Machine Learning (ML) in Economics

Questions/comments/expressions of discontent:
steve.j.miller@colorado.edu

Very High-Level Schedule

Day 1: Tools

- 9-10: Overview: machine learning goals, how they might help economists
- 10-12: Introduction to common machine learning methods
- 12-1: Lunch & brainstorm applications to your own work
- 1-4: Machine learning in the service of causal inference

Day 2: Applications

- 9-10 Other uses: imagery, text, dynamic programming
- 10-12 Applications and practical tips

Availability of materials

Slides, code, and data are all available here:

stevejmiller.com/ml

Before we start

You'll probably want to

1) Install package compilation tools (for packages that need to be installed from from source code)

   Windows: https://cran.r-project.org/bin/windows/Rtools/rtools42/rtools.html
   Mac: xcode command line tools (e.g., via `xcode-select --install`)

2) Run the Installer.R script on stevejmiller.com/ml

Caveats

1)  ,  , … >> me

2) This is a **rapidly** evolving area.

Goals & high level uses of ML for economists

What ML is and isn't
What ML is good at
Overview of applications within econ

What the heck is machine learning anyway?

To me: a collection of non- or semi-parametric tools that are really good at

- Prediction, both continuous and discrete (classification)
- Function approximation
- Dimensionality reduction

Why? They all offer a way to trade off fit and complexity.
The ability to do that a) improves generalization and b) makes them adaptive to function features (e.g., smoothness), improving convergence rates

What counts as machine learning is nebulous.

We'll focus on methods economists are less likely to be familiar with.

Common classification of ML algorithms

Supervised

"Train" model on examples of correct answers that we provide

e.g. LASSO, random forests, neural networks. Use both "x" and "y" data

→ Closest to regression tools we're familiar with

Unsupervised

Just look for patterns without knowledge of what's "correct"

e.g. clustering algorithms that group observations using only "x" data

Common view of ML tools: Prediction & classification

Task: ***out-of-sample*** assignment of a number or category (a.k.a. label) to a set of predictors (a.k.a. features)

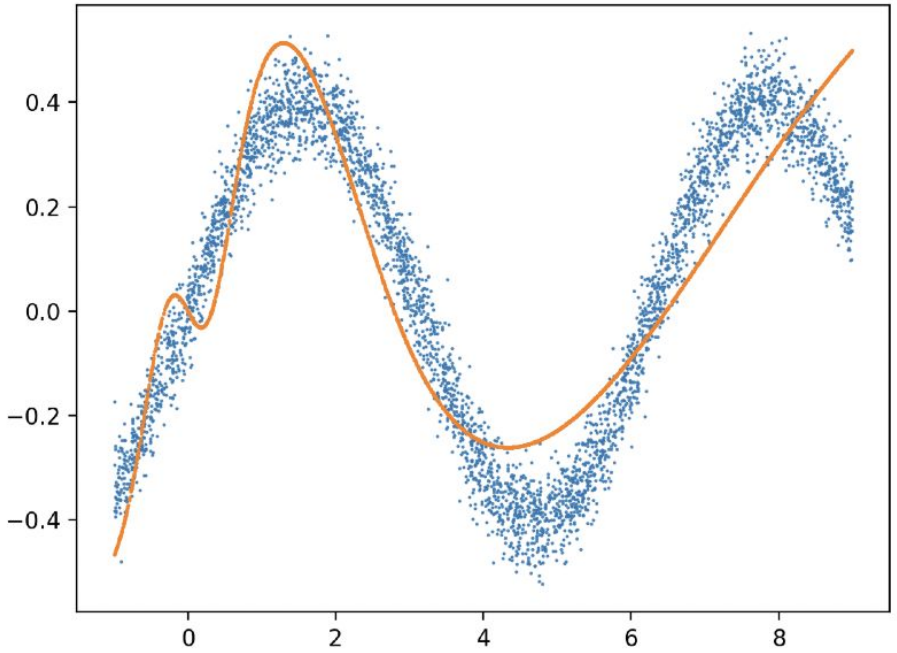
How do we evaluate success?

- **Continuous (regression)**
  - Most often, Root-mean-squared error (RMSE)
- **Discrete (classification)**
  - Confusion matrix (not a single #)
  - False (or true) positive (or negative) rates.
  - Accuracy (fraction correct)
  - Many others.

		Actual	
		Class 1	Class 2
Predicted	Class 1	511	74
	Class 2	35	613

Machine learning as function approximation

Prediction (by ML or any other tool) gives us function approximation: we can assess the value of a function at unknown input points.



Machine learning for dimensionality reduction

With more and more data, we might want (additional) principled ways to collapse high-dimensional data to a lower-dimensional representation for use in standard econometric models.

- Satellite images
- Text
- “Wide” data (e.g., knowing lots about each user of a web site)

These can be learned in supervised or unsupervised fashion.

These skills are still useful to us as economists

1. **… for prediction when that's what we're after.**


Machine Learning Approaches for Predicting Willingness to Pay for Shrimp Insurance in Vietnam

Kim Anh Thi Nguyen, Tram Anh Thi Nguyen, Brice M. Nguelifack, and Curtis M. Jolly

Beyond Early Warning Indicators: High School Dropout and Machine Learning†

Dario Sansone

2. **Generating structured data ("extraction")**

3. **Approximating messy quantities (e.g., value functions, nuisance functions)**

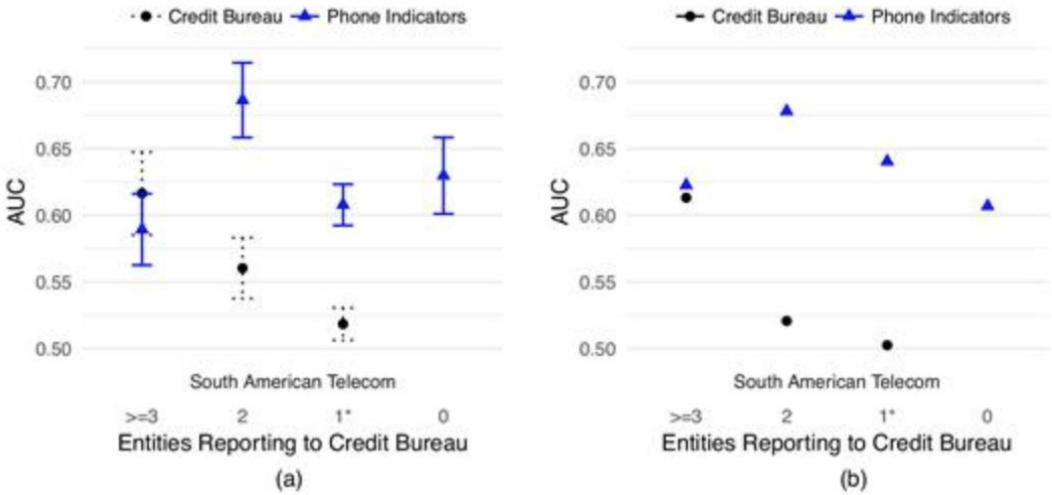
4. **Predicting intermediate quantities in causal inference**
(propensity scores, first stage estimates, etc.)

5. **Choosing among many predictors, interactions, when theory gives insufficient guidance**
(can give causal estimates under certain assumptions)

Standard prediction problems: credit example

Björkegren & Grissen (2020) seek to predict loan repayment – including using mobile phone predictors – to enable expansion of credit access to more customers (L to R on graphs are customers with less existing credit history)

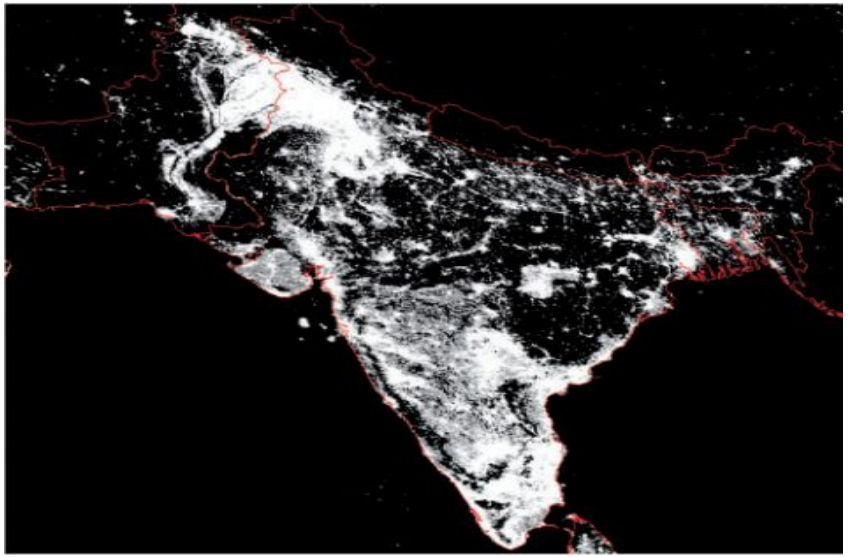
Figure 3.



See also Aiken et al (2023) for an example of anti-poverty program targeting in Afghanistan

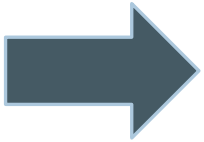
Data extraction

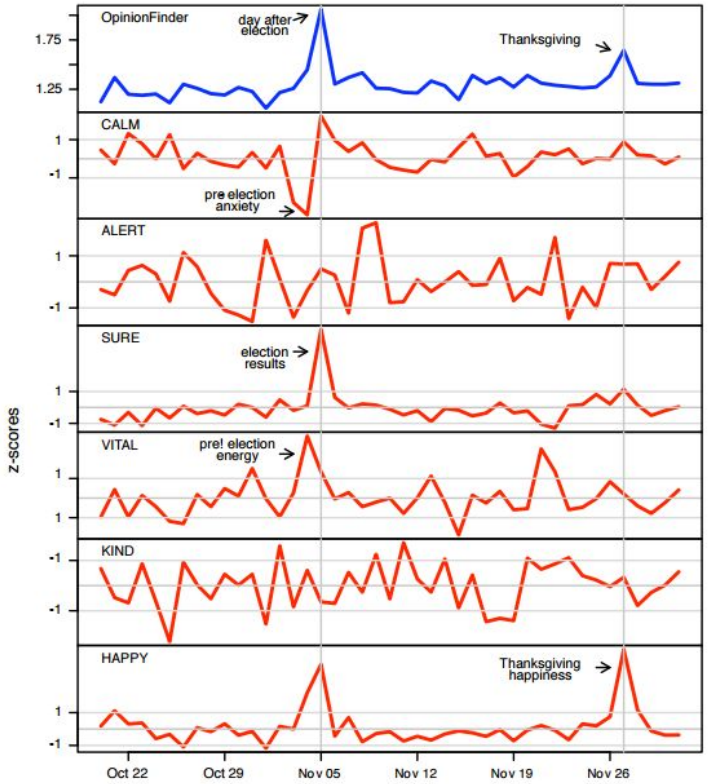
We may want to collapse data from complex sources like satellite imagery into a single number representing an economically meaningful quantity (e.g., wealth in a village)



Pinkovskiy and Sala-i-Martin (QJE 2016)

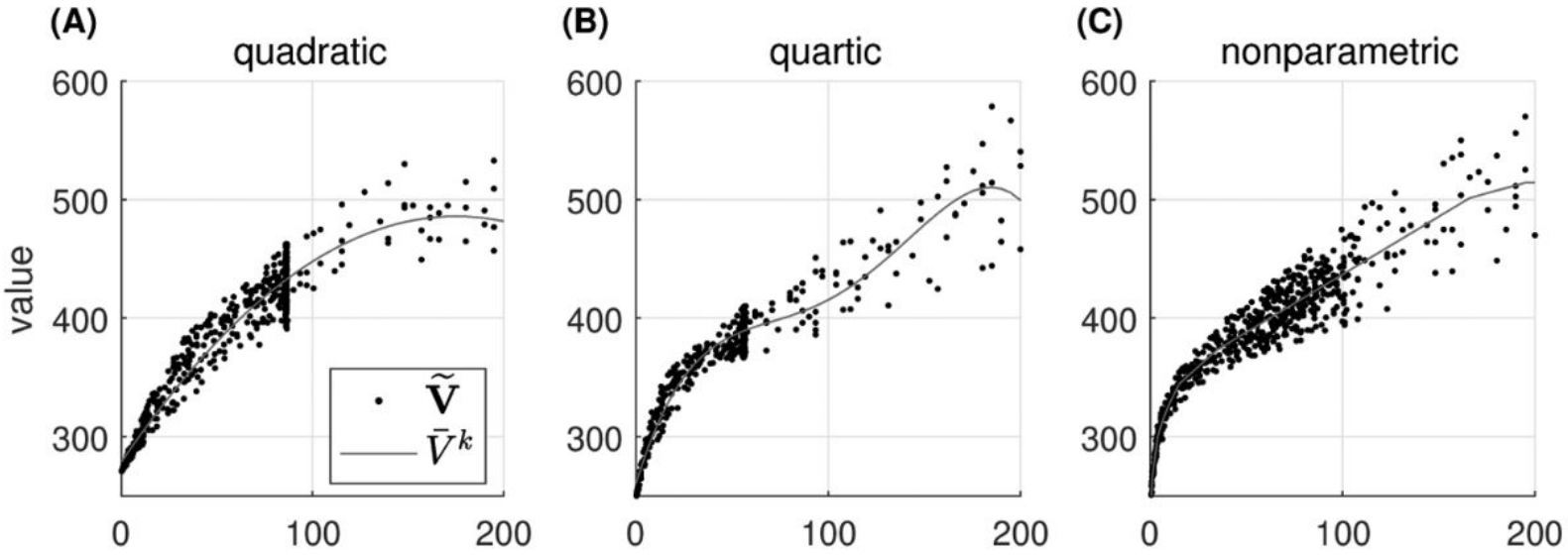
More data extraction





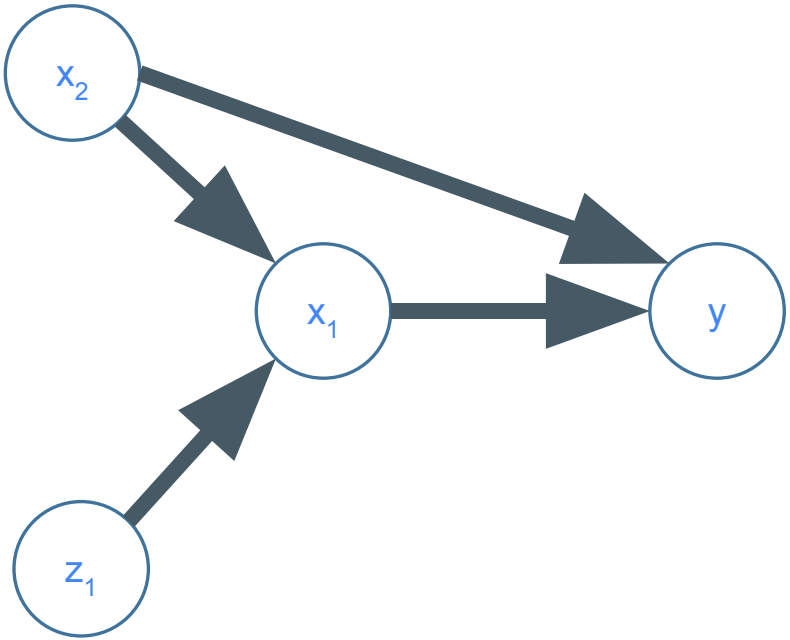
Value function approximation (Springborn & Faig, 2019)

Can be used in conjunction with approximate dynamic programming



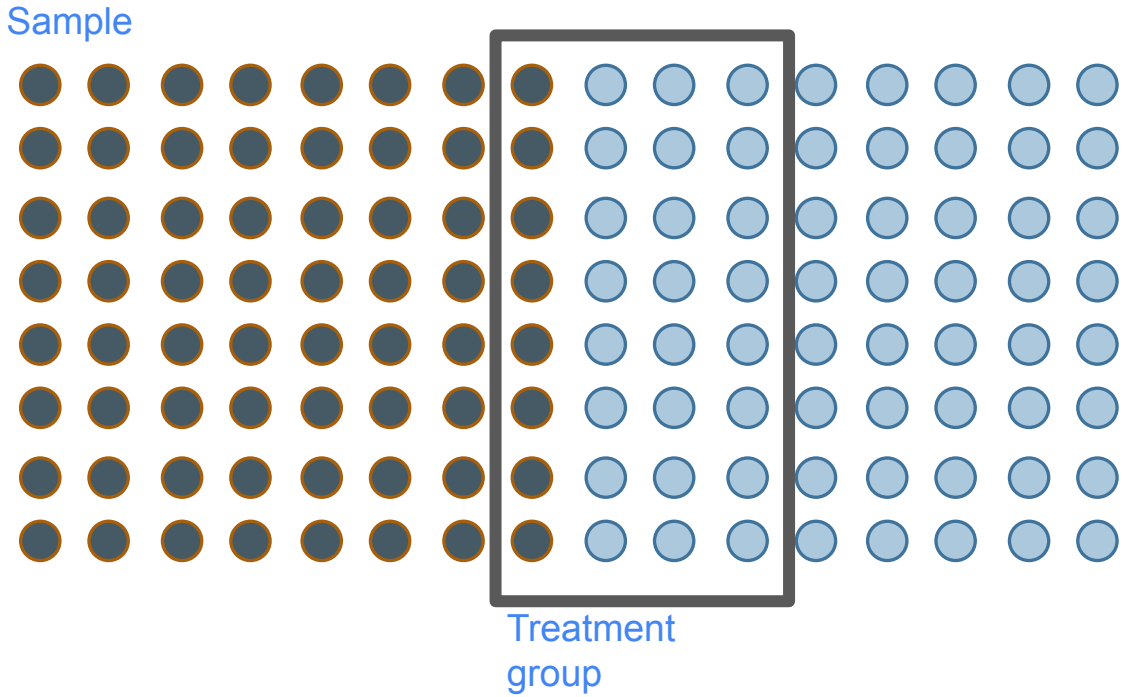
Predicting intermediate quantities

Remember 2SLS? The first stage is just a prediction problem.



Predicting propensity scores

We don’t necessarily care why a propensity score is what it is. We just want to know estimate it well, so we can (under some assumptions) adjust for selection bias.



Predicting counterfactuals

Average treatment effect:

$$E[Y_i(1) - Y_i(0)]$$

Outcome if treated — Outcome if not treated

For treated observations, we just need to know $Y_i(0)$. We don't really care why it is what it is

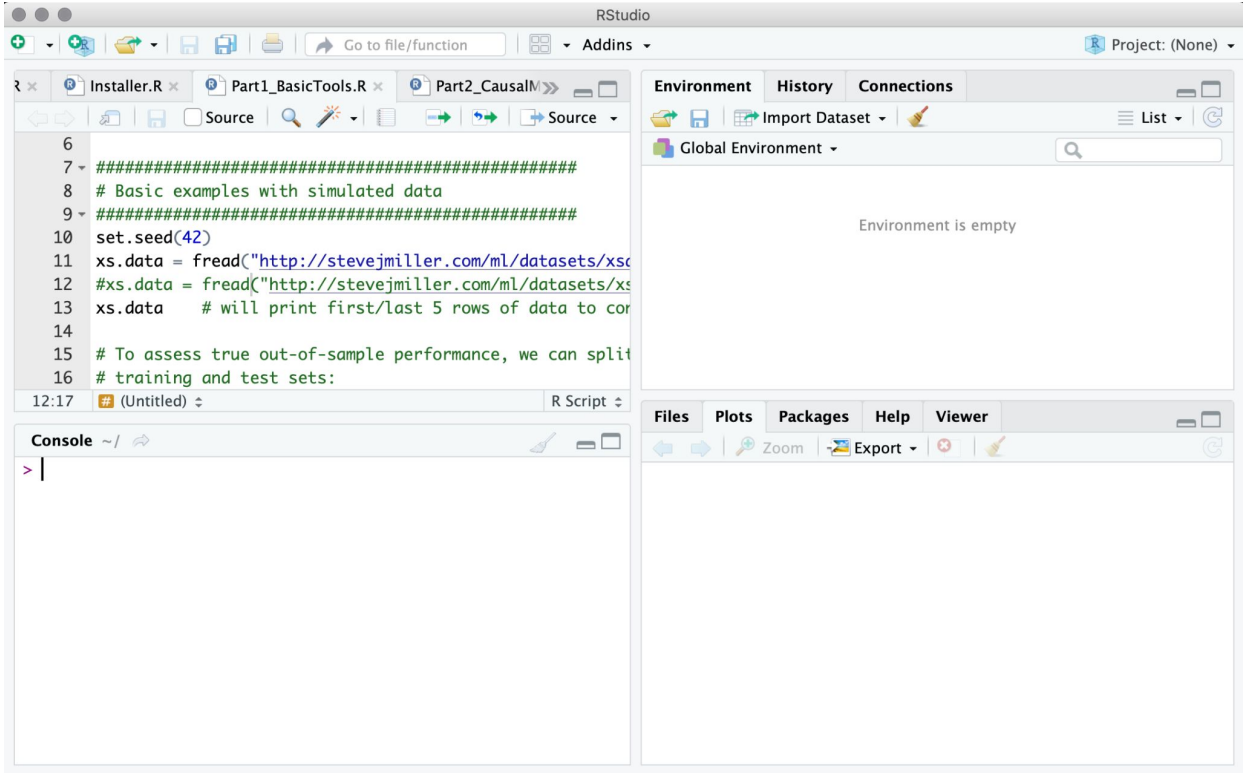
BRIEF
R/Rstudio primer

If you've programmed in almost any actual language, this won't be painful

RStudio is an integrated development environment for R

R does the actual work, but RStudio makes your life a lot easier.

Where you should write & save code



What's in memory

Where code actually gets run

Plots, auxiliary stuff

Basic R statements

R programs are basically a list of instructions called statements:

```
time.in.workshop = 1
excitement = exp(-time.in.workshop)
```

You can also add comments to your code, preceded by a #

```
time.in.workshop = 1
excitement = exp(-time.in.workshop) # Is it Friday yet?
```

Write less code: use functions

Like many programming languages, R also has more advanced capabilities wrapped up in functions. A function is simply a series of statements that is given a name. Examples:

```
hours.slept = c(5,12,1,18,0,3,10) # combine into a vector

mean(hours.slept)      # take a guess

sum(hours.slept)       # can't imagine what this does

summary(hours.slept)   # min, max, mean, 25/50/75 quantiles
```

What's in a function?

Each function has a name, (possibly) arguments, and return values

The ***name*** is how you identify the function (mean, sum, etc).

The ***arguments*** (or *parameters*) are a comma separated series of variables you give to the function so it can do its job with your data. They go between parentheses

The *return* ***value*** (or value) is the result of the calculations.

```
value = function.name(arg1, arg2, arg3)
```

Write less code: use packages

Like many programming languages, R also has more advanced capabilities wrapped up in packages. A package is a collection of related functionality that you can download and install (usually including functions you can call):

1. Install the package (**only do this *once***)

   ```
   install.packages('haven')
   ```

2. Load (attach) the package using the `library` function:

   ```
   library('haven')
   ```

3. Use (or get help on) the functions in the package

   ```
   ?read_dta  #get help on read_dta (for Stata data files)
   ```

A quick/dirty data analysis example

```
# load a package I prefer for working with tabular data
library(data.table)

# read in data (can be local or a url)
my.data = fread("http://stevejmiller.com/ml/datasets/xsdata_treatment.csv ")

# make an outcome histogram
hist(my.data$y)

# estimate a simple linear regression
mod.res = lm(formula = y ~ treatment + x1 + x2, data = my.data)

# summarize the results of estimation
summary(mod.res)
```

Common Machine Learning Algorithms

Penalized regression/shrinkage

Support vector machines/regression

Tree-based approaches

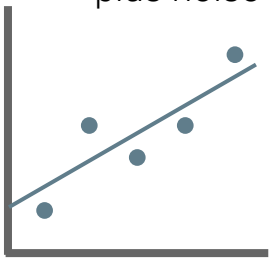
Neural networks

Penalized regression /regularization /shrinkage

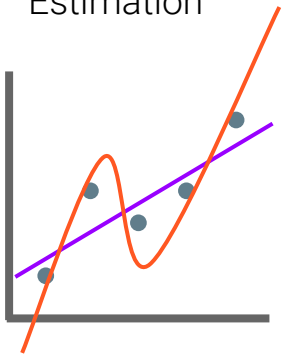
Idea #1: control overfitting

If we care about prediction out of sample, we have to be careful about overfitting a model to the data we do have

DGP: linear model plus noise



Estimation



Cubic model fits data better in sample, but may “chase” or “overfit” sample points, harming out-of-sample prediction

This is often a bias vs. variance tradeoff (in a prediction sense)

- Linear model has higher bias
- Cubic model has higher variance

Quantifying tradeoffs in prediction tools

One prediction objective: minimize mean squared error *out of sample*

$$E[(y - \hat{y}(x))^2] = E[(\hat{y}(x) - y)]^2 + E[(\hat{y}(x) - E[\hat{y}(x)])^2] + \sigma^2$$

Mean squared error

Bias squared

Variance

Irreducible (process) error

This decomposition suggests we can improve prediction by adjusting bias vs. variance of our prediction tool. For an analogous starting point, think about adjusted R^2

Idea #1: directly penalize variance of our predictor

Out-of-sample prediction variance comes from model complexity. Penalize it.

In a linear model, minimize the sum of squared residuals plus a penalty term

$$\hat{\beta}_{pen} = \min_{b} \sum_{i=1}^{n} (y_i - x_i' b)^2 + \lambda C(b)$$

Where C(b) quantifies model complexity, and λ weights in-sample fit vs. complexity

Penalized regression: popular flavors

1. **Ridge regression**

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

2. **LASSO** (least absolute selection and shrinkage operator)

$$C(b) = \sum_{j=1}^{k} |b_j|$$

Most econ applications use this

3. **Elastic Net:** weighted average of LASSO + ridge penalties

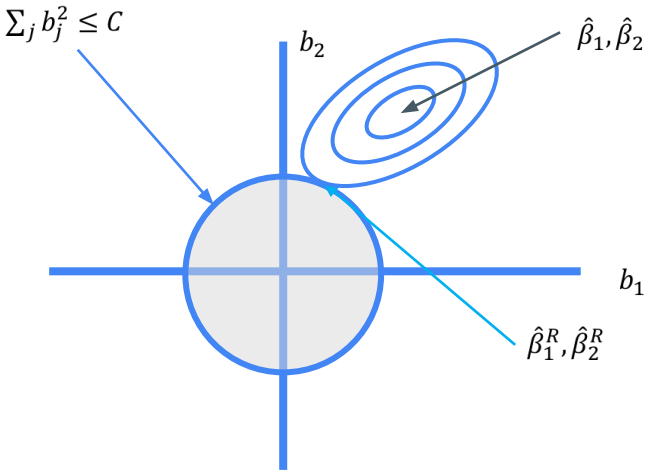
$$C(b) = \alpha_{LASSO} \sum_{j=1}^{k} |b_j| + (1 - \alpha_{LASSO}) \sum_{j=1}^{k} b_j^2$$

Graphical intuition: ridge & LASSO

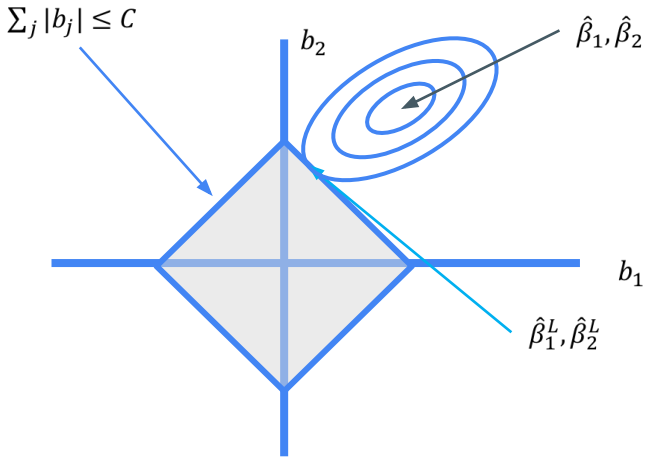
Penalty terms can be rewritten as constraints, e.g., $\hat{\beta}_{pen} = \min_b \sum_{i=1}^{n} (y_i - x_i' b)^2$

$$s.t.\ C(b) \leq \bar{C}$$

Ridge regression



LASSO

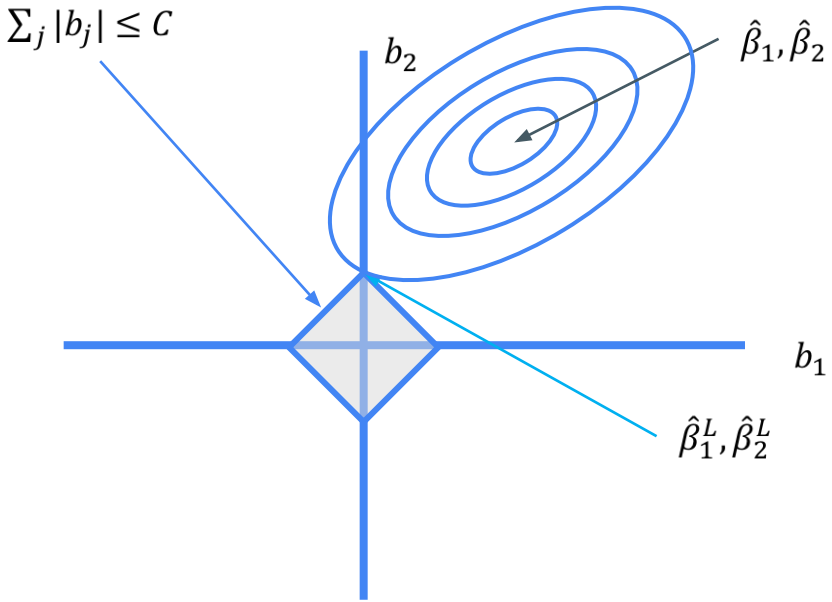


LASSO: variable selection/sparsity

As we shrink the constraint (make λ bigger), we can end up with some coefficients being equal to zero.

→ LASSO does automated variable selection.

It selects at most *n* non-zero parameters (where n is sample size)



When might LASSO make sense?

You have some reason to believe in sparsity: some of your coefficients are actually (approximately) zero but you don't have sufficient guidance on which ones.

- Maybe you have multiple measures of an abstract concept but don't know which one is best

- Aren't sure about functional form, so you include polynomials, interaction terms, etc.

→ This doesn't mean you should just use LASSO in place of OLS. It's still biased.
This just gives guidance on why sparsity might arise.
We'll come back to how to use it econometric-legally.

Penalized regression variants

There are all kinds of variants

- Penalize coefficients differently based on prior knowledge (e.g., from another estimation technique) → Adaptive LASSO

- Penalize groups of coefficients differently → Group LASSO

- Apply the same penalty ideas to other estimation objectives e.g., penalized maximum likelihood

Choosing the penalty weight (λ)

Most common approach: grid search & cross-validation

- **Grid search:** Choose a range of λ values (grid), and choose the one with the best approximated out-of-sample performance
- **Cross-validation:** approximate out-of-sample performance.
  Set aside ("hold out") a sampled fraction of your data. Often 20% or 10%
  - Train (estimate) on the remaining 80-90% of your data
  - Assess performance (e.g., root mean squared error) on the held out 10-20% as an approximation of out-of-sample performance.
  - Often: repeat this process *K* times (K-fold cross validation), each time holding out a different subset of your data. E.g., do this 10 times, each time holding out a different 10%.
  - Can repeat the entire cross-validation process too, since it's a random process

This same principle is used in most methods we'll see.

Cross-validation, visually

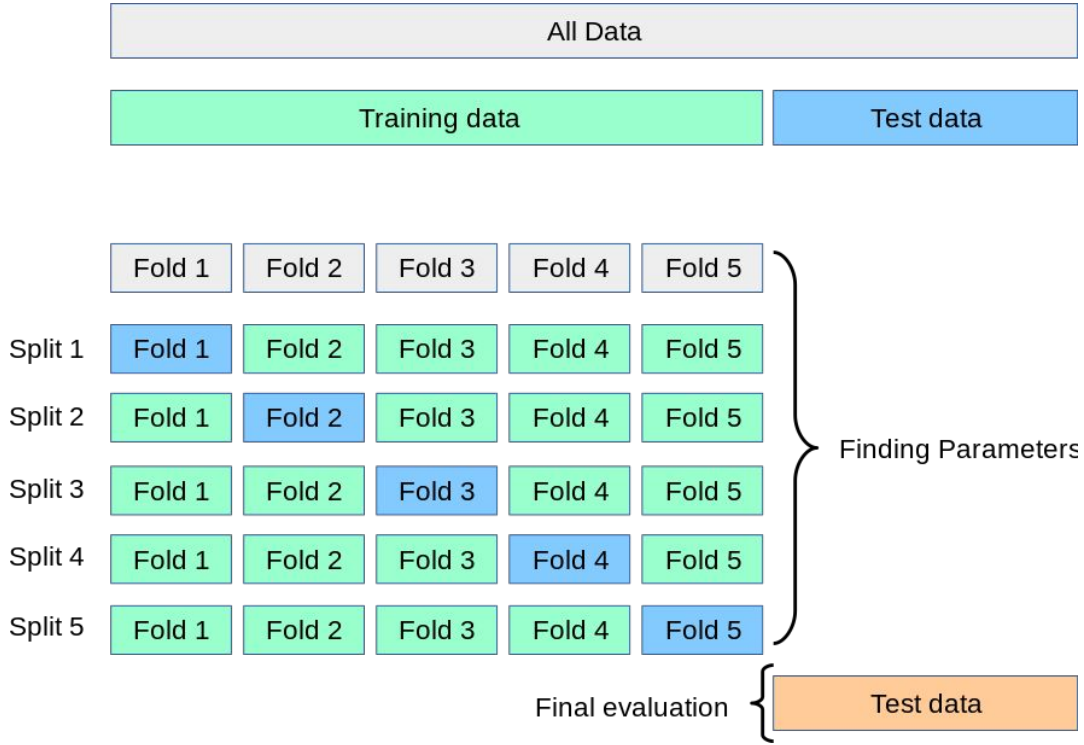


Image: scikit-learn documentation (a Python library)

Shrinkage in R

Example: predicting household emissions

Relate per-capita emissions at household level to demographics, expenditures, other data.

Using R (via RStudio), we'll

- Load the data (provided as a .dta file) using the **haven** package
- Train a LASSO model using the **caret** package
- Output & assess results

→ Let's go look at RStudio for details

ELSEVIER

Energy Economics

Volume 92, October 2020, 104942

Prioritizing driving factors of household carbon emissions: An application of the LASSO model with survey data

Xunpeng Shi [a], Keying Wang [b c], Tsun Se Cheong [d], Hongwu Zhang [e]

Show more

+ Add to Mendeley Share Cite

https://doi.org/10.1016/j.eneco.2020.104942

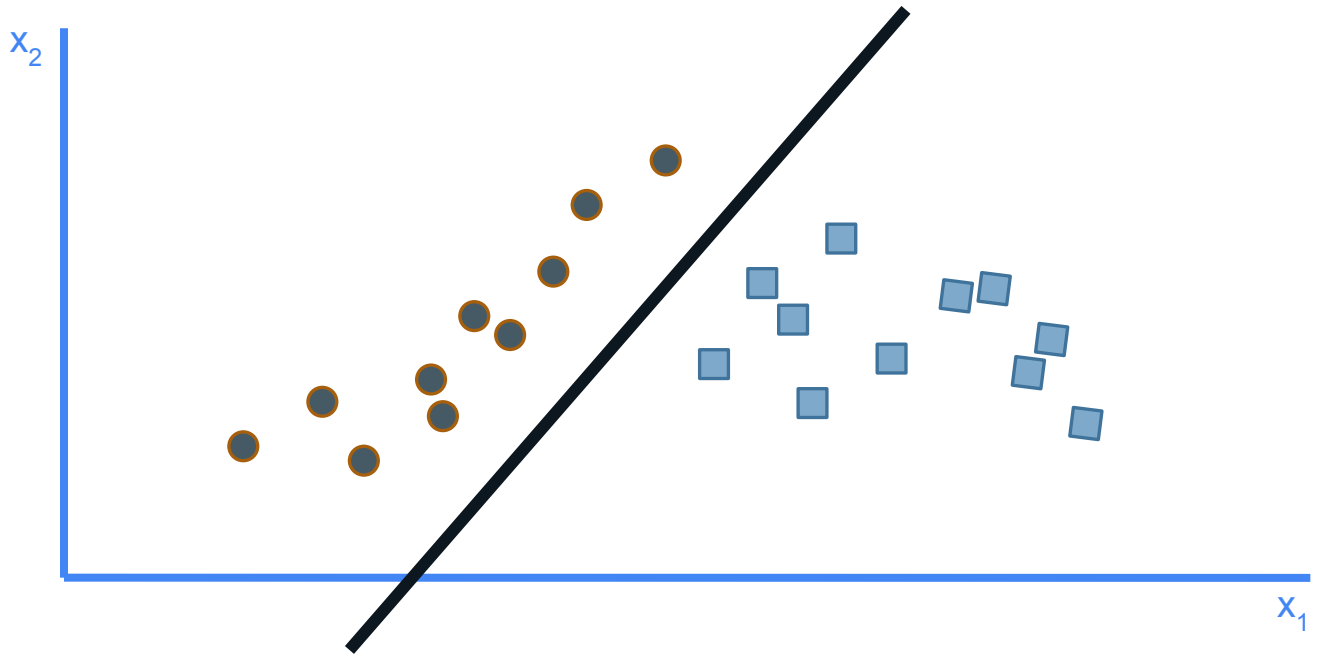
Get rights and content

Support vector machines/regression

Support vector machines

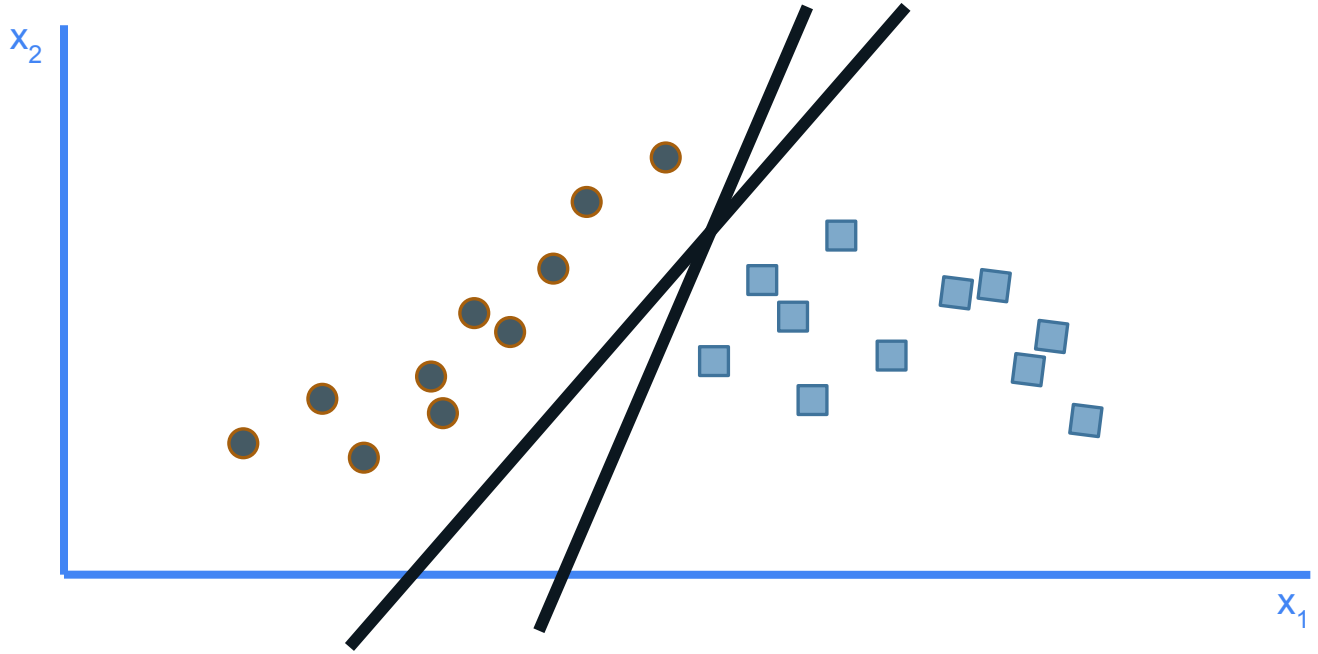
Motivation: classify inputs into categories (we'll see a variant for regression)

Approach: draw a line (hyperplane) separating observations from different categories



Support vector machines

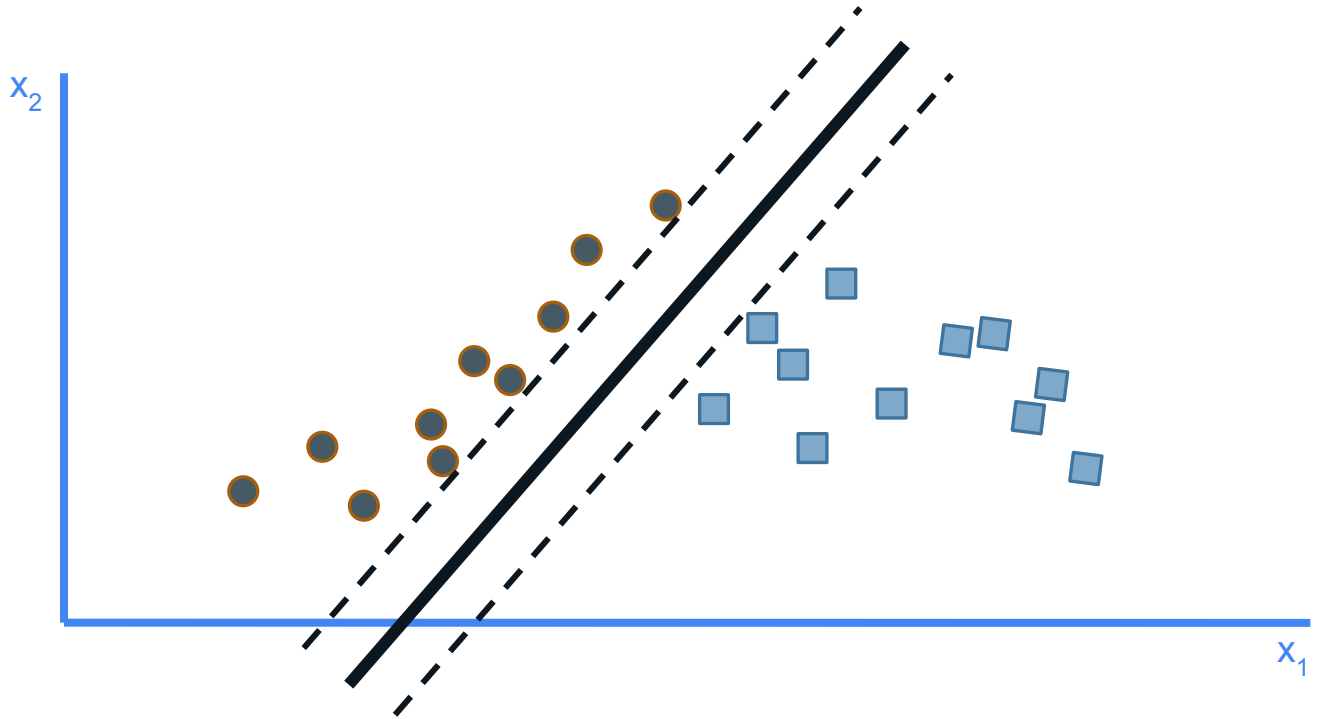
Issue 1: Many lines might work. Which one should we choose?



Support vector machines

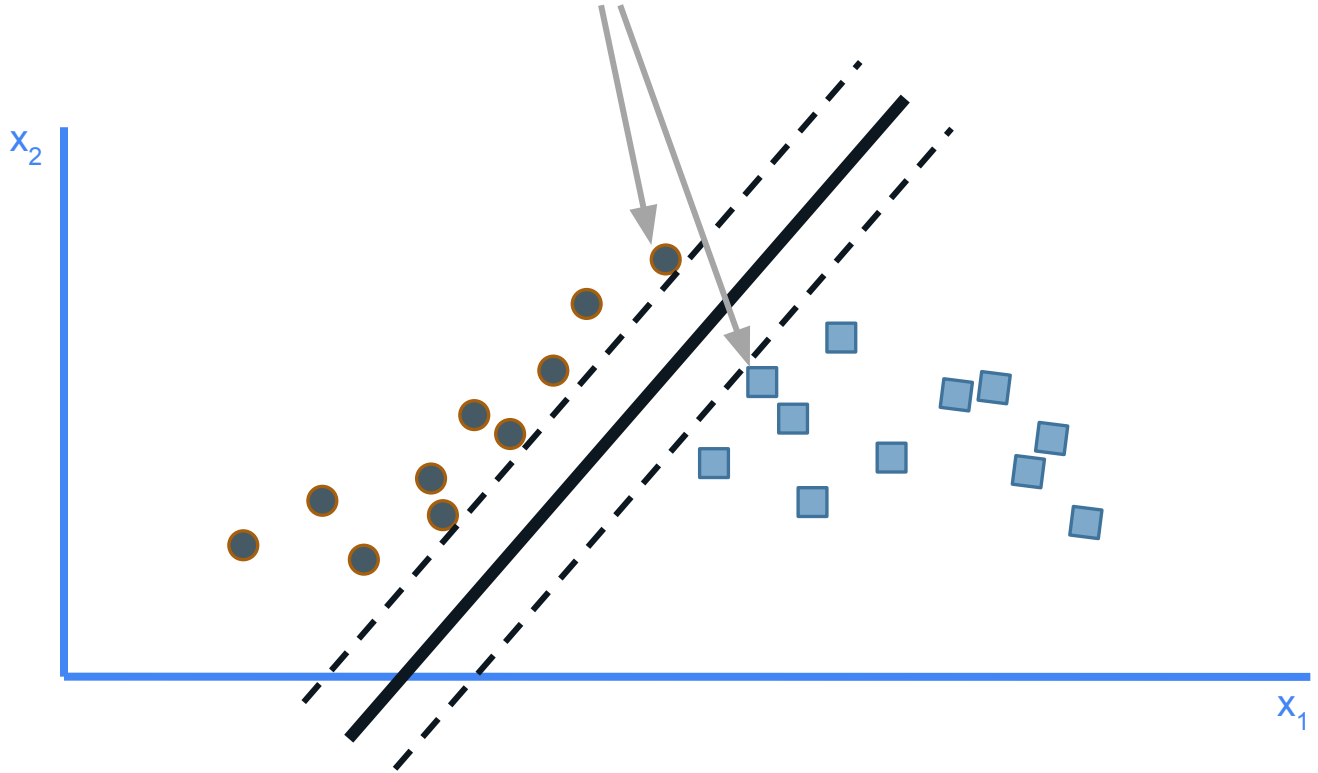
Issue 1: Many lines might work. Which one should we choose?

A: Pick the line with the largest ‘margin’ – distance to nearest points on either side



SVMs: naming digression

Those nearest points are the ‘support vectors’. They determine the classification rule.



SVM: math preliminaries

Denote our two categories by y=1 and y=-1

We can represent a hyperplane as

$$X\beta + b = 0$$

Points above a hyperplane will have $x_i^T \beta + b > 0$, while points below have $x_i^T \beta + b < 0$

Denote the distance from a support vector x_s to the hyperplane as *M* (the margin):

$$M \equiv \frac{|x_s^T \beta + b|}{\|\beta\|}$$

Why this? Let x^* be a point on the hyperplane. Then project $x_s - x^*$ onto the hyperplane's normal vector β and take the magnitude of that projection.

SVM: objective

Our goal will then be to choose β, b to maximize M, subject to the constraint that all points must be at least a distance M from the hyperplane.

We can write this as

$$\max_{\beta,b} M$$

$$\text{such that}$$

$$y_i \frac{\left(x_i^T \beta + b\right)}{\|\beta\|} \geq M \ \forall i$$

SVM: key duality trick

Note that for $X\beta + b = 0$, we can multiply β, b by a constant and get the same hyperplane.

We'll conveniently choose β, b such that $\|\beta\| = \frac{1}{M}$. Why?

Because we can rewrite our problem as

$$\min_{\beta,b} \frac{1}{2}\|\beta\|^2$$
$$s.t.$$
$$y_i\left(x_i^T\beta + b\right) \geq 1 \; \forall i$$

Or the Lagrangian

$$\min_{\beta,b} \frac{1}{2}\|\beta\|^2 - \sum_i \lambda_i\left[y_i\left(x_i^T\beta + b\right) - 1\right]$$

Calculus lets us express β in terms of x, y, and λ, and b drops out. We then simply optimize (**but maximize!**) over λ, and substitute back in to get β and b.

It ends up that the only nonzero λ_i are those for the support vectors

SVM: key duality result

Margin maximization eventually reduces to the following problem

$$\max_{\lambda_i} \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j \text{ s.t. } \lambda_i \geq 0$$

Note:

1. This is easier to solve computationally. One vector of non-negative parameters λ_i

Our predictors only enter in terms of their inner product $\langle x_i, x_j \rangle$

Intuition: inner product captures similarity (part of cosine similarity)

This is also true in the dual for ridge regression → Kernel Ridge Regression

SVM: calculus

Derivatives (gradients) with respect to β and b give us

$$\beta = \sum_i \lambda_i y_i x_i \quad \text{and} \quad \sum_i \lambda_i y_i = 0$$

Substitute the expression for β into the Lagrangian

$$\max_{\lambda_i} \min_b \frac{1}{2}\left\|\sum_i \lambda_i y_i x_i\right\|^2 - \sum_i \lambda_i \left[y_i\left(x_i^T \sum_j \lambda_j y_j x_j + b\right) - 1\right]$$

$$\max_{\lambda_i} \min_b \tfrac{1}{2}\|\textstyle\sum_i \lambda_i y_i x_i\|^2 - \sum_i \lambda_i \left[y_i\left(x_i^T \sum_j \lambda_j y_j x_j\right)\right] - b\sum_i \lambda_i y_i + \sum_i \lambda_i$$

$$\max_{\lambda_i} \frac{1}{2}\left\|\sum_i \lambda_i y_i x_i\right\|^2 - \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_i \lambda_i$$

$$\max_{\lambda_i} \frac{1}{2}\sum_k \left(\sum_i \lambda_i y_i x_{ik}\right)^2 - \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_i \lambda_i$$

$$\max_{\lambda_i} \frac{1}{2}\sum_k \left(\sum_i \lambda_i^2 y_i^2 x_{ik}^2 + 2\sum_i \sum_{j \neq i} \lambda_i \lambda_j y_i y_j x_{ik} x_{jk}\right) - \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_i \lambda_i$$

$$\max_{\lambda_i} \tfrac{1}{2}\textstyle\sum_i \lambda_i^2 y_i^2 x_i^T x_i + \sum_i \sum_{j \neq i} \lambda_i \lambda_j y_i y_j x_i^T x_j - \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_i \lambda_i$$

$$\max_{\lambda_i} \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j - \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_i \lambda_i$$

$$\max_{\lambda_i} \sum_i \lambda_i - \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j$$

SVM: naming digression for nerds

If you actually take derivatives, the solution for $\hat{\beta}$ is

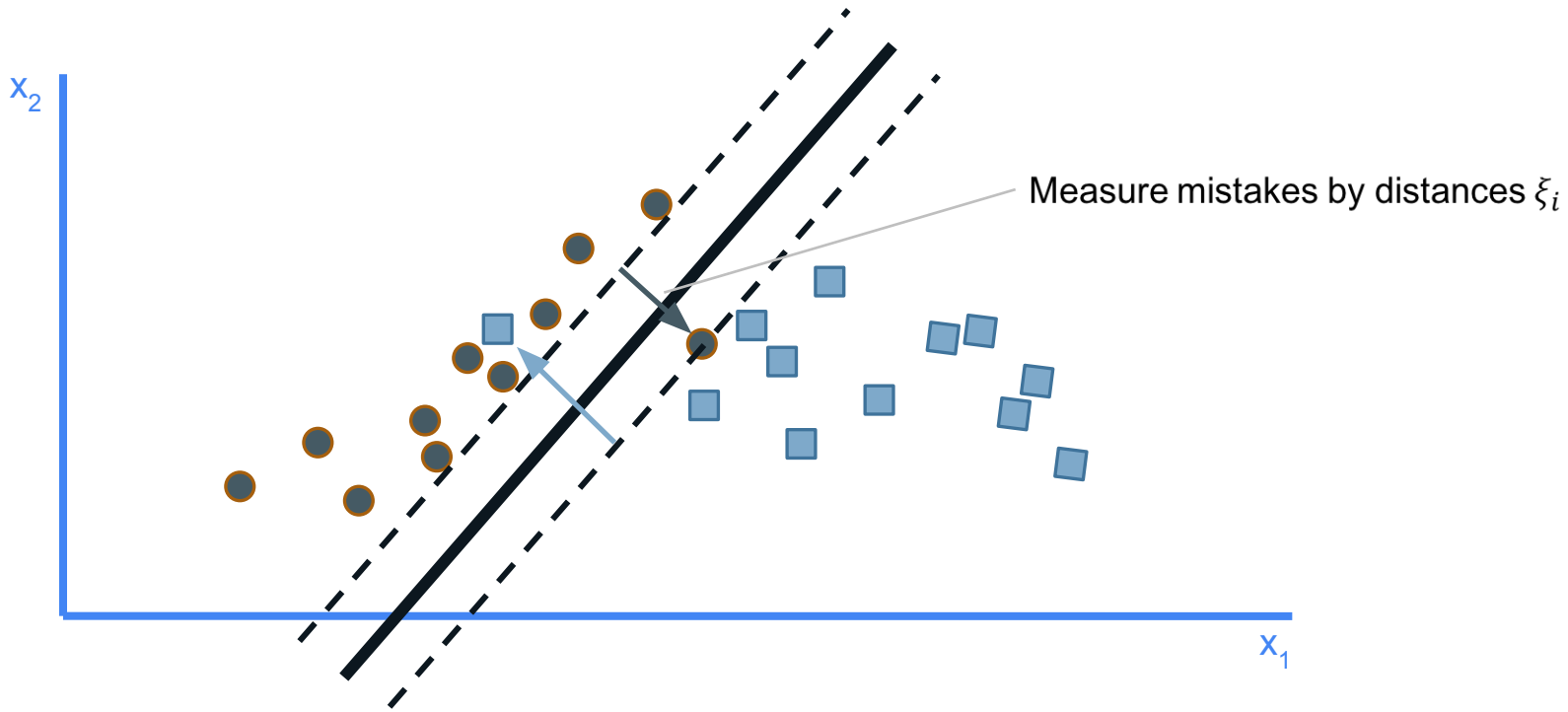
$$\hat{\beta} = \sum_i \lambda_i y_i x_i$$

Recall that for Lagrange multipliers λ_i, if the relevant constraint is non-binding, the multiplier takes the value 0.

For which points will the constraint bind exactly?

Support vector machines for messy data

Issue 2: Maybe no lines work perfectly. What can we do?



SVM: objective for messy data

With misclassification in our training data, we rewrite our problem as

$$\min_{\beta,b} \frac{1}{2}\|\beta\|^2$$
$$s.t.$$
$$y_i\left(x_i^T\beta + b\right) \geq 1 - \xi_i \; \forall i$$
$$\sum_i \xi_i \leq \gamma$$

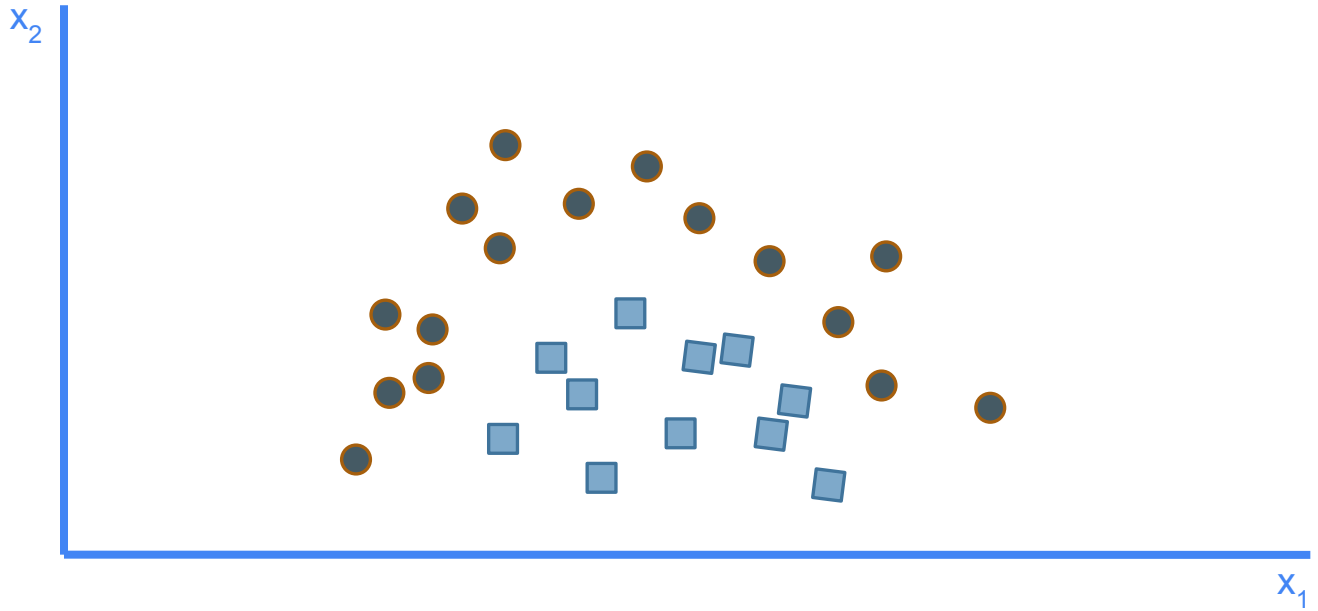
Where

- The ξ_i are (non-negative) distances from the relevant margin line to the location of the point. Points on the correct side of the margin have $\xi_i = 0$.
- γ is a constant. It puts a bound on the number and severity of misclassifications we're willing to accept.

This is just a more constrained problem with another Lagrange multiplier

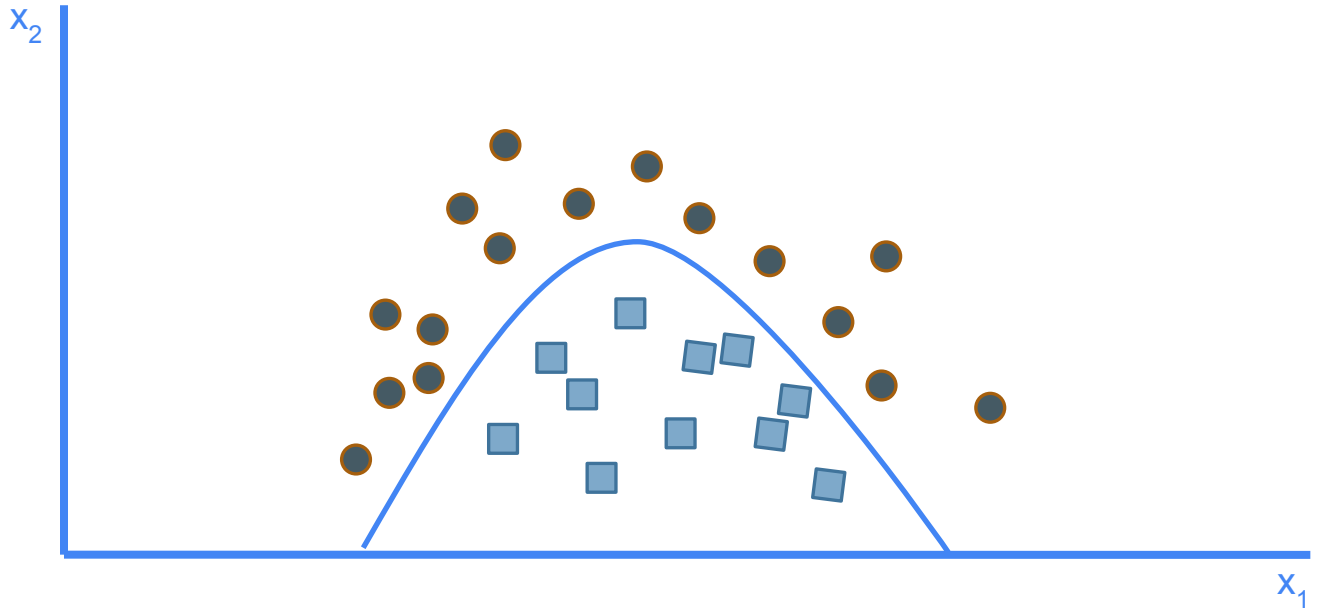
SVM: nonlinear problems

We could try to separate this with a line, and accept a lot of misclassification error. What else can we do?



SVM: nonlinear problems

What else can we do? Transform the variables we use to classify, e.g., add x_1^2, not unlike we might in a regression (LPM or GLM) context. But we can do more.



SVM: nonlinear problems, generally

For linear problems we saw building our SVM reduced to

$$\max_{\lambda_i} \sum_i \lambda_i - \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i^T x_j \text{ s.t. } \lambda_i \geq 0$$

For nonlinear problems, we saw we could expand our x variables through a function, e.g. $\Phi: (x_{i1}, x_{i2}) \rightarrow (x_{i1}, x_{i1}^2, x_{i2})$. That simply means our problem becomes

$$\max_{\lambda_i} \sum_i \lambda_i - \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j \Phi(x_i)^T \Phi(x_j) \text{ s.t. } \lambda_i \geq 0$$

We really care about $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$, and not Φ itself. Using that **Kernel function K** directly can have huge computational advantages.

$$\max_{\lambda_i} \sum_i \lambda_i - \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j K(x_i, x_j) \text{ s.t. } \lambda_i \geq 0$$

Careful choice of Kernel function can improve classification accuracy in a big way.

Choosing a kernel

In the spirit of tuning, try several.

Can start with linear (what we saw already)

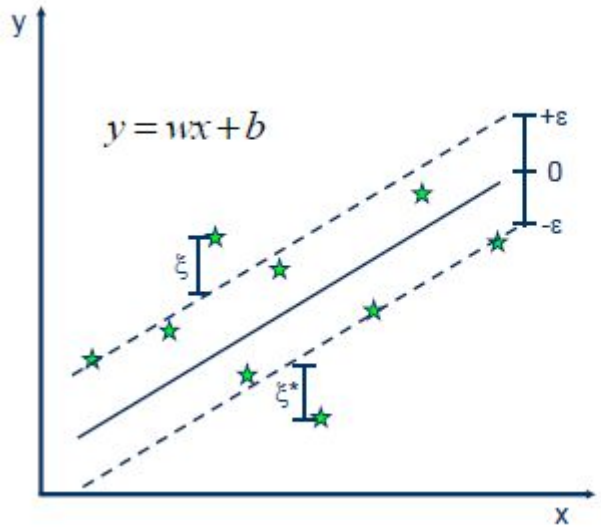
Good starting point if you suspect nonlinearity: Gaussian (radial basis) kernel

$$K(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}}$$

Support Vector Machines applied to regression

There's no margin separating groups of points in a regression task. What do we do?

Start with the formulation where we're minimizing over a norm of the coefficients, and use constraints that say our prediction errors must fall within a certain range (allowing for and penalizing "mistakes" that fall outside that band)



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon + \xi_i$$

$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \ge 0$$

SVMs in R

Packages and functions

Several choices. We'll stick with **caret** for overall infrastructure, which uses **kernlab** behind the scenes for SVMs.

Overall the workflow looks a lot like what we saw for LASSO/Ridge, but with a different specification to the *method* argument of *train*.

This supports classification and regression, as well as a range of different kernels.

→Off to RStudio again...

Tree-based approaches

What do regression/classification trees do?

Goal: use x to predict y

Approach: Repeatedly split sample based on x, make different predictions for different sub-samples. When done, each subsample is called a “leaf.”

Choose splits to minimize an objective summed across all nodes.

1. Prediction error (continuous outcomes/regression trees)

$$\sum_i (\hat{y}_i - y)^2$$

2. Within-subsample impurity (categorical outcomes/classification trees)

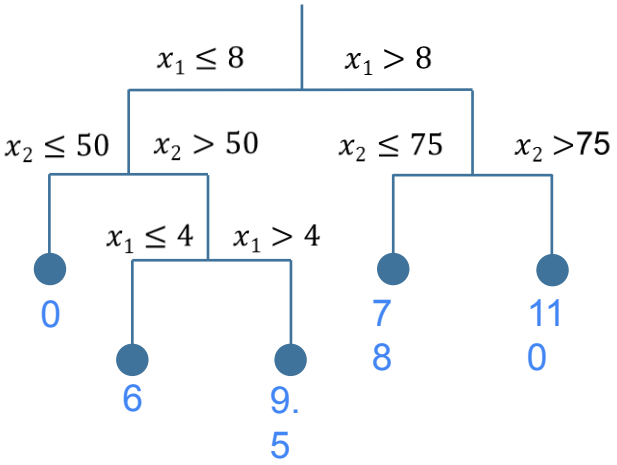
$$\sum_c p_c(1 - p_c) \text{ or } \sum_c -p_c log(p_c)$$

Overfitting is a problem here too (e.g., sub-samples with one observation). Several methods for dealing with it.

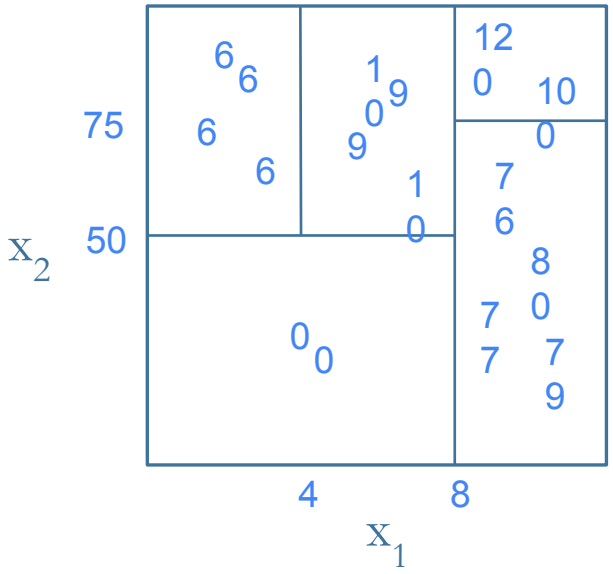
Regression trees: two views

In their simplest form, regression trees build up a piecewise constant predictor (variants are local linear, etc).

View as decision tree



View as partition



Building trees: approach

In the most basic implementation, pretty much a brute force approach:

1. Start with a tree with one leaf, and compute "loss" (prediction error or impurity)
2. For each variable and each possible split point for that variable, try an axis-aligned split (e.g., $x_1>3$), compute loss again.
3. Pick the best possible split from the previous step.
4. Repeat 1-3 for each leaf in the new tree

Some tricks exist for making step 2 fast.

When do we stop?

If we keep splitting, eventually we get a leaf per point. That's overfitting personified.

Two ways to overcome this

1. Stop splitting once the improvement in fit falls below a threshold

2. Overfit, but then prune back the tree by recombining leaves until the reduction in fit gets sufficiently large

While it might seem wasteful, #2 has advantages. Splitting is myopic; might find a useful split later.

Moving beyond a tree

Trees are very flexible and intuitive, but not without their problems:

1. Can be sensitive to input data – changing a point can give different splits, etc
2. Most functions we're trying to learn aren't really step functions, are they?
3. How do we quantify uncertainty?

All of these hint at a familiar procedure: bootstrapping.

Bootstrap aggregating (Bagging)

Apply the bootstrap procedure to tree-building:

1. Create a bootstrap training set b from the original training set
2. Grow a tree from b
3. Repeat for all b in the set B of bootstrap samples
4. To predict an outcome for a new data point x, predict for each tree, then average (For classification trees, each of the trees vote)
    - If we need class probabilities vs a prediction, can use class proportion in relevant leaf from each tree, and average those proportions

NB: Some of the tree-based causal inference methods we'll see later use subsampling rather than bootstrapping, but the basic idea is similar.

From bagging to random forests

We can go further with the randomization idea in bagging.

Idea behind both: lower variance in predictions by averaging over different trees.

- Bagging: Get different trees by randomizing sample
- Random forests: Bagging plus randomizing which variables you can split on.

Each time you go to split, only consider a randomly selected subset of variables, and choose the best split among those variables.

Trees & forests in R

Packages & functions

We'll again stick with **caret** for overall training.

We'll use the **ranger** package behind the scenes by specifying the *method* argument to the *train* function. That is for random forests (ranger: random forest generator)

If you just want a single tree, you can use the **rpart** package (rpart: recursive partitioning). The **caret** package also supports **rpart** behind the scenes.

- Random forests are usually *far* better predictors
- The primary argument for regression/classification trees is interpretability. Depends on your goals.

Network-based approaches

Neurons

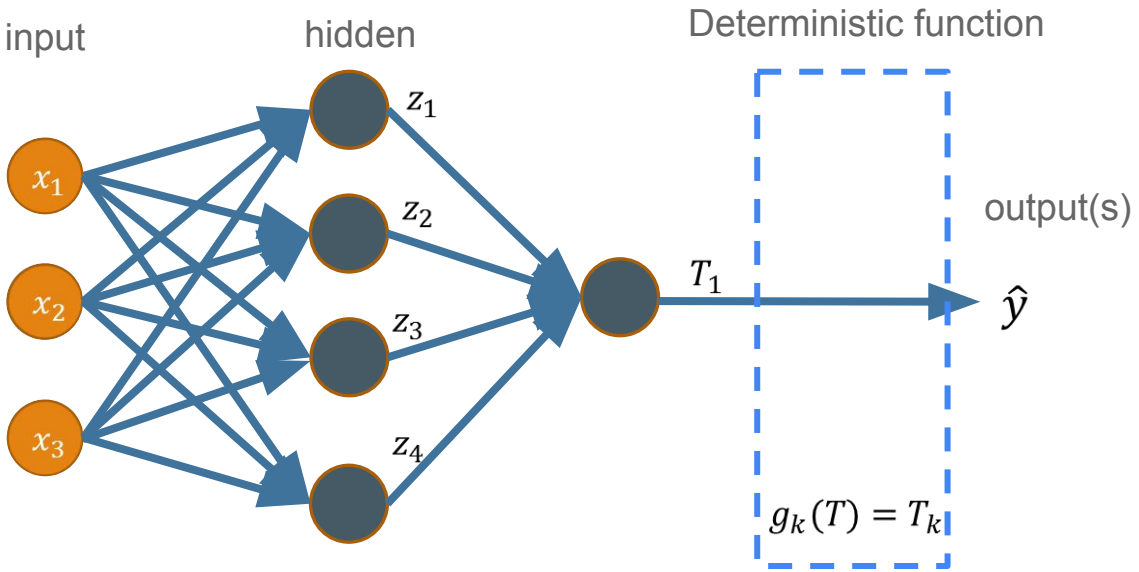
Our brain makes models problems through networks of neurons.

Each network takes a set of inputs and fires under certain conditions.
What if we mimic that process?

$$z = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

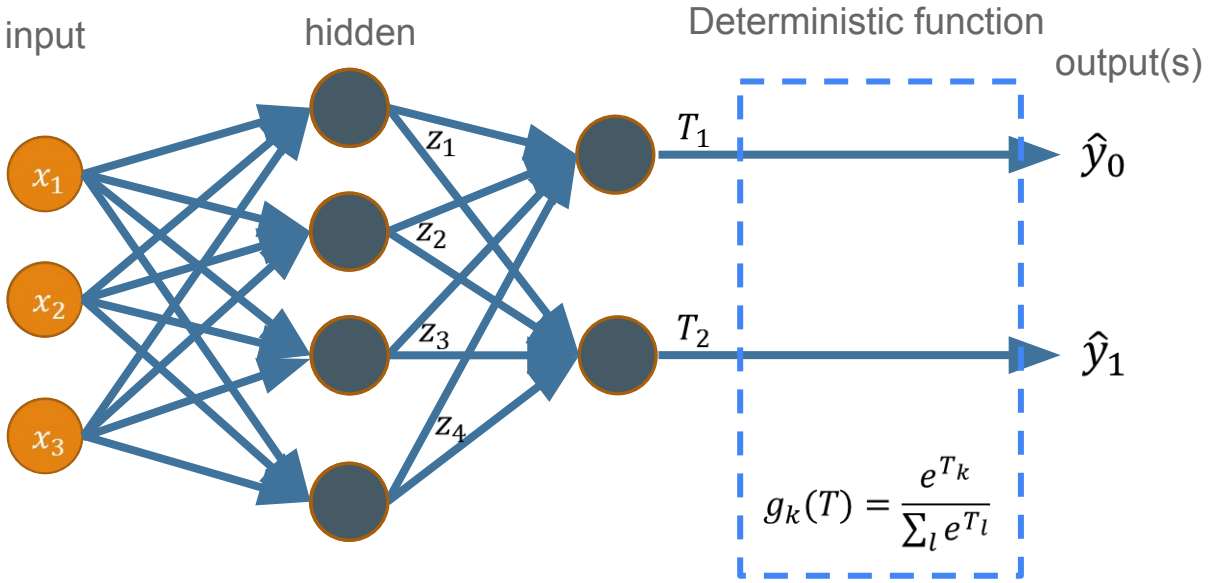
Neural networks for regression

Connect multiple layers of neurons. Often the covariates are considered an input ‘layer’



Neural networks for classification

Connect multiple layers of neurons. Often the covariates are considered an input 'layer'



Neural networks: learning

We have a bunch of neurons, each with some relationship between inputs x and output z that depend upon weights w and an offset b:

$$z=\sigma(w\cdot x+b)$$

1. What is $\sigma(\cdot)$?
   - Originally, a step function: 1 if argument positive, 0 otherwise
   - More modern approach: sigmoid shaped function (e.g. logistic)

2. How do we learn w and b for each neuron? Stochastic gradient descent.
   - Stochastic: use sample of training data (batch) rather than all data in each update step
   - Uses a trick called back-propagation to compute gradients at each neuron

Back propagation

Gradient based methods update coefficient guesses a la:

$$\beta^{r+1}=\beta^{r}-\gamma_r \nabla_\beta L$$

Where $\nabla_\beta L$ is the gradient of the loss function w.r.t. the parameter vector β, and γ_r is some constant determining step size ('learning rate')

For neural nets, it turns out we can compute that gradient as a function of 'errors' at all of the nodes:

1.Evaluate predictions at a current guess for all parameters (forward pass)

2.Compute loss at the output nodes and gradient with respect to output layer parameters.

3. Work backward through the network, applying chain rule and results from next layer forward to arrive at gradient with respect to each layer's parameters. (back propagation)

4. Use the gradient-based update rule and the gradients calculated in 2-3 to update guesses.

Neural networks: art

Throwing a bunch of nodes in layers won't necessarily work well:

- If you have lots of layers, it will take a long time to train
- Simply adding more nodes doesn't always increase predictive accuracy

1. Use problem knowledge to intelligently construct structure of network

- We'll see this tomorrow morning in the form of convolutional neural nets

2. Can use different types of neurons (e.g., not simply a nonlinear transformation of linear combination of inputs, but some other operator, e.g., max) but less common

Basic neural nets in R

Neural networks in R

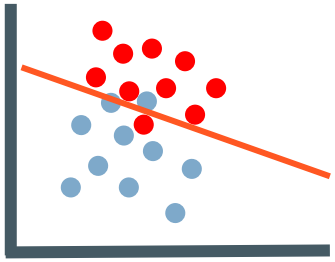
There are many packages. We'll continue to use caret with the **nnet** backend.

This is a pretty restrictive version of neural nets (single hidden layer), but it'll illustrate the point.

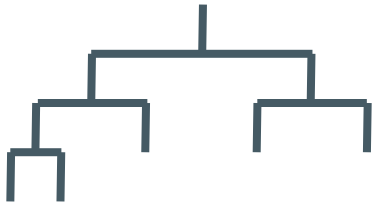
Tomorrow we'll look at some more complex networks in the service of image classification (e.g., to produce inputs to a standard econometric model).

Combining learners

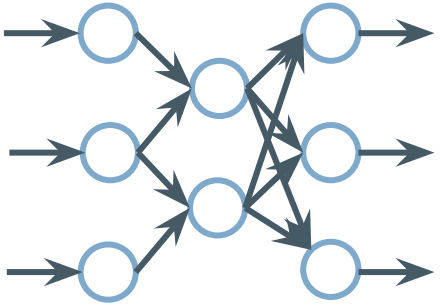
Can we do better by combining learners?



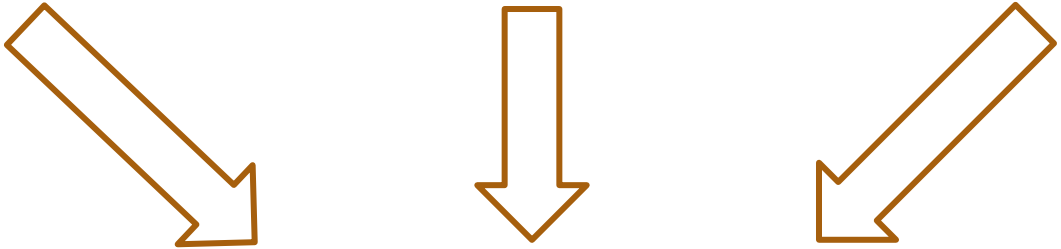
Regression-like approaches



Tree-based methods



Network-based methods



Why choose one?

Why not combine methods?

Since each ML method has pros/cons, why not combine them to improve performance, just like we do with other econ methods
(e.g., difference in discontinuities)?

- Idea #1 (“boosting”): Train a series of models iteratively to improve on areas where previous ones did poorly

- Idea #2 (“ensemble”): Train several methods separately on original data, then aggregate their predictions

Boosting

Suppose we fit a regression model using some flexible approach, yielding predictions $\hat{y}_i$. For simplicity, suppose our loss function is squared error:

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Intuition: can reduce loss by (iteratively) fitting the residuals using another model

1) fit a different model to the residuals, i.e., use $r_i = y_i - \hat{y}_i$ as outcomes that we feed to another learning model with loss $L(2) = \sum_{i=1}^{n} (r_i - \hat{r}_i)^2$

2) add our predictions from that new model to those from our first model to get overall predictions, i.e., $\tilde{y}_i = \hat{y}_i + \hat{r}_i$

3) repeat 1-2 with the residuals from the combined model, over and over

Gradient boosting

Boosting intuition is just to try to improve areas where our first model does poorly.

This turns out to be equivalent to gradient descent for the loss function. In this context, gradient descent has an update rule

$$\hat{y}_{m+1}(x_i) = \hat{y}_m(x_i) - \lambda \frac{\partial}{\partial F(x_i)} \sum_{i=1}^{n} L(y_i, F(x_i)))\Big|_{F(x_i)=\hat{y}_m(x_i)}$$

But for our loss function, the partial derivative $-2(y_i - \hat{y}(x_i))$ is just .
So the residuals tell us about the direction in which to update our predictions.
Shouldn't use residuals directly (cheating/overfitting), so we fit a new model to them (more generally we fit a model to the loss gradient w.r.t. our predictions)

We can find lambda via one-dimensional optimization.

Gradient boosting

Popular implementations make specific choices/modifications, often to reduce overfitting/improve out-of-sample performance by learning slowly (these are tuned)

- Learners at each stage are often shallow regression trees.
- Shrink (multiply by a number in (0,1)) the gradient component of the update rule. Basically move predictions toward observations, but only partly.
- Can introduce a penalty on complexity (for tree base learners, # of leaves and L1 or L2 norm on predictions across leaves, summed across learners)
- Cut off learning at some point (# rounds of updates)
- Can subsample data and/or variables used to split

Gradient boosting in R

Ensembles

Say we have multiple predictors $\hat{y}_1(x_i), \hat{y}_2(x_i), ..., \hat{y}_K(x_i)$

If their prediction errors are somewhat uncorrelated so that where one does poorly, another does well, we could potentially gain by combining them.

Ideas:

1) use a weighted average
2) use the predictions as inputs to another machine learning tool

If we allow for an intercept #1 is just using predictions as inputs to OLS

Ensembles in R

Ensembles in R

In R, this can be accomplished using the **caretEnsemble** package, which builds on the **caret** package we've seen.

- Training happens via the *caretList* function
- Can take simple weighted average (linear combination) via the *caretEnsemble* function
- Could instead train a machine learning tool using the outputs of individual predictors as inputs (via the *caretStack* function)

Brief thoughts on choosing an approach

Sparse DGP → LASSO
Nonlinear DGP → RF, boosting, SVM
Interpretability matters → regression trees, penalized regression

SVM performance may depend on kernel selection.
Tree-based methods require less up-front thought (but can overfit)

SVM can be slow to train on large datasets (but fast to predict)

Lunch

and talk about how ML can help you

← What AI thinks
“economists having lunch” look like

ML in the service of causal inference

Selection on observables
IV
RDD
Predicting counterfactuals
Heterogeneous treatment effects

Selection on observables with ML

Propensity score estimation

A simple/early/easy application of ML:

Use your favorite predictive tool to estimate a propensity score model, then use those propensity score estimates in your favorite (/least hated) propensity score approach: IPW, PSM, etc.

Mixed results on efficacy. See, e.g.(Goller et al., 2020)

- ML methods target prediction of PS, but may omit important predictors of outcome and so result in inadequate balance on outcome-relevant covariates.
- Practical performance depends on method used; they find LASSO works well, but random forests don't when share of treated small.

Estimation after covariate selection (using, e.g., LASSO)

Belloni & Chernozhukov (2013)
Belloni, Chernozhukov, & Hansen (2014)

Starting point: LASSO does model selection, but biased

LASSO does variable selection. Are we all out of a job?

Start with some general model with p predictors

$$y = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

Suppose we let LASSO select s out of p possible predictors.

We then get a fitted model

$$\hat{y} = \hat{\beta}_0 + \sum_{j:\hat{\beta}_j != 0} \hat{\beta}_j x_j$$

But because of the penalty term, LASSO shrinks estimates and

$$E[\hat{\beta}_j | x] != \beta_j$$

Idea: reduce bias by using OLS after

OLS post-LASSO:

1. Use LASSO to estimate the main model, letting it do variable selection
2. Estimate the main model with selected variables from step 1. This time use OLS.

When might this be good?

- If a limited number of parameters are actually non-zero (“sparsity”)
- If LASSO selects exactly those parameters in step 1 as the non-zero estimates.

Problem: omitted variable bias

Say first-stage LASSO falsely sets a parameter to zero.

→ We omit it from the second stage OLS estimation

When will this be a (big) problem? From standard omitted variable bias intuition, it depends upon:

1. The magnitude of the omitted coefficient
2. The strength of correlation between the omitted and included variables

What do we do? LASSO again!

OLS post double LASSO

Procedure (See the **`hdm`** package in R, and in particular the **`rlassoEffect`** function)

1. Estimate LASSO with y as outcome and controls (NO treatment) as regressors.
2. Estimate LASSO with treatment variable as outcome and controls as regressors
3. Take union of variables selected via 1 and 2 (plus others researcher thinks is important)
4. Estimate OLS with y as outcome and treatment + variables from step 3 as regressors.

Proposed benefits:

- Steps 1-3 likely to get closer to correct set of parameters than single LASSO.
- OLS in step 4 has same benefits of post-LASSO (limit bias from LASSO itself)
- Allows inclusion of researcher-specified variables but less constrained by researcher
- Cool name

Double machine learning

Chernozhukov et al. (2018)

Double machine learning

Double lasso:

- Use ML twice: for selection and outcome equations.
- Goal: identify variables that may belong in the model.

More general idea:

- We have nuisance functions in both outcome and selection equations – view that problem as a specification rather than variable selection problem

Double Machine Learning: setup

Consider a partially linear model:

$$y_i = \beta T_i + f(x_i) + \varepsilon_i$$

$$T_i = g(x_i) + \nu_i$$

Where β is the coefficient of interest associated with a treatment variable T_i.

We don't really care about $f(\cdot)$ or $g(\cdot)$ as long as they give us ignorability.

We can't simply use ML to estimate $f(\cdot)$ directly using y_i and x_i since ML tools are generally biased (which helps with prediction RMSE)

Double ML continued

Double ML also estimates $g(\cdot)$, then uses residuals from both steps.
Intuition similar to Frisch Waugh Lovell, though mechanics different:

$$\hat{\beta} = \left(\frac{1}{n} \sum_{i=1}^{n} (T_i - \hat{g}(x_i)) T_i \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} (T_i - \hat{g}(x_i))(y_i - \hat{f}(x_i)) \right)$$

This helps deal with the regularization bias from estimation of $f(\cdot)$.
Note this is basically using the treatment residuals as an instrument for treatment.

There's one final tweak...

$\hat{\beta}$

Double ML... still continued

We shouldn't use the full dataset to estimate $f(\cdot)$, $g(\cdot)$, and our parameter of interest. In short, our estimates of those nuisance functions will be correlated with structural unobserved components of T_i because they're from the same data, which can introduce bias in beta hat. This correlation will be higher the more our estimators of $f(\cdot)$, $g(\cdot)$ overfit.

$$\hat{\beta} = \left(\frac{1}{n} \sum_{i=1}^{n} (T_i - \hat{g}(x_i)) T_i \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} (T_i - \hat{g}(x_i))(y_i - \hat{f}(x_i)) \right)$$

Solution: cross-fitting. Estimate $f(\cdot)$, $g(\cdot)$ on part of sample, estimate beta on the other. Swap subsamples and repeat, then average (or take median) of estimates. Can do this 2 times, 5 times, 10 times...

Inference in double ML

The estimators we just saw are approximately asymptotically normal. For example, if score functions are affine in the parameter of interest θ:

$$\hat{\sigma}^2 = \hat{J}_0^{-2} \frac{1}{N} \sum_{k=1}^{K} \sum_{i \in I_k} \left[\psi(W_i; \tilde{\theta}_0, \hat{\eta}_{0,k}) \right]^2,$$

$$\hat{J}_0 = \frac{1}{N} \sum_{k=1}^{K} \sum_{i \in I_k} \psi_a(W_i; \hat{\eta}_{0,k}),$$

- The sums average across folds and observations within a fold
- ψ is the score function, ψ_a is the coefficient on θ in that score function, and η is the set of nuisance parameters

This has some intuitive appeal: we use variation in the scores at our estimate scaled based on the coefficient portion of the score function

Inference in double ML, continued:

For example, the double ML score function for the partially linear model

$$y_i = \beta T_i + f(x_i) + \varepsilon_i$$

$$T_i = g(x_i) + \nu_i$$

Is $\psi(W_i; \theta, \eta)=(y_i - \beta T_i - f(x_i))(T_i - g(x_i))$

This can refactored as

$\psi(W_i; \theta, \eta)=[-T_i (T_i - g(x_i))]\beta + (y_{ii} - f(x_i))(T_i - g(x_i))$

$\psi(W_i; \theta, \eta)=\psi_a(W_i; \theta, \eta)\beta + \psi_b(W_i; \theta, \eta)$

In short, $\psi_a(W_i; \theta, \eta)$ contributes to variability in $\psi(W_i; \theta, \eta)$ but is about T_i and our modeling of it via $g(x_i)$, not about β

Extension: Double ML for DiD (Chang 2020)

These ideas can be extended to a two-period DiD setting – we just need the right orthogonality conditions.

$$\psi_1(W, \theta_0, p_0, \eta_{10}) = \frac{Y(1) - Y(0)}{P(D=1)} \frac{D - P(D=1 \mid X)}{1 - P(D=1 \mid X)} - \theta_0$$
$$- \underbrace{\frac{D - P(D=1 \mid X)}{P(D=1)(1 - P(D=1 \mid X))} E[Y(1) - Y(0) \mid X, D=0]}_{c_1}$$

Here the c_1 term is the adjustment to ensure Neyman orthogonality.

Chiang (2021) also extends to two-way clustering. Adjust cross-fitting to ensure we don't use observations from either of the clusters to which *i* belongs in other sample

As of now, this is only implemented in Python library (but there are options…)

Double Machine Learning in R

The **DoubleML** package implements these approaches. Let's go take a look.

IV with ML

How should we combine instruments?

Suppose you have multiple valid instruments (lucky you). How should you use them?

Textbook answer: use all of them (e.g. via 2SLS) for efficiency reasons.

- Newey & Powell (2003) added flexibility via basis expansions

Problem is that 2SLS bias increases with # of instruments and their weakness.
See, e.g., Hahn & Hausman (2002)

-

LASSO, revisited in service of IV

One idea is to use ML to optimally combine instruments

Belloni et al. (2012) do this with LASSO. Sparsity reduces # of instruments, and should throw out the weak ones first.

Steps (modified 2SLS)

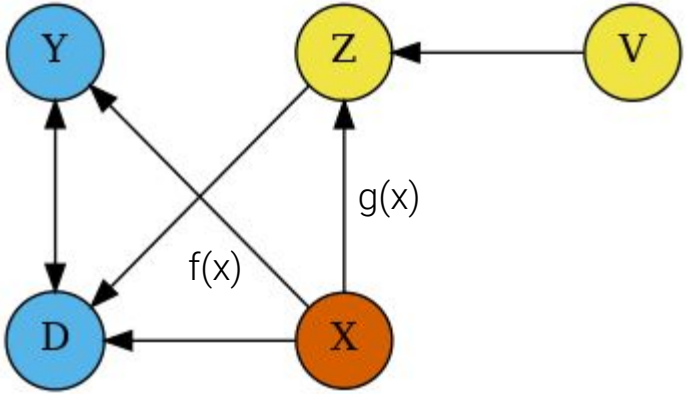
1. Estimate first stage using LASSO. If desired, refit on OLS afterward (post-LASSO first stage)
2. Estimate second stage using predictions from first stage

This achieves the semiparametric efficiency bound asymptotically, and is asymptotically normal.

See the **hdm** package again (**rlassoIVselectZ** function) for this and extensions

Double ML, revisited

Chernozhukov et al. (2018) results also apply to a partially linear IV setting where there may be unknown relationships between x and the a) outcome, and b) instrument



The idea is the same:

- Split into K folds
- For each fold, estimate g and f on all other folds, then use them to estimate beta on this fold
- How do we estimate beta?
  IV form again:

$$\hat{\beta} = \left(\frac{1}{n} \sum_{i=1}^{n} (Z_i - \hat{g}(x_i)) T_i \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} (Z_i - \hat{g}(x_i))(y_i - \hat{f}(x_i)) \right)$$

RDD with ML

Just kidding... we're not there yet

"Finally, another avenue for future research is related to incorporating modern high-dimensional and machine learning methods in RD settings.

Such methodological enhancements can aid in principled discovery of heterogeneous treatment effects as well as in developing more robust estimation and inference methods in multidimensional RD designs." (Cattaneo & Titiunik, 2022)

Predicting counterfactuals with ML

Idea #1: Just predict counterfactuals, subtract

For treatment effect settings, we're often interested in a difference between conditional mean outcomes: E[y(1)|x] - E[y(0)|x]

We could simply predict each using ML and subtract.

Issue: ML tools are typically biased predictors.

Key identifying assumption: bias is the same for both E[y(1)|x] and E[y(0)|x]

Idea #1 example (Abrell et al., 2022)

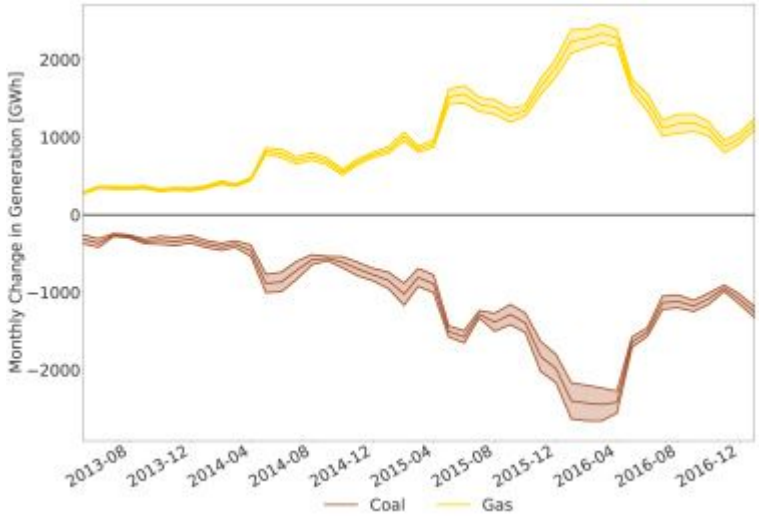
Question: how did carbon pricing scheme in UK affect generation, emissions?

Fit LASSO models in pre-policy period, use to predict both pre- and post-policy

Assumption: treatment unrelated to prediction bias

Find net effects on generation near zero but reduction in emissions

Get SEs by bootstrap (separately per year)



Idea #2: Souped up synthetic ("Artificial") controls

Synthetic controls are a data-driven way to estimate counterfactuals using a pool of donor control observations.

Carvalho et al. (2018): Why restrict ourselves to simple weighted averages?

- Can use nonlinear terms to do a basis expansion
- Then Use LASSO

Derive a number of properties of this approach, including inference
(as an alternative to randomization inference common in the synth control lit)

The **ArCo** package in R implements this approach

Idea #3: Matrix completion (Athey et al., 2021)

In a panel setting, arrange potential outcomes into two matrices, with rows representing cross-sectional units and columns representing time.

Causal inference can be framed as trying to fill in missing matrix entries.

$$\mathbf{Y}(0) = \begin{pmatrix} ? & ? & \checkmark & \dots & ? \\ \checkmark & \checkmark & ? & \dots & \checkmark \\ ? & \checkmark & ? & \dots & \checkmark \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ ? & \checkmark & ? & \dots & \checkmark \end{pmatrix} \qquad \mathbf{Y}(1) = \begin{pmatrix} \checkmark & \checkmark & ? & \dots & \checkmark \\ ? & ? & \checkmark & \dots & ? \\ \checkmark & ? & \checkmark & \dots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \checkmark & ? & \checkmark & \dots & ? \end{pmatrix}$$

Most of the paper focuses on ATT and uses observed entries in Y(0) (potentially with auxiliary info on covariates) to estimate missing entries.

Matrix completion: Assumptions

Assume **Y(0)** is the sum of a low-rank matrix $\mathbf{L}^*$ and an error matrix **ε** with $E[\boldsymbol{\varepsilon}|\mathbf{L}^*]=\mathbf{0}$.

Elements of **ε** are independent of one another (can be relaxed) and sub-gaussian (skinny-tailed).

Why might this make sense?

- If $\mathbf{L}^*$ isn't low rank, we don't have much hope. With NT entries, have to hope there are actually far fewer degrees of freedom
- More familiar assumptions that we can extrapolate across periods or across treatment status are doing something analogous

Matrix competition: objective

Take a penalization approach to pick an estimate $\widehat{\mathbf{L}}$ that minimizes the weighted (λ; chosen via cross-validation) sum of:

- The squared differences for observed entries
- The nuclear norm of L (sum of singular values; a convex relaxation of rank)

Athey et al (2021) pull out unit (Γ) and time (Δ) fixed effects from penalization, so that the final problem is

$$(\widehat{\mathbf{L}}, \widehat{\Gamma}, \widehat{\Delta})$$

$$= \arg\min_{\mathbf{L},\Gamma,\Delta} \left\{ \frac{1}{|\mathcal{O}|} \left\| \mathbf{P}_{\mathcal{O}}(\mathbf{Y} - \mathbf{L} - \Gamma \mathbf{1}_T^\top - \mathbf{1}_N \Delta^\top) \right\|_F^2 + \lambda \left\| \mathbf{L} \right\|_* \right\}$$

Matrix completion: inference

There are no asymptotics. However, the authors suggest a resampling approach:

- Pick a random subset of actually observed entries, treat those as the only ones that are observed. If focused on ATT, these are control observations.
- Use the matrix completion method using that hypothetical set of observed entries, assigning some of them to be “treated” as placebos.
- Repeat, giving a distribution of predictions for each entry. Can use those to construct distributions of ATT, etc.

Matrix completion in R

The **MCPanel** package (requires **devtools** in R as it's not an 'official' package yet) implements these methods. It's available on Susan Athey's github.

Heterogeneous treatment effects with ML

Can't we already study heterogeneity?

Yes...	But
Estimate models on subsamples	Which subsamples should we choose?
Interaction terms in models	Which interaction terms do we include?
Regression Trees/Random Forests	Predictive tools are not causal

General challenge: model some unknown treatment effect function $\tau(x)$

What should we choose for $\tau(x)$?

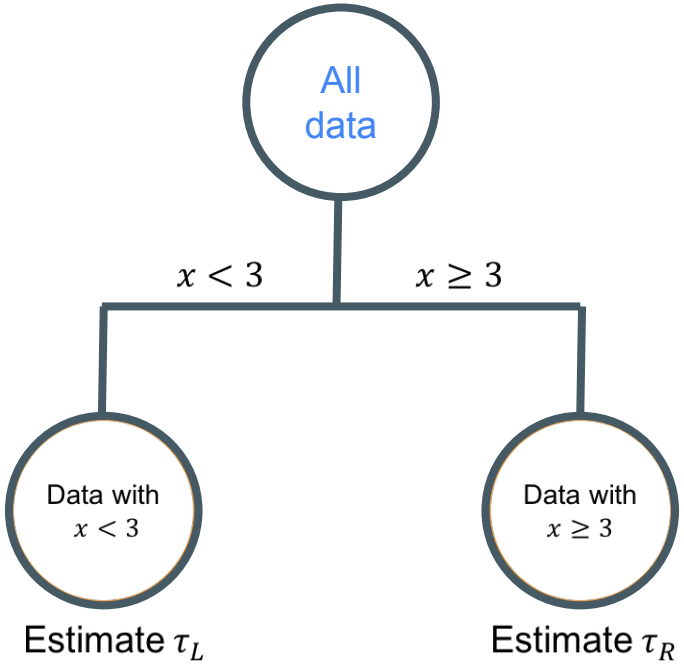
Conventional wisdom:

- Use theory
- Try a few different specifications to test different hypotheses
- Maybe use some specification tests

Why not let machine learning approximate the treatment effect function?

- Athey and Imbens (2016)
  - Can estimate $\tau(x)$ within each leaf of a regression tree using any number of methods
  - Innovation: modify splitting rule to improve treatment effect identification instead of outcome prediction
- Wager & Athey (2017)
  - Extend idea above to random forests. In principle, leaves small enough that conditional ignorability should hold. Also derive pointwise CIs
- Athey, Tibshirani, & Wager (2019)
  - Generalize the idea of using random forests to study heterogeneity to more types of estimation, including IV

Athey & Imbens (2016) intuition



Core estimation in each branch is still using econometric methods, assumptions.

Tree just helps us find subsamples where the treatment effect differs across subsamples and a constant estimate within subsamples is ok.

When is this a legitimate thing to do?

If we have unbiased estimators of τ_L and τ_R

Could be experimental data, or could use an unbiased estimator under some other assumption (e.g., conditional ignorability → IPTW)

Causal Tree method (Athey & Imbens 2016)

1. Split data into training & estimation datasets. Build trees with training data
2. Want to minimize Expected MSE of $\hat{\tau}$ but don't observe τ
3. Estimate EMSE using only an unbiased $\hat{\tau}$

$$-\widehat{\mathrm{EMSE}}_\tau\left(\mathcal{S}^{\mathrm{tr}}, N^{\mathrm{est}}, \Pi\right) \equiv \frac{1}{N^{\mathrm{tr}}} \sum_{i \in \mathcal{S}^{\mathrm{tr}}} \hat{\tau}^2\left(X_i; \mathcal{S}^{\mathrm{tr}}, \Pi\right)$$

Favors heterogeneity in estimated treatment effect across leaves

$$-\left(\frac{1}{N^{\mathrm{tr}}} + \frac{1}{N^{\mathrm{est}}}\right) \cdot \sum_{\ell \in \Pi} \left(\frac{S^2_{\mathcal{S}^{\mathrm{tr}}_{\mathrm{treat}}}(\ell)}{p} + \frac{S^2_{\mathcal{S}^{\mathrm{tr}}_{\mathrm{control}}}(\ell)}{1-p} \right)$$

Penalizes within-leaf variance in outcomes (& thus estimator)

Extension to causal forests

Causal trees are piecewise constant approximations of the treatment effect function.

- That may be a bad approximation.
- Could be high variance if, e.g., outliers can drive very different splits

Using random forest techniques (sampling + averaging) can help with both.

Identification rests on trees being deep enough that leaves contain observations with very similar covariate values, so that conditional ignorability is plausible within leaves

For each tree, use some of data to learn splits, then held out data to compute estimates to reduce overfitting (“honesty”)

SEs can be approximated by infinitesimal jackknife, which includes sensitivity of individual tree predictions for an observation to the use of that observation in each tree

Even more general: gradient forests (Athey, Tibshirani, & Wager 2019)

Regression trees and causal trees develop local estimates of a parameter by partitioning the covariate space.

One view is that trees assign weights of 1 to observations in the same leaf and 0 to other observations, then compute a weighted estimator.

Forests do this too, averaging weights over trees:

Tree b's weight for obs *i* for x

Is obs *i* in same leaf as x in tree *b*?

$$\alpha_{bi}(x) = \frac{\mathbf{1}(\{X_i \in L_b(x)\})}{|L_b(x)|},$$

obs in leaf containing x in tree b

Average of those weights across trees

$$\alpha_i(x) = \frac{1}{B}\sum_{b=1}^{B} \alpha_{bi}(x).$$

GRFs, continued

With those weights, can solve local estimating equations to estimate parameter of interest θ and nuisance parameter ν. Here ψ is a function to be minimized while O_i is the observed outcome-relevant information (e.g., y_i).

$$(\hat{\theta}(x), \hat{\nu}(x)) \in \underset{\theta,\nu}{\operatorname{argmin}} \left\{ \left\| \sum_{i=1}^{n} \alpha_i(x) \psi_{\theta,\nu}(O_i) \right\|_2 \right\}.$$

That's it. The huge contributions in ATW (2019) are:

- To define a forest implementation in a general way under this framework.
- To establish consistency and normality of the resulting estimator
- To illustrate applications of forests in this general framework

Splitting in generalized random forests

If we're after local satisfaction of estimating equations and are considering splitting the sample in a current leaf of our tree, ATW show that the expected squared error in the parameter estimates is improved if we increase heterogeneity in estimates across nodes:

$$\Delta(C_1, C_2) := n_{C_1} n_{C_2} / n_P^2 \left(\hat{\theta}_{C_1}(\mathcal{J}) - \hat{\theta}_{C_2}(\mathcal{J})\right)^2$$

Prediction on left

Prediction on right

That's expensive to compute for every possible split...

Approximations and gradients to the rescue

Approximate $\hat{\theta}_C$ based on the estimate in the parent node $\hat{\theta}_P$ and an gradient-based update rule (ξ is just a selector of θ)

Inverse of estimated gradient of expectation of ψ

$$\tilde{\theta}_C = \hat{\theta}_P - \frac{1}{|\{i : X_i \in C\}|} \sum_{\{i:X_i \in C\}} \xi^\top A_P^{-1} \psi_{\hat{\theta}_P, \hat{\nu}_P}(O_i)$$

$\xi^\top A_P^{-1} \psi_{\hat{\theta}_P, \hat{\nu}_p}(O_i)$ is just the influence function for observation i, so to approximate the estimate in a child node for a candidate split, we're just adjusting the parent estimate by the average of the influence function for observations that would end up in that child.

GRFs – the final trick

Because we're trying to maximize heterogeneity in $\hat{\theta}_C$ and we approximate that by

$$\tilde{\theta}_C = \hat{\theta}_P - \frac{1}{|\{i : X_i \in C\}|} \sum_{\{i:X_i \in C\}} \xi^\top A_P^{-1} \psi_{\hat{\theta}_P, \hat{\nu}_P}(O_i)$$

We can really choose splits to maximize heterogeneity in $\frac{1}{|\{i : X_i \in C\}|} \sum_{\{i:X_i \in C\}} \xi^\top A_P^{-1} \psi_{\hat{\theta}_P, \hat{\nu}_P}(O_i)$

which is a sample average. Regression trees already do that. So we compute pseudo-outcomes $-\xi^\top A_P^{-1} \psi_{\hat{\theta}_P, \hat{\nu}_P}(O_i)$ for each observation, then leverage standard splitting software

GRFs applied to IV

Consider a model $Y_i = \mu(X_i) + \tau(X_i)W_i + \varepsilon_i$, where $\mu(X_i)$ is a nuisance function, $\tau(X_i)$ is the CATE of interest, W_i is treatment, and we have an instrument Z_i. Our estimating equations are

$$\mathbb{E}[Z_i(Y_i - W_i\tau(x) - \mu(x)) \mid X_i = x] = 0$$

$$\mathbb{E}[Y_i - W_i\tau(x) - \mu(x) \mid X_i = x] = 0$$

The pseudo-outcomes here are just

$$\rho_i = (Z_i - \overline{Z}_P)((Y_i - \overline{Y}_P) - (W_i - \overline{W}_P)\hat{\tau}_P)$$

i.e., the contribution of observation i to the first moment condition in the parent

Implementation for forest methods

Check out the **grf** package (Generalized Random Forests)

But there is built-in support for cases from the paper:

- causal_forest — binary treatment, conditional ignorability
- Instrumental_forest — IV example we just saw
- lm_forest — local linear model

Note these do local centering (/orthogonalization) of variables first behind the scenes. This can help if data aren't dense in parts of the parameter space so that the leaves aren't sufficiently "small" in covariate space.

NB: forests aren't the only approach to heterogeneity

Double ML approaches also allow for this too, via an "interactive regression model" and "interactive IV model"

DAY 2

Other uses & Applications

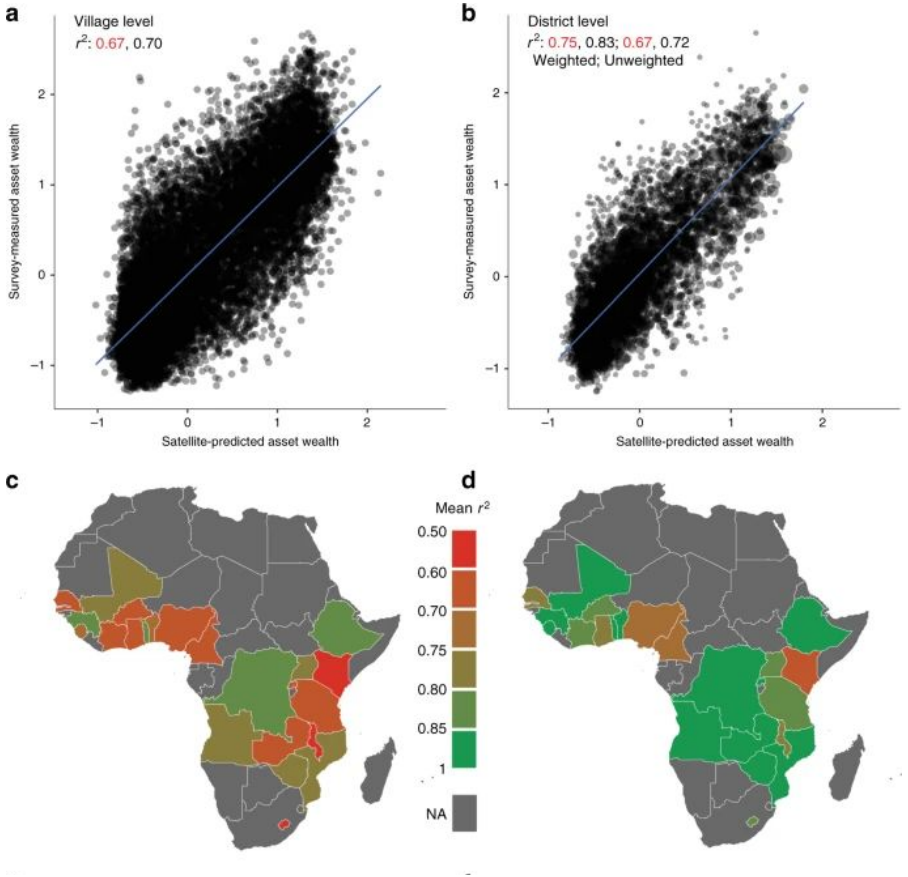
Convolutional neural networks

Goal: classify (or score) images to use in some other model

Example: want more (or more frequent) indicators of poverty (Yeh et al., 2020)

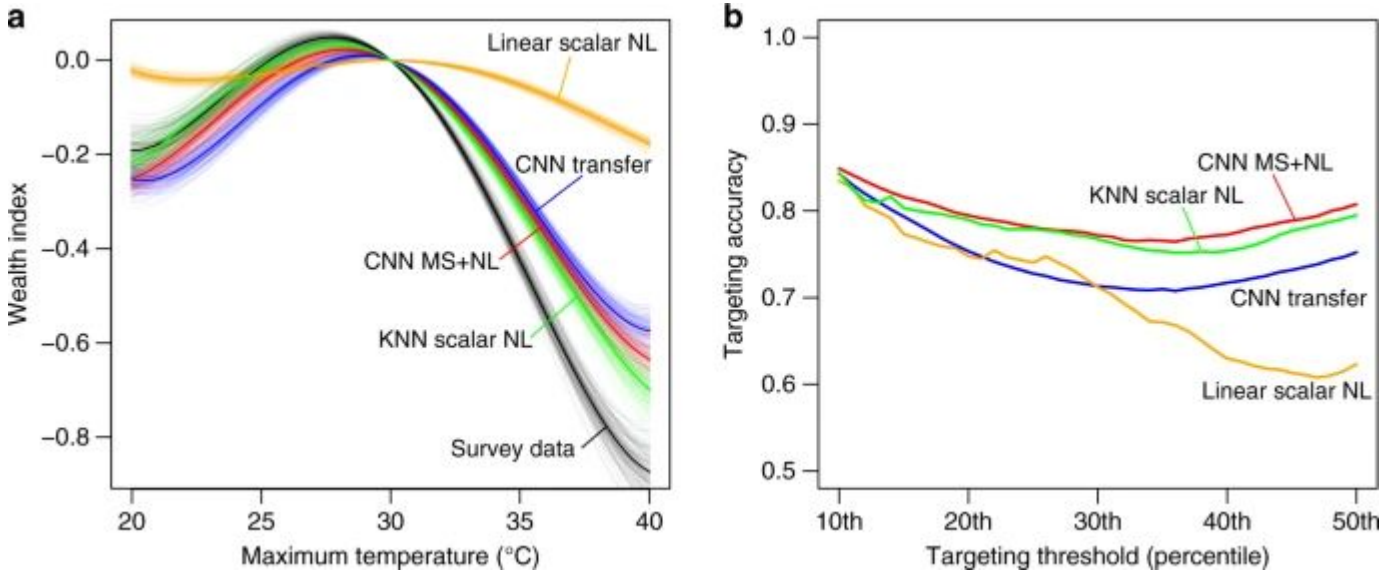
Idea: use daytime and nighttime lights satellite imagery to predict indices of wealth from household survey data.

Can recover *most* of variation in wealth.



Why does this matter? (Yeh et al., 2020 continued)

Good news: swapping in machine-learned indices of wealth for survey data, get answers that approximate what we'd get with survey data.
Left: wealth-environment relationship; right: means-based program targeting



How might we go about image classification or scoring?

Lots of ways, but state-of-the-art these days is deep learning.

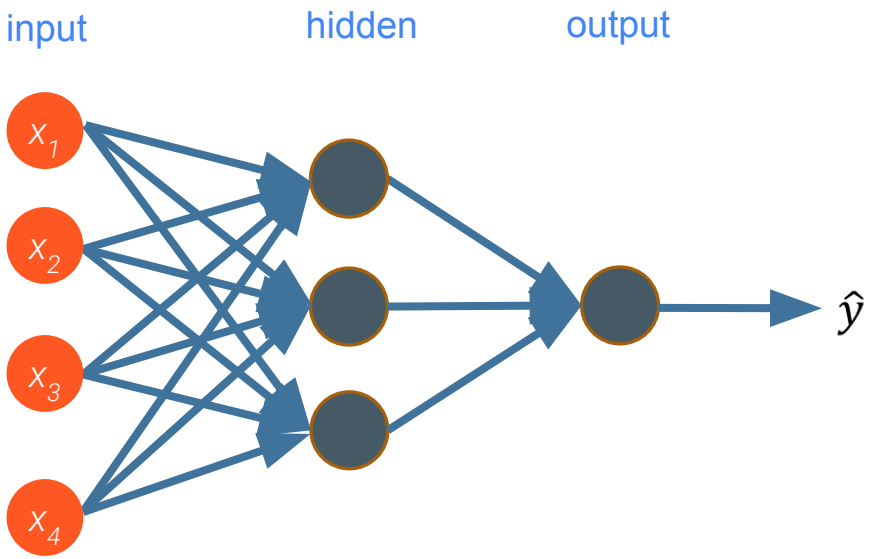
Deep learning is:

a) Another buzzword
b) Another word for specific neural networks with many layers (“deep”)
c) Part art, part science

Specifically, we’ll do a brief overview of convolutional neural networks

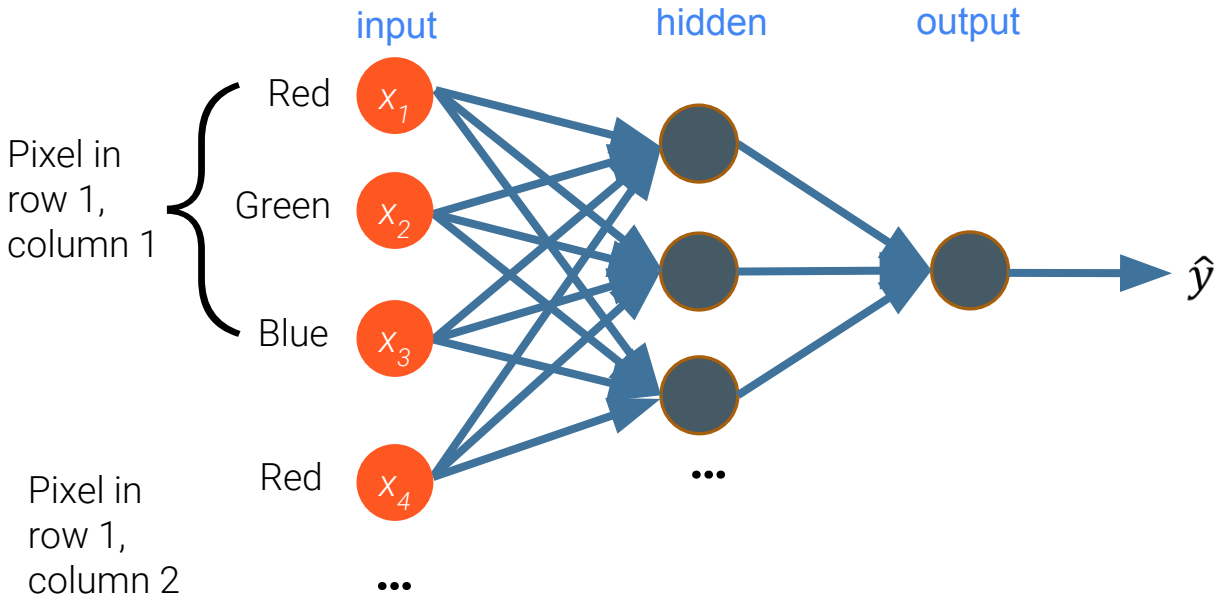
Neural networks, recap

Arbitrary connections between inputs and hidden layer. Learn weights w and offsets b for each connection



Neural networks, with image data as inputs

Each input node in a neural network might be one band of one pixel (e.g., how much red is in that pixel, either 0-255 or 0-1). Satellite applications may also use other non-visible bands.



Why might this be problematic?

Take the application we started with, where we want to predict a wealth index at a village level using a satellite image containing that village.

Say we use the red, green, and blue bands from that image, which is 100x100 pixels. That's 30,000 inputs.

Suppose we have one hidden layer with 10 neurons.

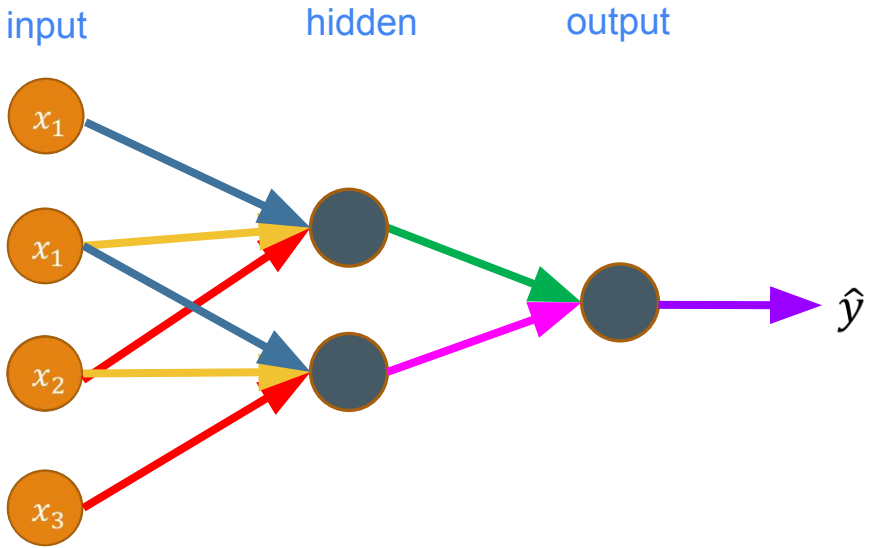
If we use a fully connected network, we would have 300,000 weights and 10 offsets to learn for that (fairly small) hidden layer.

If we want a 'deep' network, this will explode quickly.

As with most models, surely we can make some assumptions to add structure and improve our chances of learning something (if our assumptions are reasonable).

One view of CNNs: Imposing structure on network weights/parameters

In diagram below, identical arrow colors would be restricted to have same weights, offsets. Why might we do that? In the poverty example, we likely don't care if there are nighlights to the east of our town or to the west. Just that they're *somewhere*.



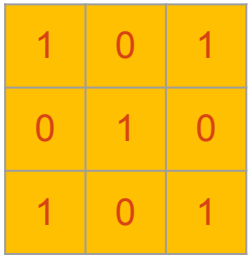
More intuitive view of a convolution layer

We have a 3x3 “filter” or “kernel” that we slide over our input image.

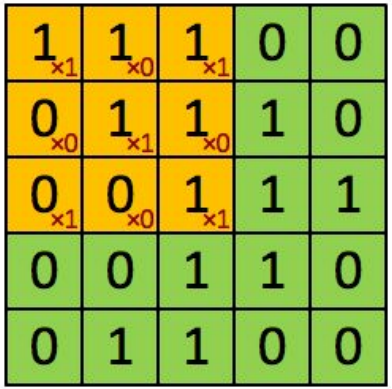
In each position, we multiply the pixel values with the values in the corresponding spot in the kernel, then add them up.

This is just **w·x** in the formulation of neural nets
Where the kernel entries are the weights **w**
and the pixel values are **x.**

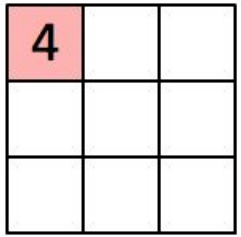
But we’re using the same **w** in many locations
(parts of the network)



Kernel/Filter

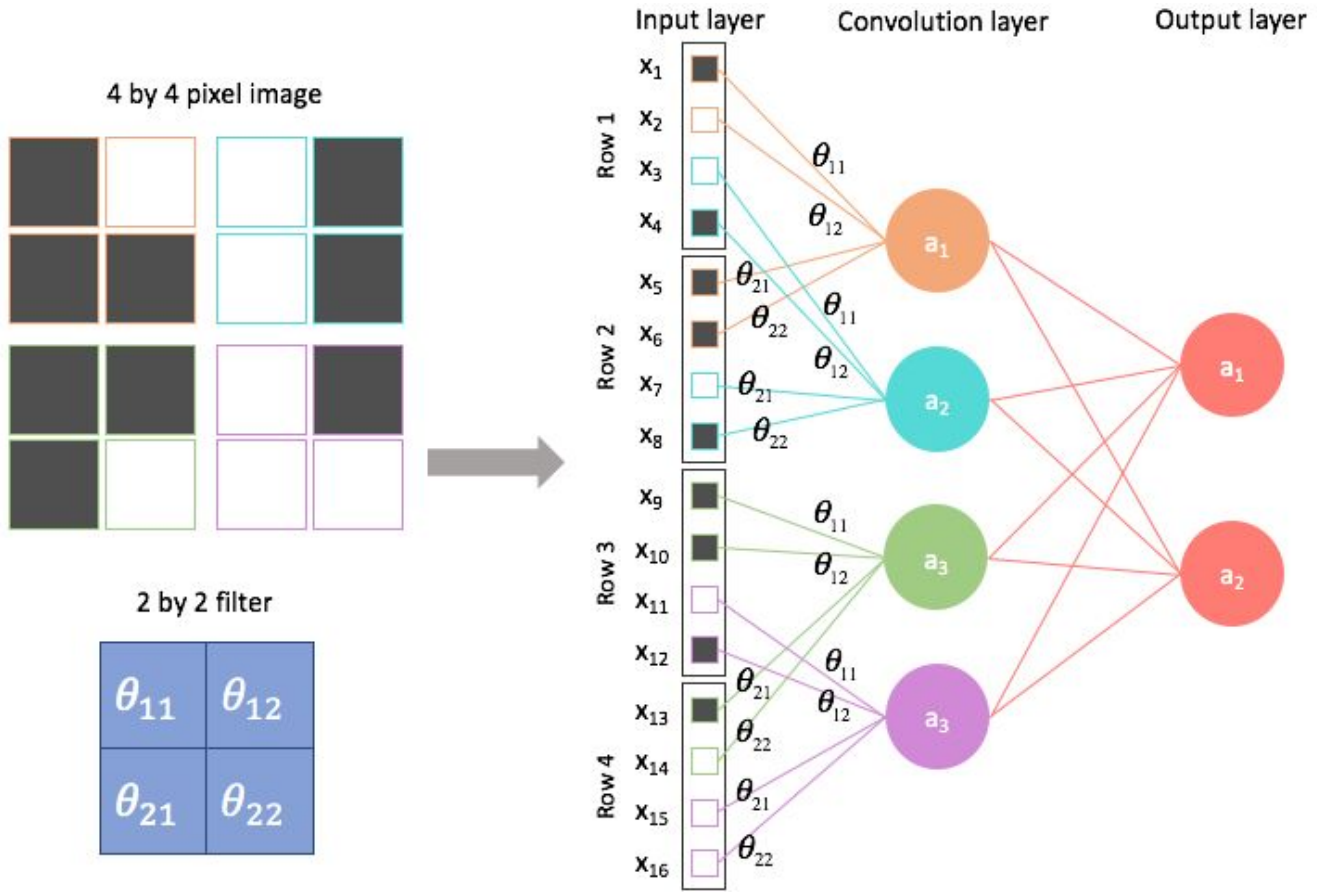


Image



Convolved Feature

More exact link between the two views



Even better intuition for convolution

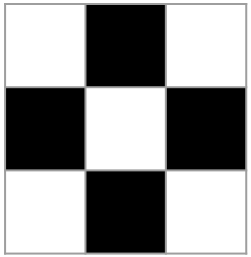
Remember a dot product like **w**·**x** can be viewed as a similarity measure.

For a given value of **w** (fixed kernel entries), we're basically looking for parts of the image that are similar to our kernel.

For example, suppose 1 encodes white and 0 is black. Our example kernel

1	0	1
0	1	0
1	0	1

is just

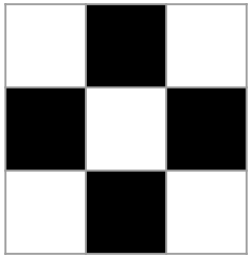


So the output of our convolution operation essentially rates how similar different locations in the image are to that 3x3 feature

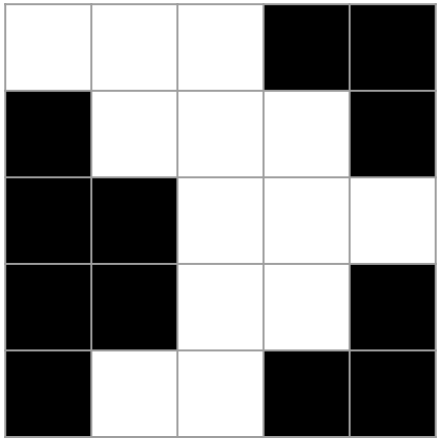
Even better intuition for convolution

Revisiting the animation, we can interpret the output as saying the lower left part of the image resembles the filter less, while, say, the upper right resembles the filter more.

Kernel/Filter



Image



Output

4	3	4
2	4	3
2	3	4

How do we pick filters/kernels?

We do have to pick their size, how we slide them, etc, how we treat image edges, etc.

But remember the “entries” in the kernel are weights **w** that we learn, just like we learn weights in a far simpler neural network.

So as we train it, we’re going to start to identify what kernels are most useful for, say, predicting wealth indices from satellite data.

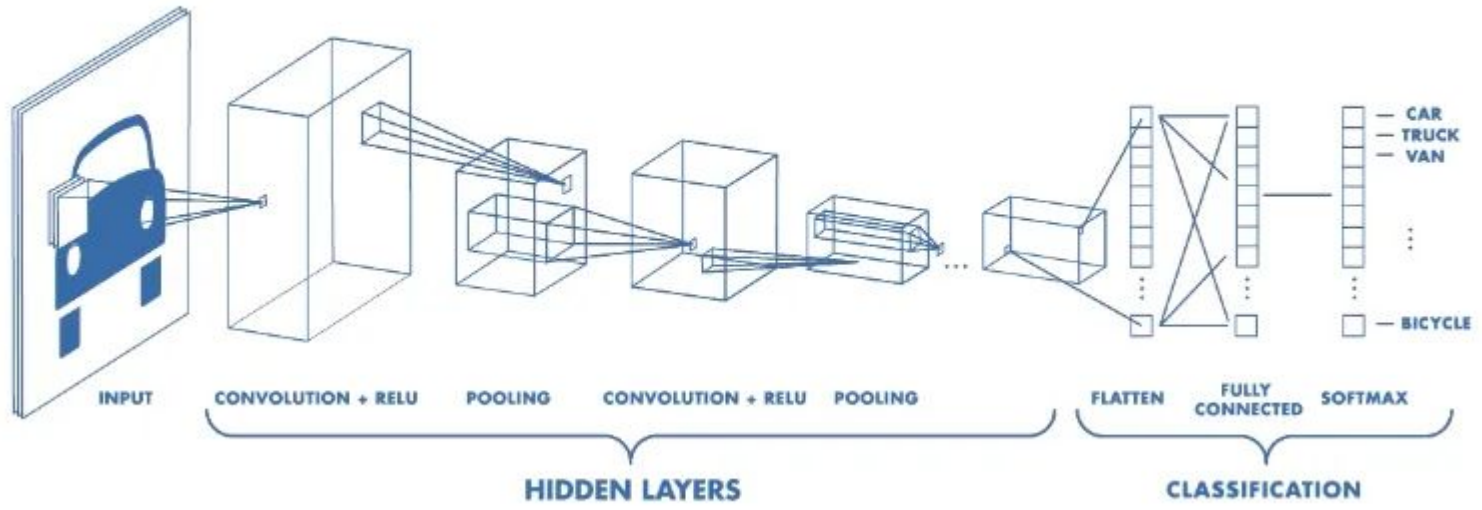
In a single convolution layer, we’ll have many kernels, each of which could be 3 dimensional (e.g., a 3x3 kernel that uses inputs from the red, green, and blue channels, making it 3x3x3).

Having several kernels lets us “look for” several different features that serve as inputs to the next layer in the network.

Broader CNN architecture

There will often be many other layers in a CNN (it's "deep"). Examples:

- "pooling" layers do things like take a max to reduce dimension.
- We can convolve again to look at higher level organization of features that we found.
- Toward the end we might use fully connected networks to map onto output nodes that represent our predictions



CNNs in R

Check out the keras package:

```
install.packages('keras')
install_keras()
```

There are options for GPU versions; likely not worth it unless you really get into this.

Lets you do things like build a model by sequentially adding layers of different types,

e.g.
layer_conv_2d
layer_max_pooling_2d
layer_dense

If you actually are interested in/want to do this

- Consider modifying an existing, well-performing architecture and fixing some of the weights to pre-tuned values (“transfer learning”). E.g., lop off last layer and tune to your data

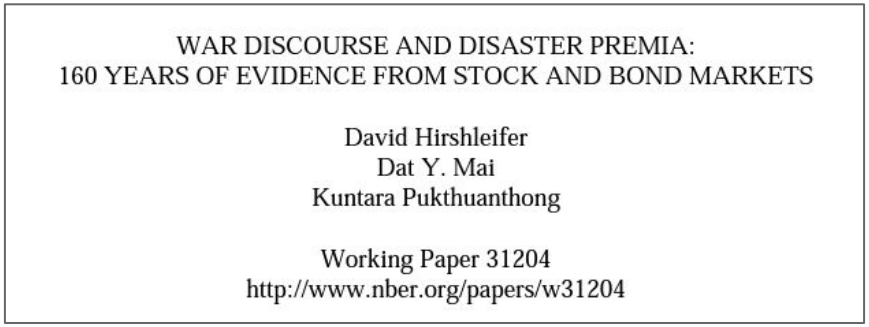
- You can expand small-ish training datasets by using tweaked versions of your original inputs (e.g., image rotation, flipping, zooming)

- Collaborate with an expert!

Text classification

Why should we care about classifying (or scoring) text?

We might want to use news articles, tweets, public comment on policies, etc as covariates or outcomes of interest.



WAR DISCOURSE AND DISASTER PREMIA:
160 YEARS OF EVIDENCE FROM STOCK AND BOND MARKETS

David Hirshleifer
Dat Y. Mai
Kuntara Pukthuanthong

Working Paper 31204
http://www.nber.org/papers/w31204

What could we do to turn text into a category (or number)?

- Create a labeled training set, somehow represent a document as a set of predictors, and use one of the supervised learning tools we saw to predict/classify documents in a test set.
- Use unsupervised clustering tools to group documents

Representing text documents quantitatively

Many approaches. Two ones that are simple(-ish) to understand:

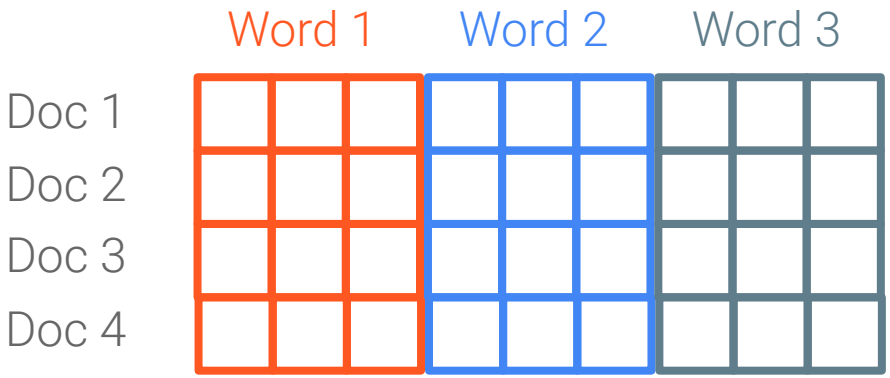
1) **TF-IDF** (an old idea): a document x word matrix with entries that correspond to the product of a) the frequency of that term in that document and b) the inverse frequency with which documents contain that term

2) **Word embedding** (e.g., word2vec): try to represent words using numeric vectors that preserve co-occurrence and context in a training corpus (i.e., related words have similar vectors). One way to do this is via a shallow neural network, e.g., try to predict a word from the 2 (or more) surrounding words:
   a) Inputs to network: one-hot encoding representation of surrounding words
   b) Output: one-hot encoding representation of target word
   c) Outputs from nodes in hidden layer are the embedding

   A document is a (padded or truncated) concatenation of word vectors

Supervised learning approaches to text classification

With documents represented by vectors, we can apply any of the predictive tools we want. As a simplified example, if we use word embeddings, we could construct a predictor matrix like this, which we can pair with training samples of categories, etc.



If you're interested in supervised classification

1) Don't recreate word embeddings. Start with one trained on a huge corpus (e.g., word2vec, GloVe, BERT). These can be further tuned for your context if needed.

2) There's usually text preprocessing to be done first (stripping punctuation, transforming case, etc). R and Python have packages for this (e.g., **tm** in R)

3) If you're excited about deep learning as your method, use an existing architecture that has proven performance. LOTS of material on the web

Semi-supervised text classification: motivation

- We may not want to use a supervised learning tool, especially if creating training data is expensive (e.g., labeling complex text documents)

- Unsupervised text classification algorithms can group documents (e.g., Latent Dirichlet Allocation, or LDA) but in many econ applications we care about groups having a particular meaning in order to test specific hypotheses.

- Semi-supervised tools offer an interesting balance, e.g.
  Seeded LDA (applied in Hirschleifer et al., 2023) or
  Keyword-Assisted Topic Models (Eshima, Imai, Sasaki, 2023)

Latent Dirichlet Allocation (LDA) & seeded LDA

LDA: documents are distributions over topics, topics are distributions over words

- This is a probabilistic model of how documents are generated
- Assume Dirichlet prior over both distributions
- Observe words, want to infer probability a document contains each of (a fixed number of latent) topics.

View as Bayesian problem, estimate posterior probability of a document belonging to a topic after observing words. Can estimate via expectation-maximization or Gibbs sampling.

Seeded LDA: define a set of “seed” words for each topic. Boost probability those words appear in the topic they belong to. See **seededlda** package in R

Dynamic programming

Curse of dimensionality, again

Numerical dynamic programming using conventional methods like value function iteration grows intractable as the number of state variables grows.

For example, if we discretize each of S state variables into M steps, we have M^S states at which we need to compute optimal actions and update the value function.

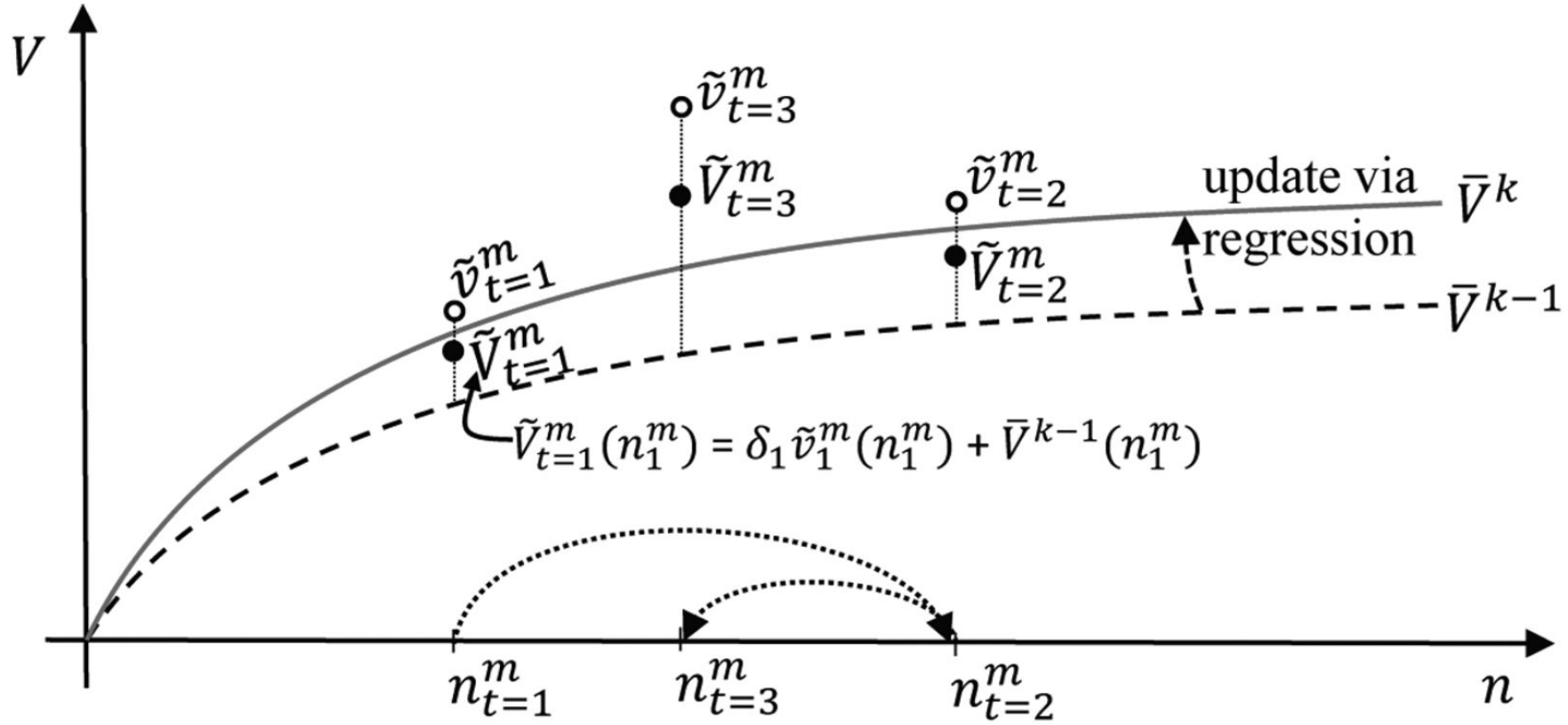
Some ML tools are good at approximating functions even in high dimensions!

Approximate dynamic programming (ADP)

Given a guess for a) the value function and b) the optimal policy function,

1. Repeatedly simulate forward in time using the current value function and policy. For each sample and time step, note the Bellman function value, and take a weighted average of it and the old value function at that state.
2. Fit a nonparametric (**ML**) model to the outputs from step 1.
3. Repeat until convergence

ADP intuition, graphically (Springborn & Haig, 2019)



Applications & practical advice

Prediction: US GDP growth (Chu & Qureshi, 2022)

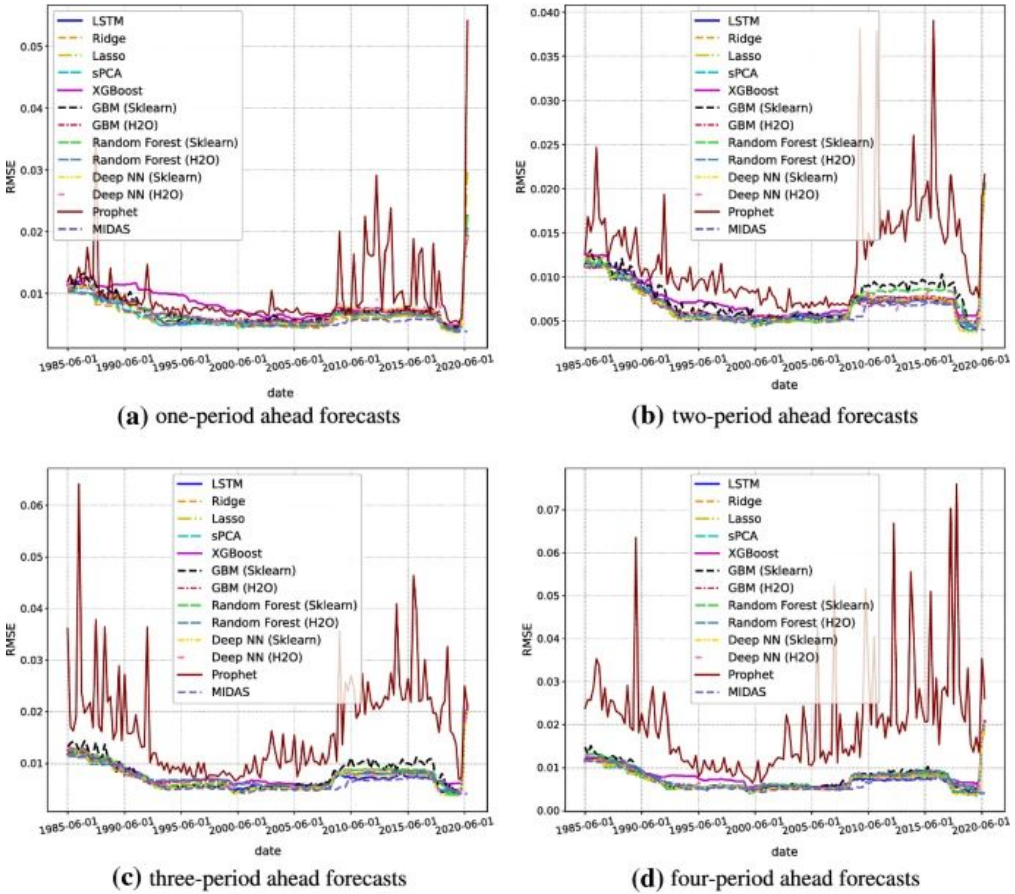
Want to be able to forecast US GDP growth from a set of predictors.

Compare a variety of methods using three different predictor datasets:

A) A set of 224 quarterly predictors
B) A set of 9 pre-selected “strong” quarterly predictors
C) B plus a high-frequency business condition index

Task is to predict GDP growth given 100 previous observations of predictors in set.
Train and validate on 60 of those, then predict using full 100

Chu & Qureshi (2022), continued



Example results for large set of predictors (A):

- Most methods fairly comparable
  (except Prophet, which is a Facebook time series model using piecewise linear trends, Fourier series, dummies, etc)

- *If anything*, boosting (XGBoost) struggles a bit compared to other ML tools

Classification as input to a model (Cengiz et al., 2022)

Question: how do minimum wage changes alter labor market outcomes?

Empirical Approach: event study

Challenge: don't know exactly who *might* be affected by policy. What's the relevant population to be including in the dataset (and averaging over)?

→ Use ML to predict membership in minimum wage set from demographics, then estimate over several groups defined by classification as a "likely" minimum wage worker (or not).

Cengiz et al. (2022) continued

"Classification" results: authors use boosting to use demographics to predict minimum wage (<125% of mandated minimum) status.
Really, boosting gives a probability of being a minimum wage worker, so can define lots of groups.

Then estimate model on different data subsets

Table 1. Demographic Characteristics for Each Predicted Probability Decile

	Teen	20 ≤ Age < 30	LTHS	HSG	Female	White	Black or Hispanic
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Most likely decile	.719	.038	.752	.145	.592	.837	.244
Probability decile 9	.047	.405	.534	.238	.674	.847	.359
Probability decile 8	.004	.341	.344	.437	.594	.834	.243
Probability decile 7	.004	.298	.187	.575	.571	.833	.351
Probability decile 6	.000	.191	.085	.660	.673	.873	.150
Probability decile 5	.000	.187	.100	.475	.492	.784	.253
Probability decile 4	.000	.178	.067	.236	.512	.794	.237
Probability decile 3	.000	.162	.004	.297	.404	.865	.175
Probability decile 2	.000	.088	.000	.143	.385	.848	.122
Least likely decile	.000	.015	.000	.039	.314	.741	.134

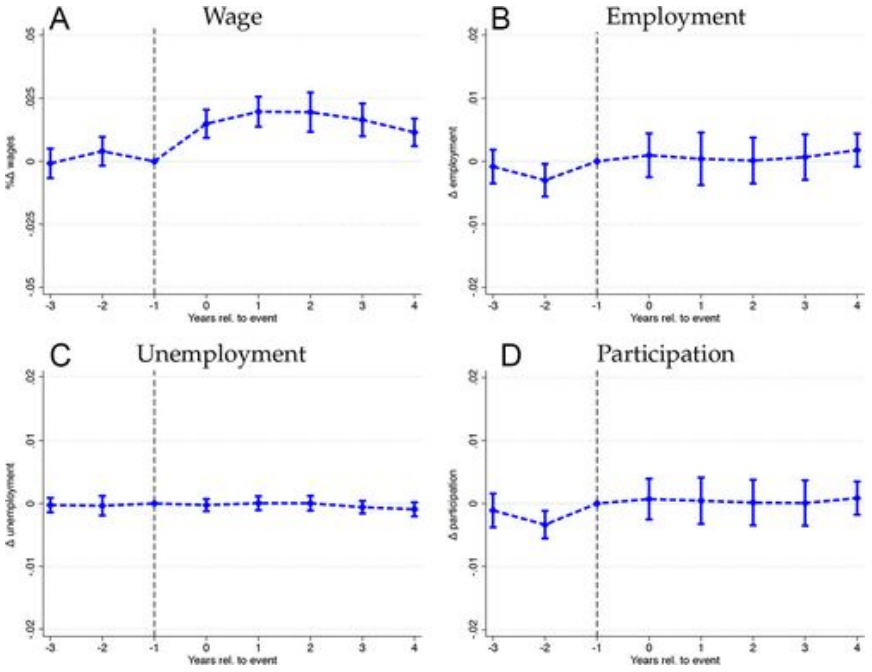
Cengiz et al. (2022) continued

Estimate event studies of the form

$$\mathscr{Y}_{st}^{g} = \sum_{\tau=-3}^{4} \beta_{\tau} treat_{st}^{\tau} + \Omega_{st} + \mu_{s} + \rho_{t} + u_{st},$$

Where state *s*, year *t*, and quarter tau index units, but the model is estimated separately for different definitions of groups *g* that could have been exposed.

Find no effect except for wages.



Counterfactual prediction (Prest et al., 2023)

Authors examine household energy use under an RCT.

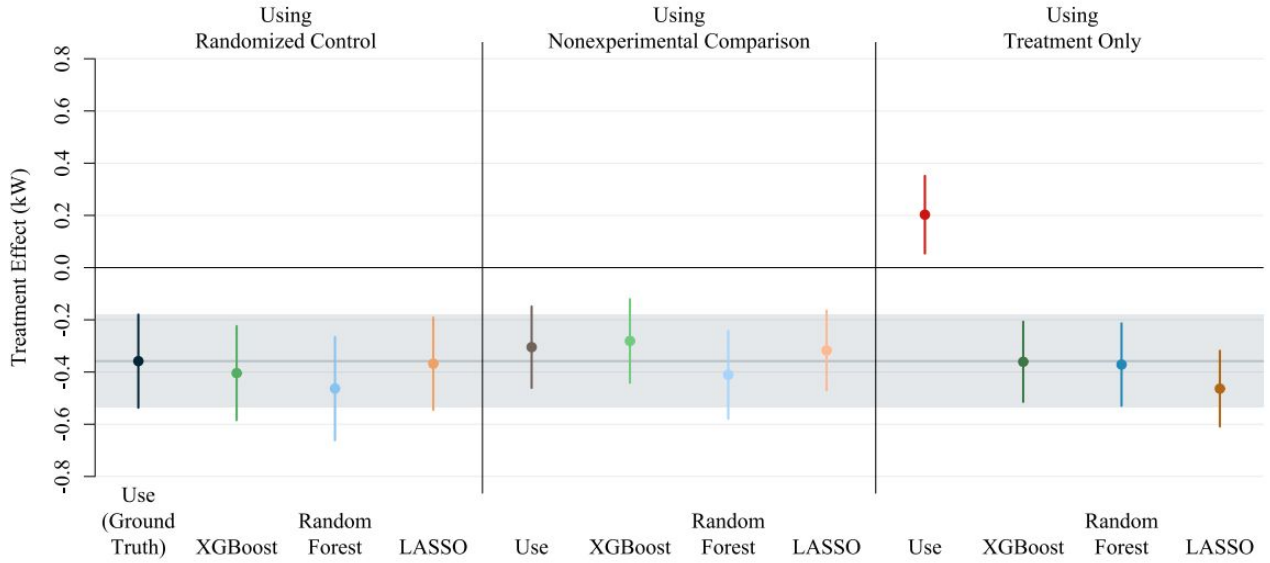
Use three different ML methods to predict counterfactuals for treated and control households, then use standard DiD on the residuals.

Compare results to RCT estimates.

Useful question: Because of regularization, prediction of counterfactuals using most ML tools is biased (but can be consistent). But how bad is it in practice?

Prest et al. (2023) continued

In short, ML-derived estimates generally do fairly well despite bias in counterfactual prediction. Could be because of large dataset and/or close matching in covariates across treatment/control obs so that prediction bias is similar across groups and is thus subtracted out in the final DiD step.



Single ML (Cole et al., 2020)

1) Predict pollution with weather & time trends using random forests
2) Randomly sample weather, feed through forest, repeat, and average predictions. Call this “weather-normalized” pollution levels
3) Use the output of 2 as a new outcome variable

This is essentially residualizing the outcome, which is the single ML approach that Chernozhukov et al., point out is biased (if used in a partially linear model).

Cole et al do this in an augmented synthetic control setting, but likely still problematic

Double ML (Dube et al., 2020)

Question: Are online labor markets like Amazon Mechanical Turk competitive?

Approach: Quantify relationship between how long a task sits unclaimed to the reward offered for the task.

Double ML: Have lots of data about the tasks (text, characteristics, etc), use double ML to try to isolate plausibly exogenous variation in task rewards.

$$\ln(duration) = -\eta \ln(reward) + g_0(Z) + \epsilon, \quad E[\epsilon \mid Z, \ln(reward)] = 0,$$

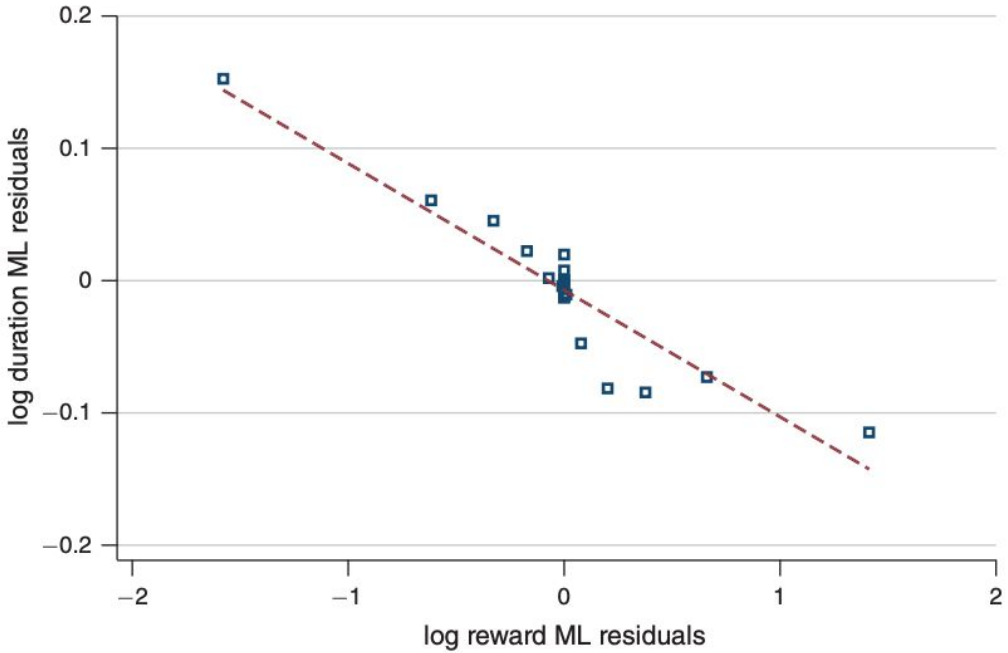
$$\ln(reward) = m_0(Z) + \mu, \quad E[\mu \mid Z] = 0.$$

Dube et al. (2020) continued

Find low elasticities using Double ML, suggesting task posters (employers) can set wages with low concern for that causing long delays/failing to match with workers

Also compare estimates to those From experiments, find Double ML yields similar point estimates.

Aside: some of their predictors in the nuisance functions use text representations we talked about



Causal forests (and dynamics) (Miller, 2020)

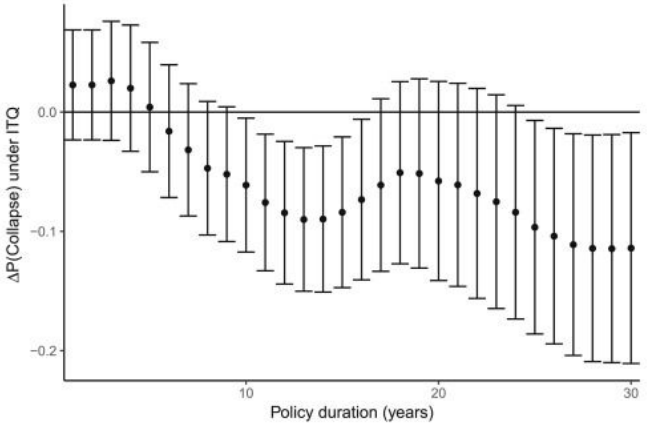
In principle, ML should be able to uncover not only cross-sectional heterogeneity in treatment effects but also information about learning and dynamics.

Pair causal forests with weak dynamic conditional independence, and allow splitting on not just pre-treatment characteristics, but also a) timing of introduction and b) duration of policy exposure.

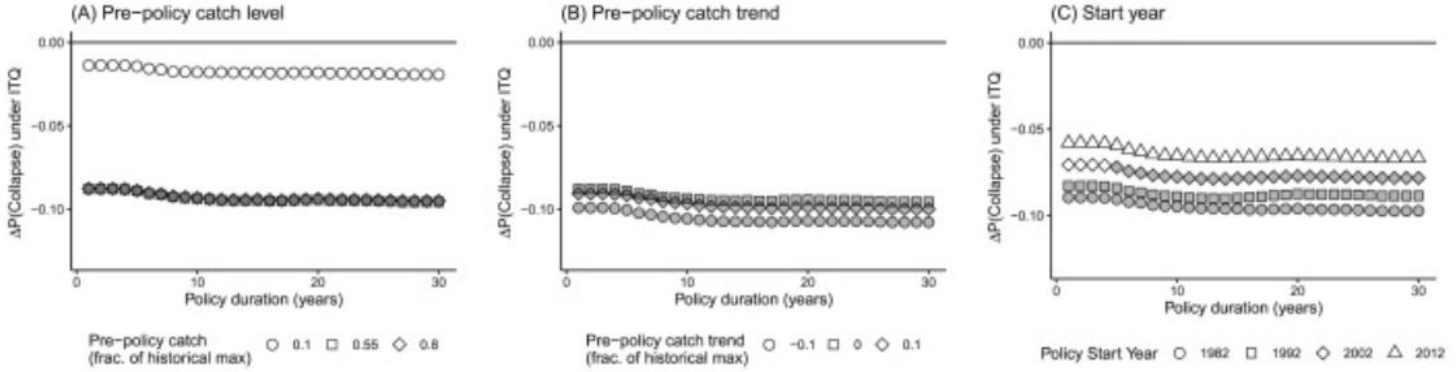
Miller (2020) continued

Looked at impacts of property rights in fisheries. In short, what might appear to be dynamic effects could actually be both cross-sectional heterogeneity and learning

Splitting on duration alone



Apparent dynamics likely a function of heterogeneity in other dimensions

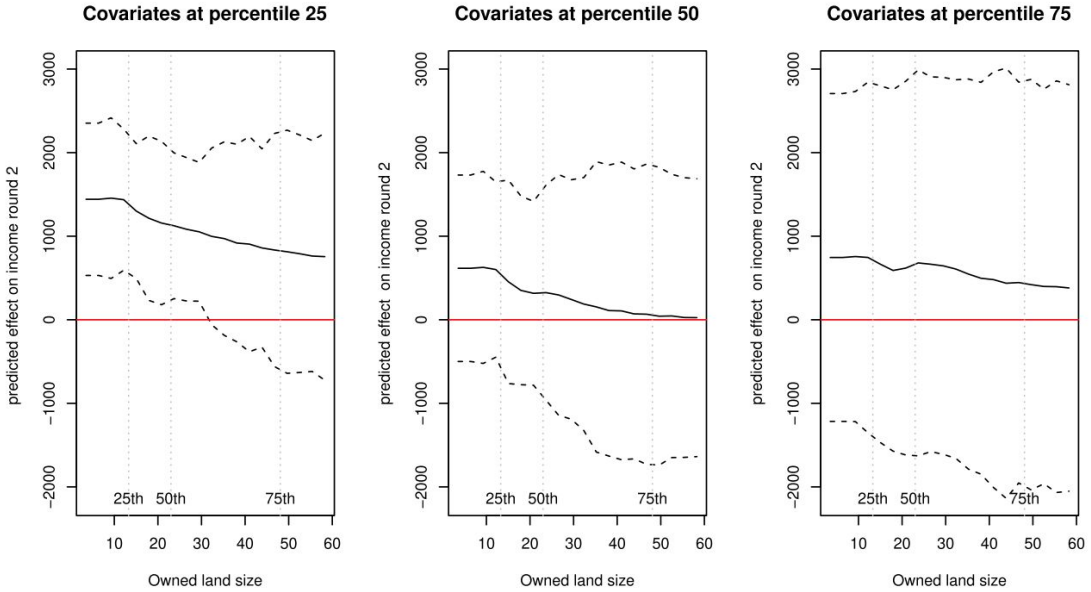


Using GRFs for heterogeneity (Carter et al. 2019)

Evaluate rural business development program in Nicaragua (experimentally).

Quantile effects suggest more effects on the right tail, and stability: high performers remain high performers.

GRFs (right) suggest estimated Effects more precise and larger for more disadvantaged farmers.

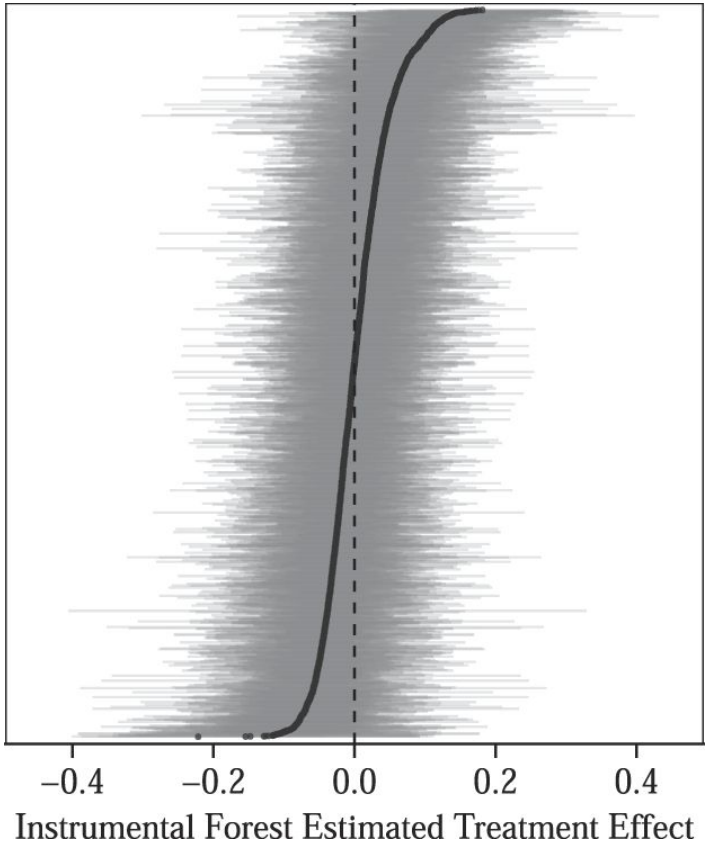


Instrumental forests, trust, & redistribution (Peyton 2020)

Question: how does trust in government alter support for redistributive policies

Approach: randomize exposure to fake op-eds designed to shift trust. Then relate trust to support, all via surveys on Mechanical Turk

Use instrumental forests to get at heterogeneity in relationship.
Amounts to a way to show a robust null.

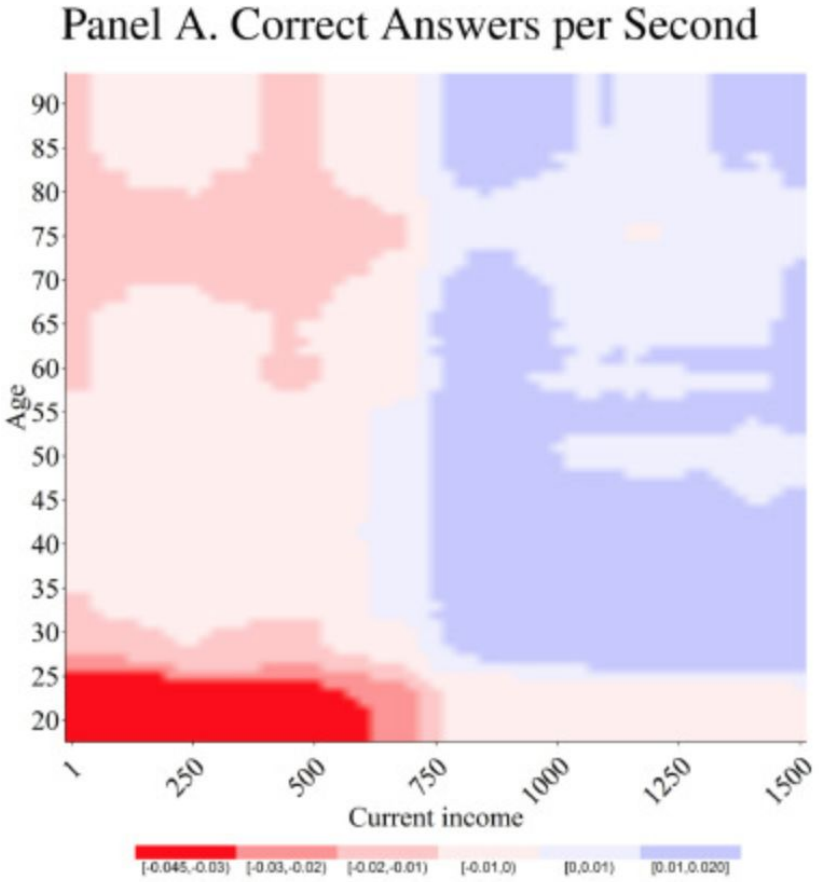


GRFs example 2 (Farbmacher et al., 2021)

Question: do financial circumstances affect cognitive performance (via attention)

Approach: Reanalyze experiment in which participants randomly assigned to perform tasks before vs after payday. See how it varies across covariates with GRFs.

Use variable importance to select a few dimensions for visualization



Text processing (Shen & Ross, 2021)

House listings contain both structured and unstructured data, but we really only (think we) understand how structured features (e.g., # bedrooms) affect prices.

Descriptions likely matter too.

Get at this by:

- **Coming up with a vector representation of property descriptions**
- Quantifying distance between vector representations of two properties
- Quantifying the “uniquness” of a property’s description based on how different it is from geographically nearby properties.

Step 1 uses a text embedding approach to represent the entire description (not just a single word) as a vector. But the idea is similar.

Advice

You are still useful

Aside from being users of new tools, what can we contribute?

Economic theory, objectives, etc. could guide development of new methods:

- Are there restrictions (e.g., monotonicity) that should impose on predictions?
- What's the right loss function for prediction problems? E.g., is underprediction (or false negatives) more costly to a firm?
- Are there other computationally expensive functions embedded in numerical models that we might want to employ simulation/approximation tools for?
- Are there other choices we make that we might want to instead learn? (e.g., can we use classifiers to identify choice sets for discrete choice models?)

Help all of us understand when these tools (don't) work well in practice!

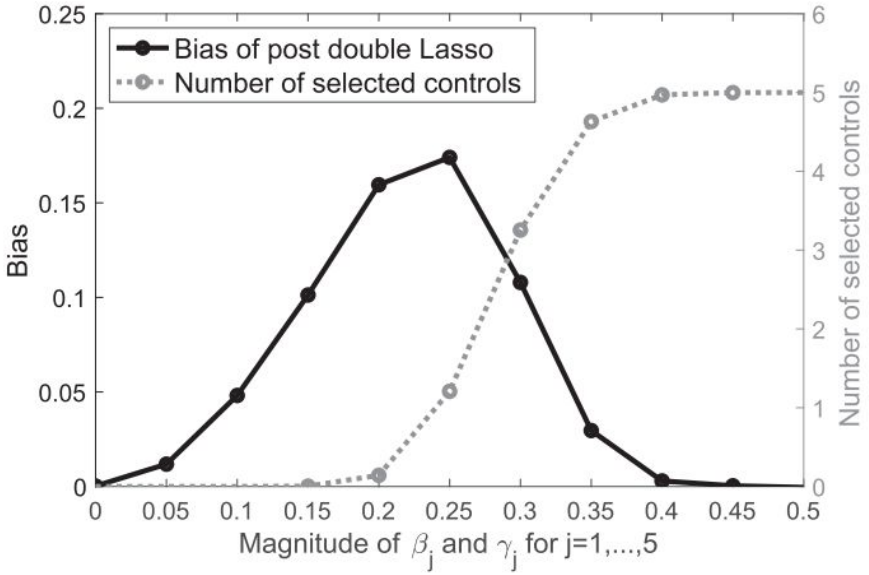
Thou shalt not run blindly into the Machine Light

As (hopefully sufficiently) emphasized, much of the justification for these estimators is asymptotic.

We're still learning a lot about how they perform in finite samples. And it isn't always pretty.
Wüthrich & Zhu, 2023 (see figure) highlight concerns about post double LASSO

Angrist & Frandsen (2022) highlight some concerns about ML + IV

FIGURE 1.—ILLUSTRATION OF BIAS AND DOUBLE UNDERSELECTION



Thou shalt still think carefully about identification

Causal approaches that use ML still make identifying assumptions.

- Causal forests rely on neighborhoods being small enough that conditional ignorability is plausibly satisfied
- Double ML for a partially linear model relies on selection on observables

Bad controls are… still bad! (Hünermund et al., 2022).

- If you include them in your set of confounders for double LASSO to select, it will probably pick them (e.g., strongly related to treatment)

Thou shalt think carefully about which predictor to use

Do you expect only a small fraction of your predictors matter?
Consider LASSO

Do you expect a lot of nonlinearity?
Consider forests or boosting or SVMs with a nonlinear kernel

Do you think the function you want to learn is actually piecewise constant?
Consider a tree.

Thou shalt preprocess your predictors appropriately

If you are using a form of regularization in your tool (e.g., LASSO) center and scale your predictors first. Otherwise you're unfair to some of them.

If your method doesn't do some type of basis expansion to handle nonlinearity, do it yourself
- e.g., for LASSO, include quadratics, products, etc. of your original covariates.

Thou shalt tune

Hyperparameters are **VERY** important for predictive accuracy (and for establishing asymptotic results). Take the time to tune them.

Extreme example:

A giant weight on the L1 norm in LASSO will just mean you force all coefficients to zero, so you'll have a constant predictor. Probably not useful.