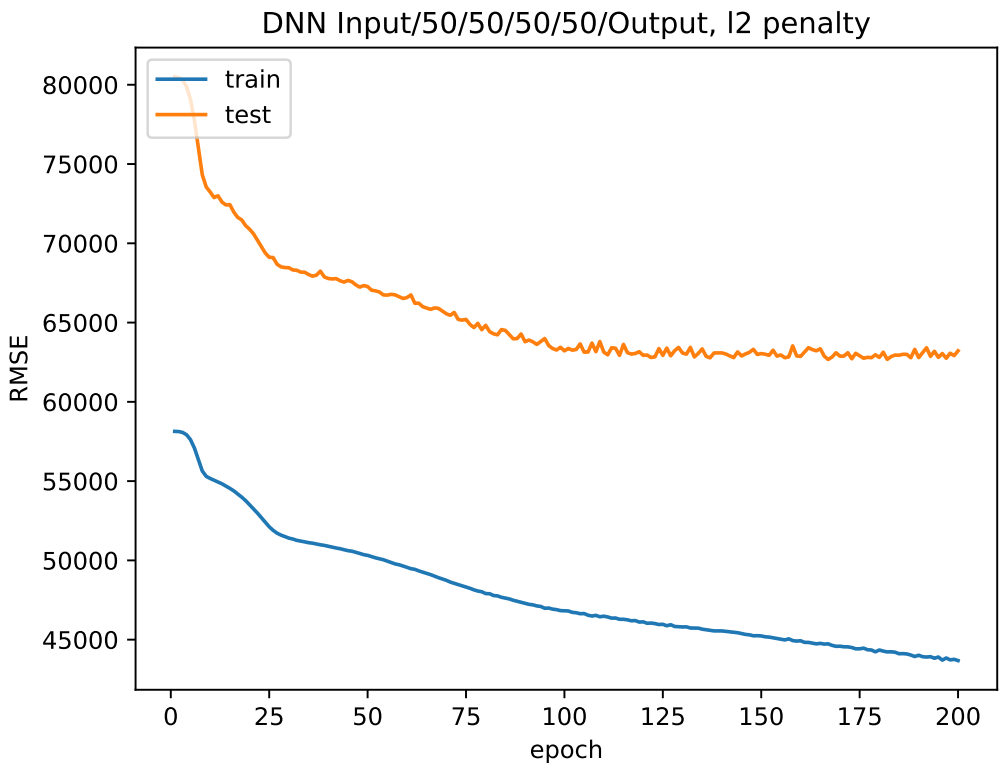
- Input layer $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ output layer
  - .5 dropout on hidden layers
  - trained for 200 epochs, batch size of 200
  - trained w/ 20% hold-out sample
- R^2 on test data: 0.212



401(k) Example: Early Stopping and ℓ_1 Penalty

Model C:

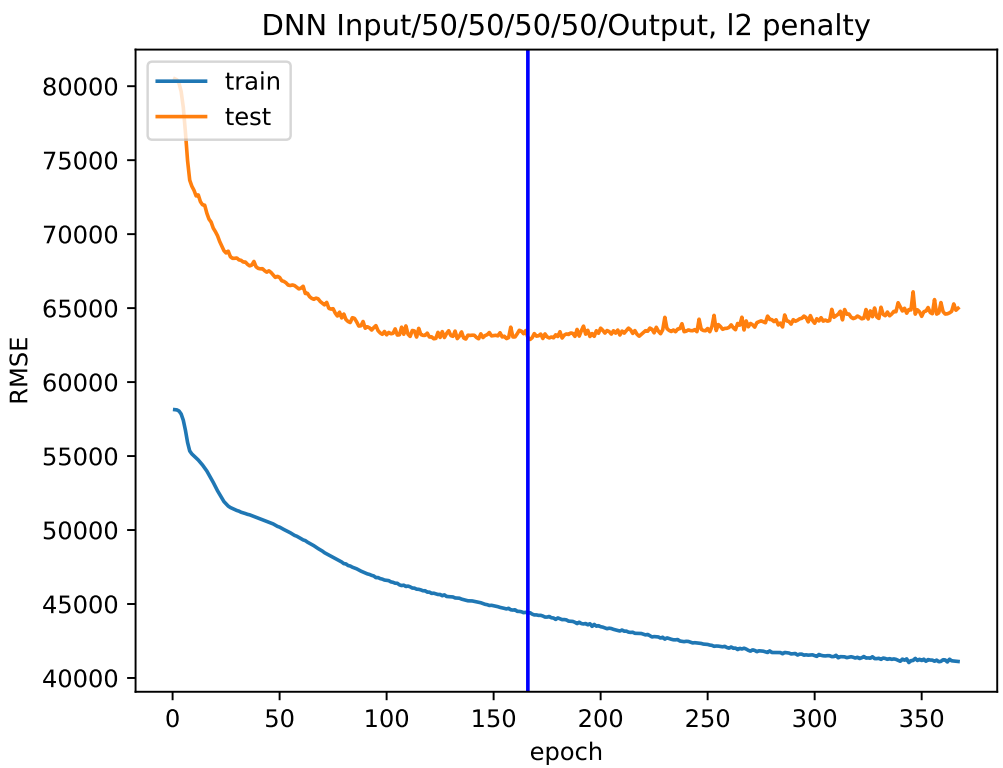
- Input layer $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ output layer
  - ℓ_2 penalization on all layers (penalty parameter pulled out of a hat)
  - trained for 200 epochs, batch size of 200
  - trained w/ 20% hold-out sample
- R^2 on test data: 0.210



401(k) Example:: DNN with Early Stopping

Model D:

- Input layer $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ 50 neurons $\mapsto$ output layer
  - **Early stopping** - stop after 200 epochs of no improvement and use estimates at best iteration
  - trained w/ 20% hold-out sample
- R^2 on test data: 0.222



401(k) Example: Other Tuning Choices

Structure	ℓ_2/e.s.	RMSE (in)	RMSE (out)	Validation R^2
20/20	0	55642	68122	0.138
20/20	0.1	55569	68049	0.140
20/20	1	55556	68034	0.140
20/20	e.s.	51188	64411	0.229
50/10/50	0	50938	64268	0.233
50/10/50	0.1	50958	64258	0.233
50/10/50	1	50957	64256	0.233
50/10/50	e.s.	46849	65553	0.202
20	0	61517	74145	-0.021
20	0.1	61511	74140	-0.021
20	1	61511	74140	-0.021
20	e.s.	57129	69728	0.097

401(k) Example: Could keep going...

Lots of tuning choices - could keep trying things out and attempting to fine tune

- Try different depth/width combinations
- Try different dropout rates
- Try different optimization schemes and early stopping rules
- Try different connectivity structures between layers
- Try different explicit penalizations and different penalty parameters
- ...

Stacking: Leveraging Strength of Many Learners

Stacking

Introduced several methods for building predictions all of which rely on additional tuning parameters

- Hard to know what will work best in any given actual setting
- rather than choose a single approach, can try to combine several

Stacking:

- jargon for using a linear combination of prediction rules to improve prediction accuracy
- i.e. stacking $\approx$ model-averaging
- traditional to impose linear combination coefficients are positive and sum to one
- intuitively can't do worse than choosing a single model (i.e. could just put weight one on that model)

Stacking Implementation

Implementing stacking is straightforward.

Stacking objective function using K-fold CV:

$$\min_{w_1,\ldots,w_J} \sum_{i}^{n} \left(y_i - \sum_{j=1}^{J} w_j \widehat{f}_j^{I^c_{k(i)}}(z_i) \right)^2, \qquad \text{s.t. } w_m \geq 0, \ \sum_m |w_m| = 1$$

where $\widehat{f}_j^{I^c_{k(i)}}(z_i)$ are cross-validated predicted values

- Could also estimate stacking weights in validation/test data
- Can employ other penalization methods (e.g. Lasso, Ridge)
- Can drop constraints - especially with small number of learners

401(k) Example: Stacking

We've looked at a bunch of candidate prediction approaches in asset data.

Let's see how stacking works.

Candidate learners:

1. OLS - basic
2. OLS - interactive
3. Lasso (CV) - interactive
4. Ridge (CV) - interactive
5. Random forest
6. Boosted trees - depth 3
7. Boosted trees - depth 5
8. DNN - input/50/50/output, .5 dropout on each layer
9. DNN - input/50/50/50/50/output, .5 dropout on each layer
10. DNN - input/50/50/50/50/output, early stopping

401(k) Example: Stacking Results

	In R^2	CV R^2	Test R^2	Stacking Weight
OLS - basic	0.245	0.238	0.195	0.000
OLS - flexible	0.513	-85111	0.087	0.001
Lasso (CV)	0.386	0.176	0.200	0.000
Ridge (CV)	0.351	-0.130	0.195	0.224
Random forest	0.345	0.256	0.223	0.000
Boost - depth 5	0.374	0.256	0.205	0.000
Boost - depth 3	0.322	0.261	0.187	0.045
DNN - 50/10/50	0.295	0.273	0.233	0.280
DNN - 50/50/50/50	0.420	0.288	0.188	0.000
DNN - 100/100/100/100/100	0.450	0.320	0.175	0.496
Stacking	0.420	0.420	0.213	-

Black Box Model Interpretation: Variable Importance and Partial Dependence

Black-box models

Random forests, boosted trees, neural nets are effectively “black-box” prediction algorithms.

- produce forecasts but it’s hard to really see what goes into the prediction rule
- many (all?) flexible/nonparametric procedures have this feature
  - big trees hard to think about
  - parameterized models with many interactions and nonlinear terms hard to think about

One sensible option is to view such rules as algorithms to make predictions and not try to (over-)interpret the results

- Good enough for many uses in the social sciences and industry

Variable Importance

Sometimes one wants to dig a bit deeper into a “black-box” algorithm

- Variable Importance measures aim to assess how different variables contribute to a model’s predictions

Several variable importance measures out there - consider two simple and general approaches

- variable importance via dropping columns
  - computationally intensive
- look at variable importance via permutations (original idea: Breiman (2001))
  - relatively cheap computationally
  - evaluates how important variables are to given prediction rule

Drop Column Variable Importance

Variable Importance via Dropping Columns

Input: Data $(Y, X) = \{y_j, x_j\}_{j=1}^m$, loss function $L(y, g)$ (e.g. MSE), target variable X_j

1. Compute prediction rule $\widehat{g}(X)$ using all predictors X. Compute $e_0 = L(Y, \widehat{g}(X))$
2. Compute prediction rule $\widehat{g}_j(X_{-j})$ using all predictors except predictor X_j. Compute $e_j = L(Y, \widehat{g}_j(X_{-j}))$
3. Compute drop column variable importance as $VI_j = e_j / e_0$.

- Typically compute loss on hold out data
- Typically cycle through all variables
- Can take difference instead: $VI_j = e_j - e_0$
- Variables are ranked in terms of VI_j

Permutation Variable Importance

Variable Importance via Permutations

Input: Prediction rule $\widehat{g}(X)$, data $(Y, X) = \{y_j, x_j\}_{j=1}^m$, loss function $L(y, \widehat{g})$ (e.g. MSE)

1. Compute $e_0 = L(Y, \widehat{g}(X))$
2. For each $j \in \{1, \ldots, p\}$
   - Create X_{perm} by randomly permuting feature j in X
   - Compute $e_{perm} = L(Y, \widehat{g}(X_{perm}))$
   - Compute permutation variable importance as $VI_j = e_{perm}/e_0$

- Typically compute loss on hold out data
- Can repeat step 2 many times and average
- Can take difference instead: $VI_j = e_{perm} - e_0$
- Variables are ranked in terms of VI_j

Drop Column Variable Importance in Asset Example

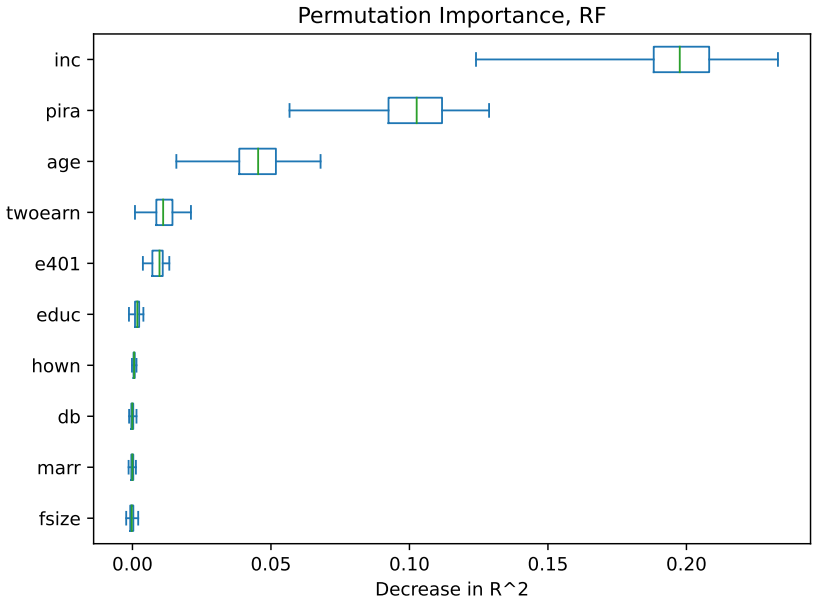
Reporting *VI*-1: Loss of fit would be $VI > 0$

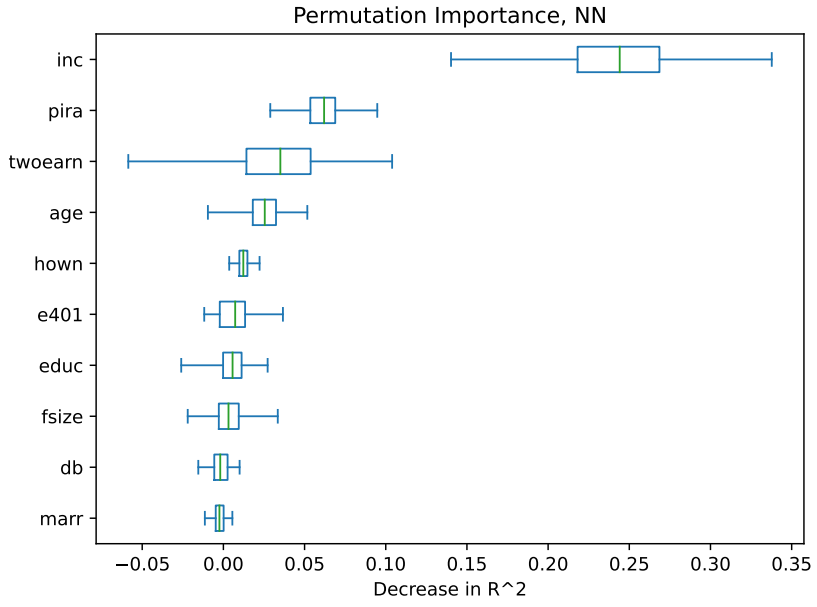
	VI (NN)	VI (RF)
inc	0.029	0.074
pira	0.033	0.079
age	0.030	0.075
hown	0.001	0.046
twoearn	-0.029	0.014
e401	0.001	0.045
educ	0.010	0.054
fsize	-0.006	0.038
marr	-0.010	0.034
db	0.003	0.047

Variable importance for random forest with leaf size of 20 and neural net (50/50/50/50)

- Recall larger numbers mean more loss of fit when variable is excluded - i.e. variable is more important to prediction quality
- No loss of fit would be $VI = 1$
- Showing VI computed on validation data (be careful with in-sample loss of fit)

Permutation Variable Importance in Asset Example





Variable importance for random forest with leaf size of 20 and neural net (50/50/50/50)

- Recall larger numbers mean more loss of fit when variable is excluded - i.e. variable is more important to prediction quality
- Reporting difference in R^2 (baselines: RF: .223, NN: .182)
- Showing VI computed on validation data (be careful with in-sample loss of fit)

Partial Dependence

Other common interpretation device is to plot estimated function (or derivatives)

Obviously hard when dimension of X is bigger than two

Instead, report "partial dependence" of function on different X variables

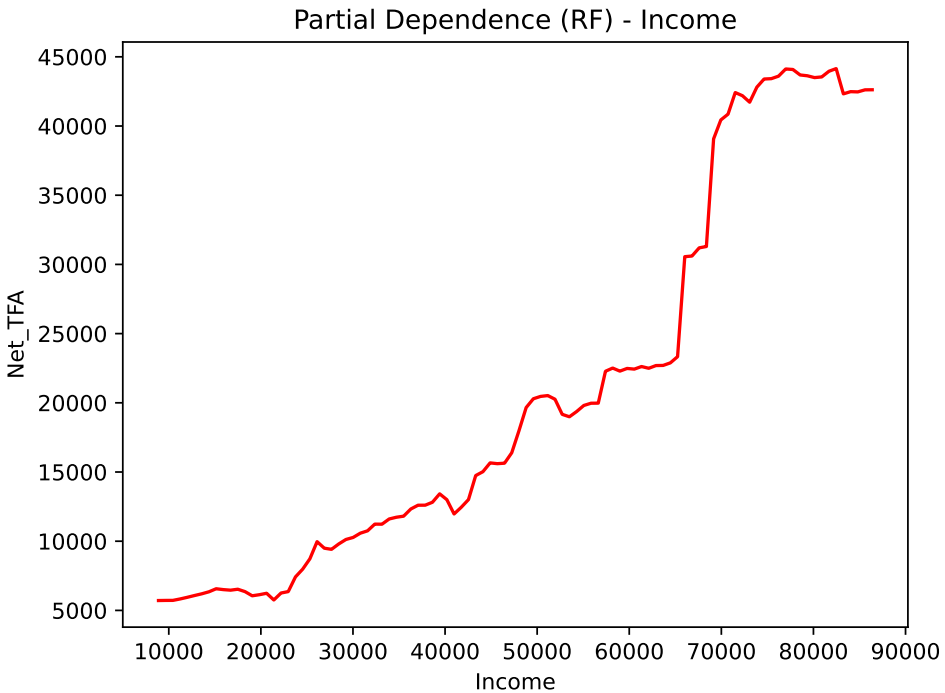
- Let X_j be a variable of interest
- Let X_{-j} be the complement of X
- Black-box prediction function: $\widehat{g}(X) = \widehat{g}(X_j, X_{-j})$
- Partial dependence is just

$$h_j(z) = \mathbb{E}_{nX_{-j}}[\widehat{g}(z, x_{-j})] = \frac{1}{m}\sum_{i=1}^{m}\widehat{f}(z, x_{i,-j})$$

  - sometimes referred to as Average Structural Function
  - could also look at derivatives (Average Marginal Effects)

Partial Dependence in Asset Example - Random Forest

Partial dependence in income



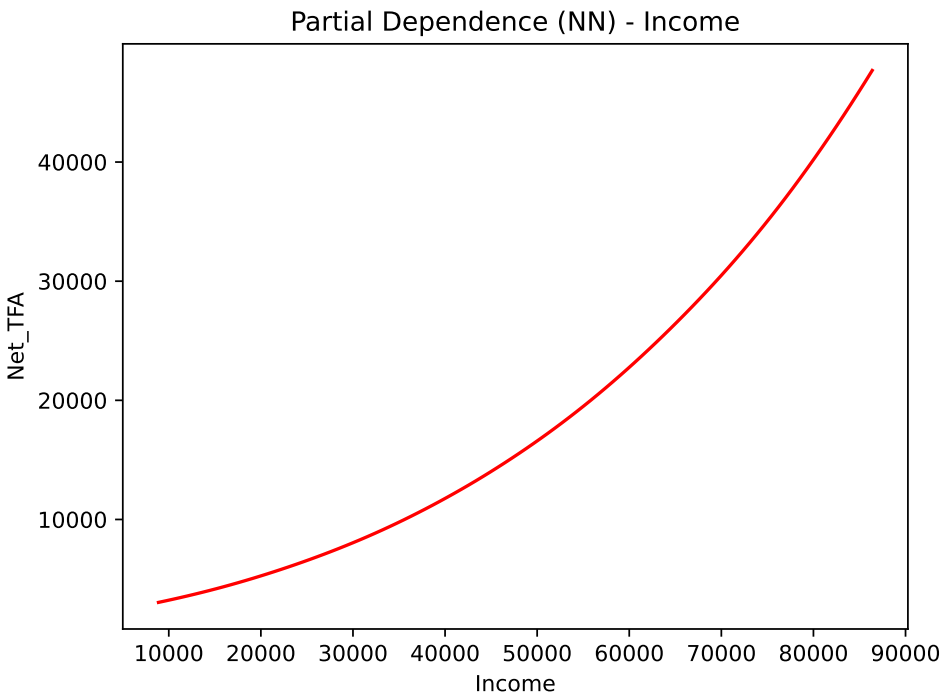
Partial dependence in `pira`

	Average Predicted Wage
`pira` = 0	6631
`pira` = 1	53994

Partial dependence from random forest with leaf size of 20

Partial Dependence in Asset Example - Random Forest

Partial dependence in income



Partial dependence in `pira`

	Average Predicted Wage
`pira` = 0	8597
`pira` = 1	27863

Partial dependence from 50/50/50/50 Neural Net

Overview of Theoretical Results

Heuristic Theoretical Result

Form of Theoretical Results

Let g be a target/population prediction rule. Let $\widehat{g}$ be an estimator of g. Assume the structure of g satisfies suitable restrictions that restrict its complexity compatible with the learner $\widehat{g}$ and regularity conditions hold. Let ξ represent the complexity of g. Then

$$\|\widehat{g} - g\| \leq Cf(n, \xi) \quad \text{where} \quad f(n, \xi) \to 0$$

for an appropriate choice of $\|\cdot\|$.

General comments:

- specifics vary depending on the estimator $\widehat{g}$
- IMPORTANT: Results do not say you get the "right" g
  - estimated prediction rule is highly correlated to g
  - many ways to produce prediction rules that look like g when input variables are correlated and/or moderate strength predictors are allowed
  - e.g. no guarantees that you get the "right" variables

Flavor of Results using Lasso

- Typical Lasso convergence result looks like
  $\sqrt{\mathrm{E}_X(\beta' X - \widehat{\beta}' X)^2} \leq C\sqrt{\mathrm{E}\epsilon^2}\sqrt{\frac{s_n \log(\max\{p,n\})}{n}}$
  - E_X denotes expectation with respect to X
  - s_n is the "effective dimension" - intuitively, number of variables with large enough coefficients to matter in sample of size n

- See, e.g., Bickel et al. (2009), Belloni and Chernozhukov (2013), Belloni et al. (2012); Belloni et al. (2016); Chernozhukov et al. (2021); Chetverikov et al. (2021); Chetverikov and Sørensen (2022) for complete statements of conditions in different settings

- $\frac{1}{n}\sum_i (x_i'\beta - x_i'\widehat{\beta})^2 = O_p(s_n \log(p)/n)$ is the best possible forecast rate
  - If you knew the right s_n regressors *ex ante*, forecast rate would be $O_p(s_n/n)$.
  - Lose $\log p$ because need to learn which regressors matter.
  - Only says your forecast rule is close to the best linear predictor $\beta' X$

References for Other Methods

Trees and related methods: See, e.g., Wager and Walther (2015), Syrgkanis and Zampetakis (2020), Ročková and van der Pas (2020)

- provide concentration bounds for (restricted) trees and forests
- restrictions do not actually cover the way many tree-methods are used in practice
- restrictions on complexity of g impose that only a few variables matter and target function is appropriately smooth

DNN: See, e.g., Schmidt-Hieber (2020), Farrell et al. (2021), Polson and Ročková (2018)

- provide concentration bounds for DNN with modest number of layers and neurons per layer at global optimum of learning problem
- restrictions on complexity of g impose that only a few variables matter at each layer and that target function is appropriately smooth

Recent ML literature looking at highly overparameterized settings (e.g. such that training data is perfectly fit) and taking optimization approach into account developing theory (not assuming reach global optimum)

- e.g. Montanari and Zhong (2022), Dou and Liang (2021)

Conclusion

Commentary

Some concluding thoughts on takeaways:

- rarely know the right specification/model/learner
  - try several
  - use sensible aggregates/ensembles (essentially the “super-learner” of van der Laan and Rose (2011))
    - accurately report what you’ve done

Other things to look at

We've just hit some key models

Lots of other topics:

- Other learners: SVM, kernel methods, Bayesian methods, ...
- Classification
- Matrix completion/Factor models
- Clustering (e.g. k-means, hierarchical clustering)
- Many modifications to DNN
- Auto ML
- ...

Software used

Stata + python +

- Scikit-learn - Pedregosa et al. (2011)
  - nice Python library of ML tools
- lassopack - Ahrens et al. (2020)
  - regularized regression in Stata
  - lots of other options now
- rforest - Schonlau and Zou (2020)
  - pure Stata implementation of random forest
- pystacked - Ahrens et al. (2023)
  - many ML tools in Stata, including stacking

References

References

Achim Ahrens, Christian B. Hansen, and Mark E. Schaffer. lassopack: Model selection and prediction with regularized regression in stata. *The Stata Journal*, 20(1):176–235, 2020. doi: 10.1177/1536867X20909697. URL https://doi.org/10.1177/1536867X20909697.

Achim Ahrens, Christian B. Hansen, and Mark E. Schaffer. pystacked: Stacking generalization and machine learning in stata. *The Stata Journal*, 23(4):909–931, 2023.

Susan Athey and Guido Imbens. Machine learning methods that economists should know about. *Annual Review of Economics*, 11:685–725, 2019.

Alexandre Belloni and Victor Chernozhukov. Least squares after model selection in high-dimensional sparse models. *Bernoulli*, 19(2):521–547, 2013. ArXiv, 2009.

Alexandre Belloni, Daniel L. Chen, Victor Chernozhukov, and Christian B. Hansen. Sparse models and methods for optimal instruments with an application to eminent domain. *Econometrica*, 80 (6):2369–2429, 2012. Arxiv, 2010.

Alexandre Belloni, Victor Chernozhukov, Christian Hansen, and Damian Kozbur. Inference in high-dimensional panel models with an application to gun control. *Journal of Business and Economic Statistics*, 34(4):590–605, 2016.

Peter J. Bickel, Ya'acov Ritov, and Alexandre B. Tsybakov. Simultaneous analysis of lasso and dantzig selector. *Annals of Statistics*, 37(4):1705–1732, 2009.

Leo Breiman. Bagging predictors. *Machine learning*, 26:123–140, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Victor Chernozhukov, Wolfgang Karl Härdle, Chen Huang, and Weining Wang. Lasso-driven inference in time and space. *The Annals of Statistics*, 49(3):1702–1735, 2021.

Denis Chetverikov and Jesper Riis-Vestergaard Sørensen. Analytic and bootstrap-after-cross-validation methods for selecting penalty parameters of high-dimensional m-estimators. *arXiv preprint arXiv:2104.04716*, 2022.

Denis Chetverikov, Zhipeng Liao, and Victor Chernozhukov. On cross-validated lasso in high dimensions. *Annals of Statistics*, 49(3):1300–1317, 2021.

Xialiang Dou and Tengyuan Liang. Training neural networks as learning data-adaptive kernels: Provable representation and approximation benefits. *Journal of the American Statistical Association*, 116(535):1507–1520, 2021.

Max H. Farrell. Robust inference on average treatment effects with possibly more covariates than observations. *Journal of Econometrics*, 189(1):1–23, 2015.

Max H. Farrell, Tengyuan Liang, and Sanjog Misra. Deep neural networks for estimation and inference. *Econometrica*, 89(1):181–213, 2021. URL `https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA16901`.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–407, 2000.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. Random design analysis of ridge regression. In *Conference on learning theory*, volume 23, pages 9.1–9.24. JMLR Workshop and Conference Proceedings, 2012.

Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks. In Jacob Abernethy and Shivani Agarwal, editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 2306–2327. PMLR, 09–12 Jul 2020. URL https://proceedings.mlr.press/v125/kidger20a.html.

Guillaume Lecué and Charles Mitchell. Oracle inequalities for cross-validation type procedures. *Electronic Journal of Statistics*, 6:1803–1837, 2012.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6232–6240, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. When and why are deep networks better than shallow ones? In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 2343–2349. AAAI Press, 2017.

Andrea Montanari and Yiqiao Zhong. The interpolation phase transition in neural networks: Memorization and generalization under lazy training. *Annals of Statistics*, 50(5):2816–2847, 2022.

Sendhil Mullainathan and Jann Spiess. Machine learning: An applied econometric approach. *Journal of Economic Perspectives*, 31(2):87–106, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Nicholas G Polson and Veronika Ročková. Posterior concentration for sparse deep learning. *Advances in Neural Information Processing Systems*, 31, 2018.

James M Poterba, Steven F Venti, and David A Wise. Do 401 (k) contributions crowd out other personal saving? *Journal of Public Economics*, 58(1):1–32, 1995.

Veronika Ročková and Stéphanie van der Pas. Posterior concentration for bayesian regression trees and forests. *The Annals of Statistics*, 48(4):2108–2131, 2020.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

Johannes Schmidt-Hieber. Nonparametric regression using deep neural networks with relu activation function. *Annals of Statistics*, 48(4):1875–1897, 2020.

Matthias Schonlau Schonlau and Rosie Yuyan Zou. The random forest algorithm for statistical learning. *The Stata Journal*, 20(1):3–29, 2020. doi: 10.1177/1536867X20909688. URL https://doi.org/10.1177/1536867X20909688.

Vasilis Syrgkanis and Manolis Zampetakis. Estimation and inference with trees and forests in high dimensions. In Jacob Abernethy and Shivani Agarwal, editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 3453–3454. PMLR, 09–12 Jul 2020.

Mark J. van der Laan and Sherri Rose. *Targeted learning: causal inference for observational and experimental data*. Springer Science & Business Media, 2011.

Stefan Wager and Guenther Walther. Adaptive concentration of regression trees, with application to random forests. *arXiv preprint arXiv:1503.06388*, 2015.

Marten Wegkamp. Model selection in nonparametric regression. *The Annals of Statistics*, 31(1): 252–273, 2003.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.