Making Decisions in High Dimensions

Matt Taddy, Chicago Booth

faculty.chicagobooth.edu/matt.taddy/research

High Dimensional Decisions

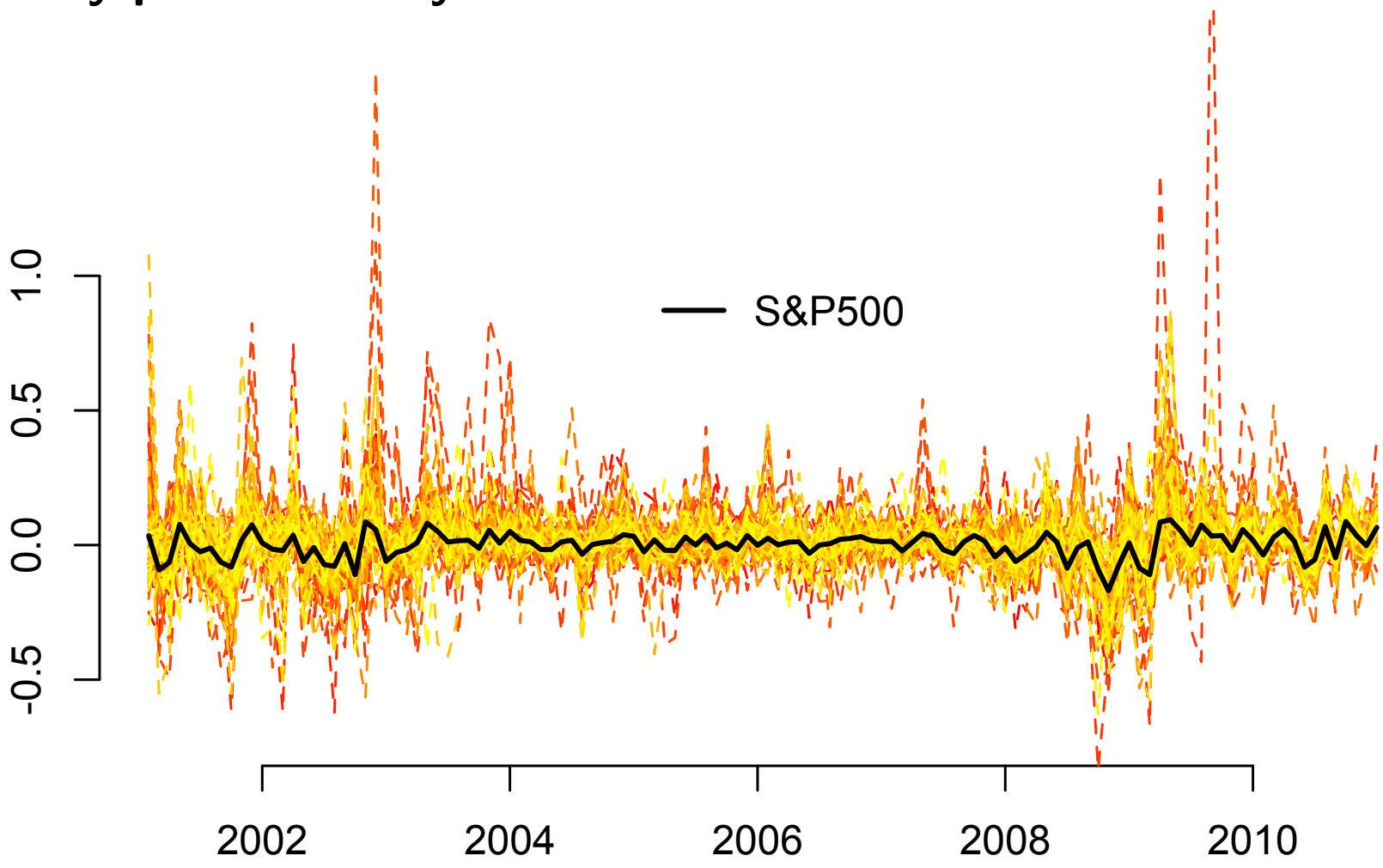
One of the ways that data is 'Big' is in the number of inputs.

This number often grows with the sample size (e.g., words in text, websites in browsers) and you never reach a statistically safe place.

For this type of Big, we need dimension reduction techniques: project from the full input set down to a useful low-D summary.

Crucially, this must be focused on the decisions you'd like to make.

Fancy plot: monthly stock returns



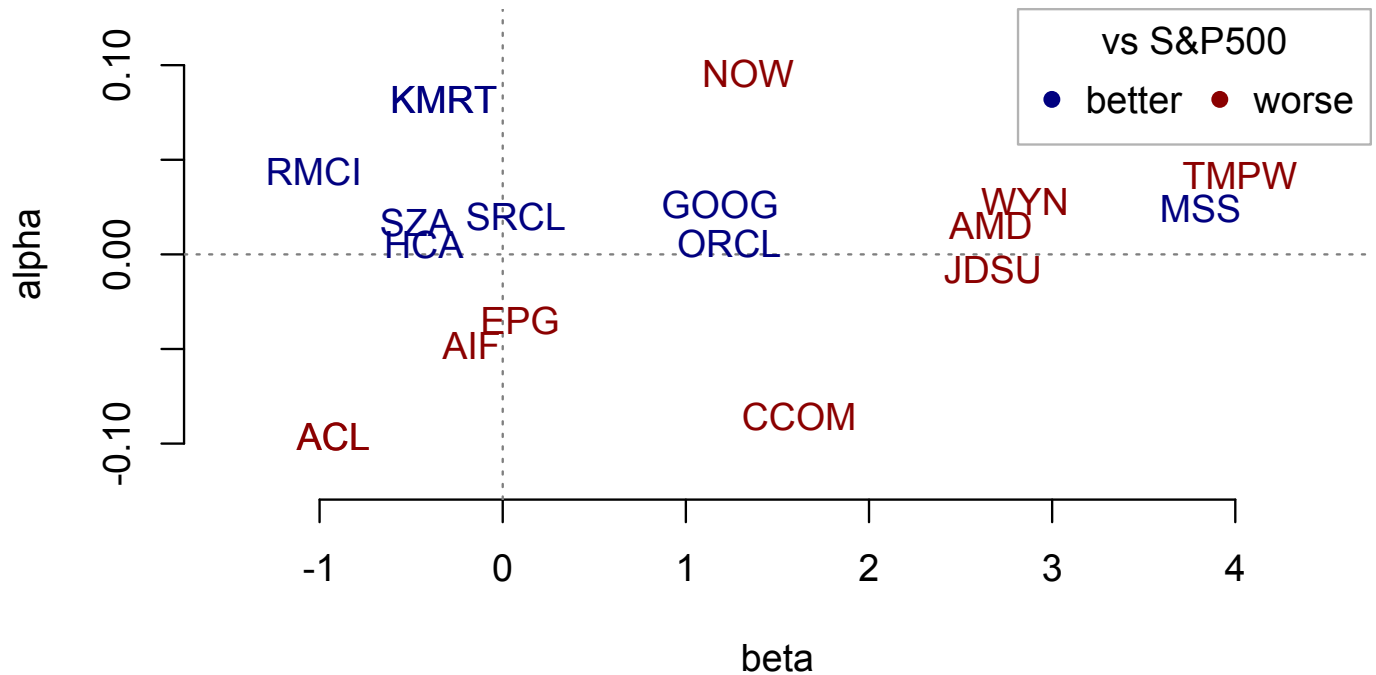
What do we learn?

Useful plot: market model coefficients

Fit a line between stock returns R_t and market returns M_t (SP).

$$R_t \approx \alpha + \beta M_t$$

α is money you make regardless of what the market does.
β is the asset's sensitivity to broad market movements.



Many problems in BD involve a response of interest ('y')
and a set of covariates ('**x**') to be used for prediction.

If your 'decision' is to predict y for new $\mathbf{x}$, we need to reduce dimension in the direction of y. That is, we want to map $\mathbf{x} \rightarrow \hat{y}$.

A general tactic is to deal in averages and lines.
We'll model the conditional mean for y given $\mathbf{x}$,

$$\mathbb{E}[\ y \mid \mathbf{x}\ ] = f(\mathbf{x}'\boldsymbol{\beta})$$

$\mathbf{x} = [1, x_1, x_2, \ldots x_p]$ is your vector of covariates.

$\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \ldots \beta_p]$ are the corresponding coefficients.

Some basic facts about linear models

The model is always $\mathbb{E}[y|\mathbf{x}] = f(\mathbf{x}\boldsymbol{\beta})$.

- Gaussian (linear): $y \sim \mathrm{N}(\mathbf{x}\boldsymbol{\beta}, \sigma^2)$.
- Binomial (logistic): $\mathrm{p}(y = 1) = e^{\mathbf{x}\boldsymbol{\beta}}/(1 + e^{\mathbf{x}\boldsymbol{\beta}})$.

Likelihood (LHD) is $\mathrm{p}(y_1|\mathbf{x}_1) \times \mathrm{p}(y_2|\mathbf{x}_2) \cdots \times \mathrm{p}(y_n|\mathbf{x}_n)$.

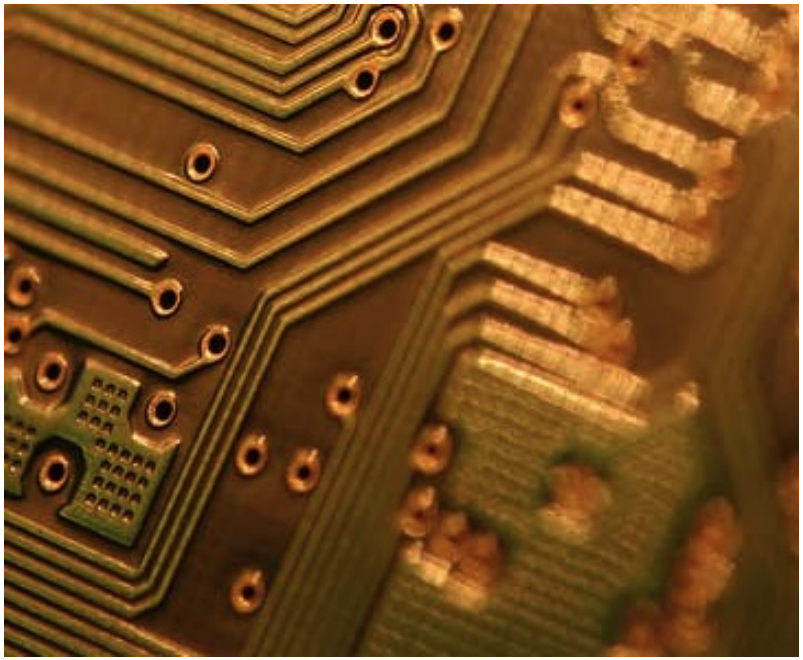
The Deviance (dev) is proportional to -log(LHD).

$\hat{\boldsymbol{\beta}}$ is commonly fit to maximize LHD $\Leftrightarrow$ minimize deviance.

Fit is summarized by $R^2 = 1 - \mathrm{dev}(\hat{\boldsymbol{\beta}})/\mathrm{dev}(\boldsymbol{\beta} = 0)$.

The only R^2 we ever really care about is out-of-sample R^2.

Example: Semiconductor Manufacturing Processes



Very complicated operation
Little margin for error.

Hundreds of diagnostics
Useful or debilitating?

We want to focus reporting and better predict failures.

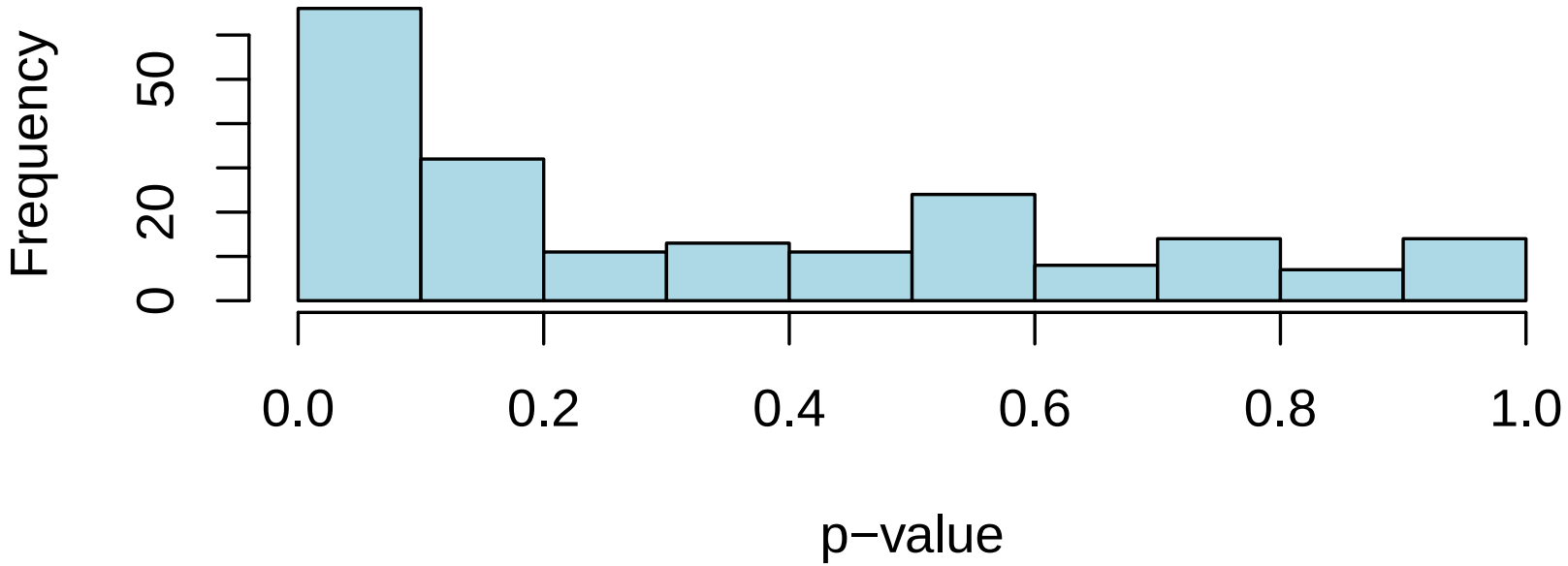
$\mathbf{x}$ is 200 input signals, y has 100/1500 failures.

Logistic regression for failure of chip i is

$$p_i = \mathrm{p}(\mathtt{fail}_i|\mathbf{x}_i) = e^{\alpha+\mathbf{x}_i\boldsymbol{\beta}}/(1+e^{\alpha+\mathbf{x}_i\boldsymbol{\beta}})$$

The full model has $R^2 = 0.56$ (based on *binomial* deviance).

The p-values for these 200 coefficients:



Some are clustered at zero, the rest sprawl out to one.

FDR of $q = 0.1$ yields $\alpha = 0.0122$ p-value rejection cut-off.

Implies 25 'significant', of which approx 22-23 are true signals.

A *cut* model, using only these 25 signals, has $R^2_{cut} = 0.18$.
This is much smaller than the full model's $R^2_{full} = 0.56$.

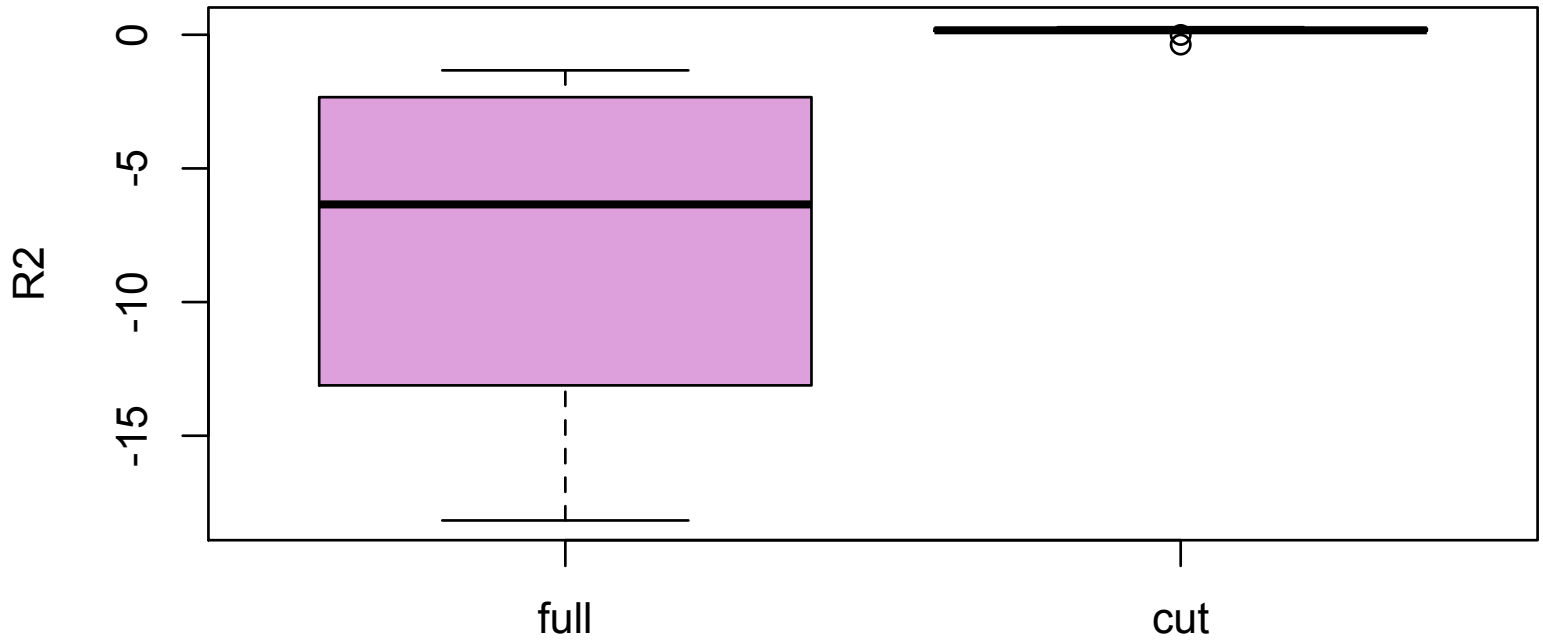
In-Sample (IS) R^2 *always* increases with more covariates.
This is exactly what MLE $\hat{\beta}$ is fit to maximize.
But how well does each model predict *new* data?

An out-of-sample (OOS) experiment

- split the data into 10 random subsets ('folds').
- Do 10x: fit model $\hat{\beta}$ using only 9/10 of data,
  and record R^2 on the left-out subset.

These OOS R^2 give us a sense of how well each
model can predict data that it has not already seen.

We gain predictive accuracy by *dropping* variables.



Cut model has mean OOS R2 of 0.09, about 1/2 in-sample R2.

The full model is terrible. It is overfit and worse than $\bar{y}$.
Negative R2 are more common than you might expect.

All that matters is Out-of-Sample R^2.
We don't care about In Sample R^2.

Using OOS experiments to choose the best model is called *cross validation*. It will be a big part of our big data lives.

Selection of 'the best' model is at the core of all big data.

But before getting to selection, we first need strategies to build good sets of candidate models to choose amongst.

Regularization

The key to contemporary statistics is regularization:

depart from optimality to stabilize a system.

Common in engineering: I wouldn't drive on an optimal bridge.

We minimize deviance

$$\min -\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta})$$

Regularization

The key to contemporary statistics is regularization:

depart from optimality to stabilize a system.

Common in engineering: I wouldn't drive on an optimal bridge.

We minimize deviance plus a cost on the size of coefficients.

$$\min -\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta}) + \lambda \sum_k |\beta_k|$$

This particular cost gives the 'lasso': the new least squares.

Decision theory: Cost in Estimation

Decision theory is based on the idea that choices have costs.
Estimation and hypothesis testing: what are the costs?

Estimation:

Deviance is the cost of distance between data and the model.
Recall: $\sum_i (y_i - \hat{y}_i)^2$ or $-\sum_i y_i \log(\hat{p}_i) - (1 - y_i) \log(1 - \hat{p}_i)$.

Testing:

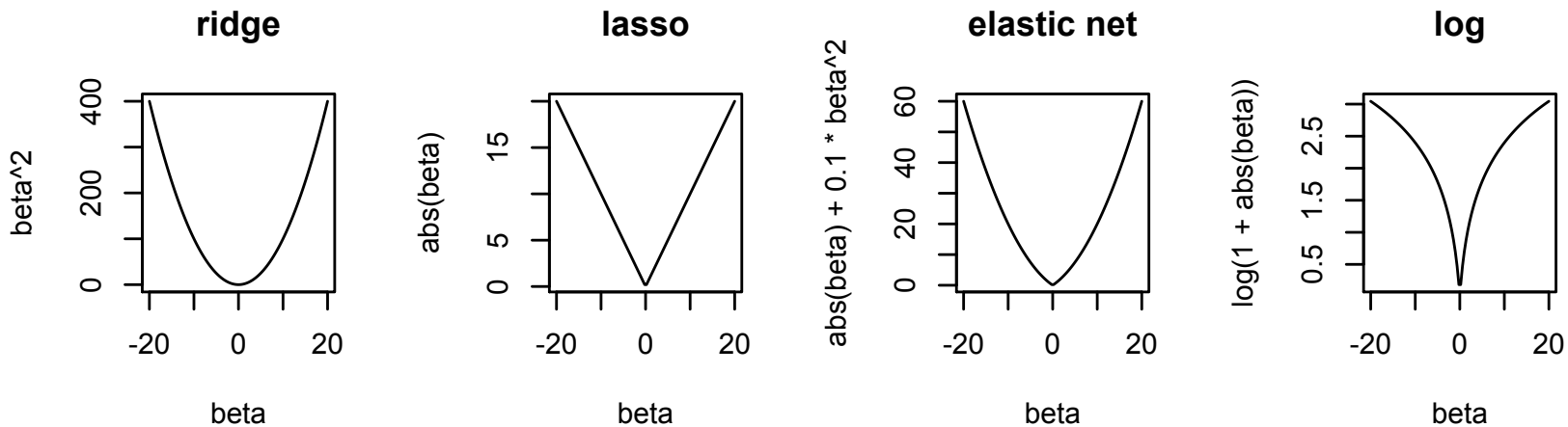
Since $\hat{\beta}_j = 0$ is *safe*, it should cost us to decide otherwise.

$\Rightarrow$ The cost of $\hat{\beta}$ is deviance plus a penalty away from zero.

[Sparse] Regularized Regression

$$\min \left\{ -\frac{2}{n} \log \mathrm{LHD}(\boldsymbol{\beta}) + \lambda \sum_j c(\beta_j) \right\}$$

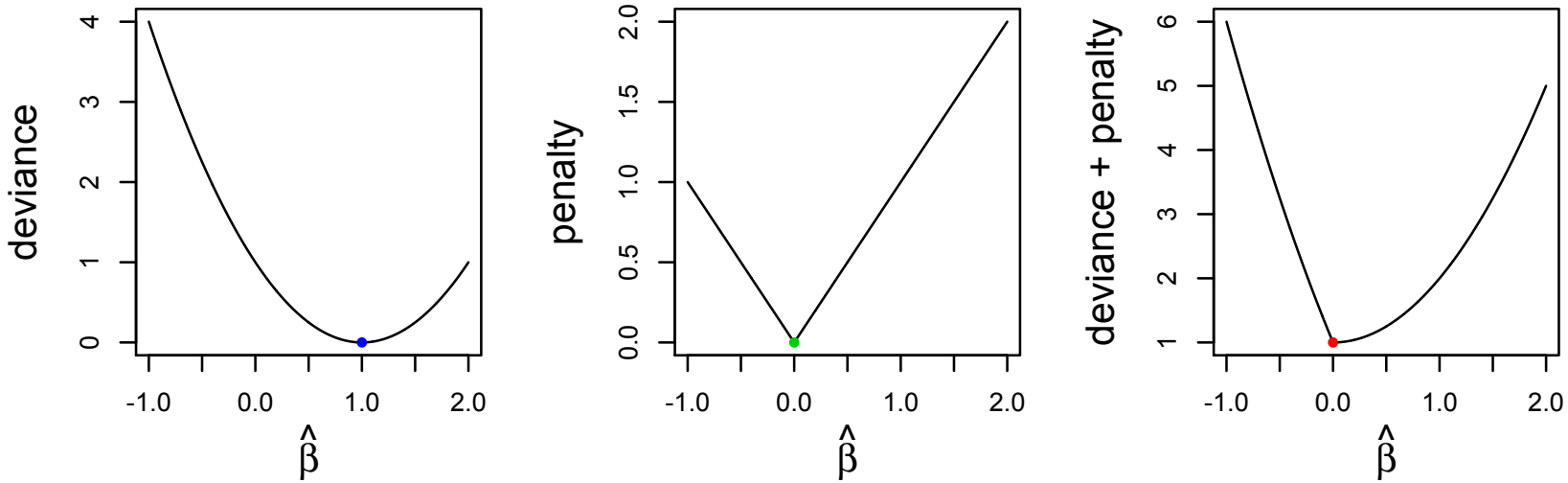
$\lambda > 0$ is the penalty weight, c is a cost (penalty) function.
$c(\beta)$ will be lowest at $\beta = 0$ and we pay more for $|\beta| > 0$.



Options: ridge β^2, lasso $|\beta|$, elastic net $\alpha\beta^2 + |\beta|$, $\log(1 + |\beta|)$.

Penalization can yield automatic variable selection

The minimum of a smooth + pointy function can be at the point.



Anything with an absolute value (e.g., lasso) will do this.

There are MANY penalty options and far too much theory.

Think of lasso as a baseline, and others as variations on it.

Lasso Regularization Paths

The lasso fits $\hat{\boldsymbol{\beta}}$ to minimize $-\frac{2}{n}\log \text{LHD}(\boldsymbol{\beta}) + \lambda \sum_j |\beta_j|$.

We'll do this for a *sequence* of penalties $\lambda_1 > \lambda_2 ... > \lambda_T$.

Then we can apply model selection tools to choose best $\hat{\lambda}$.

Path estimation:

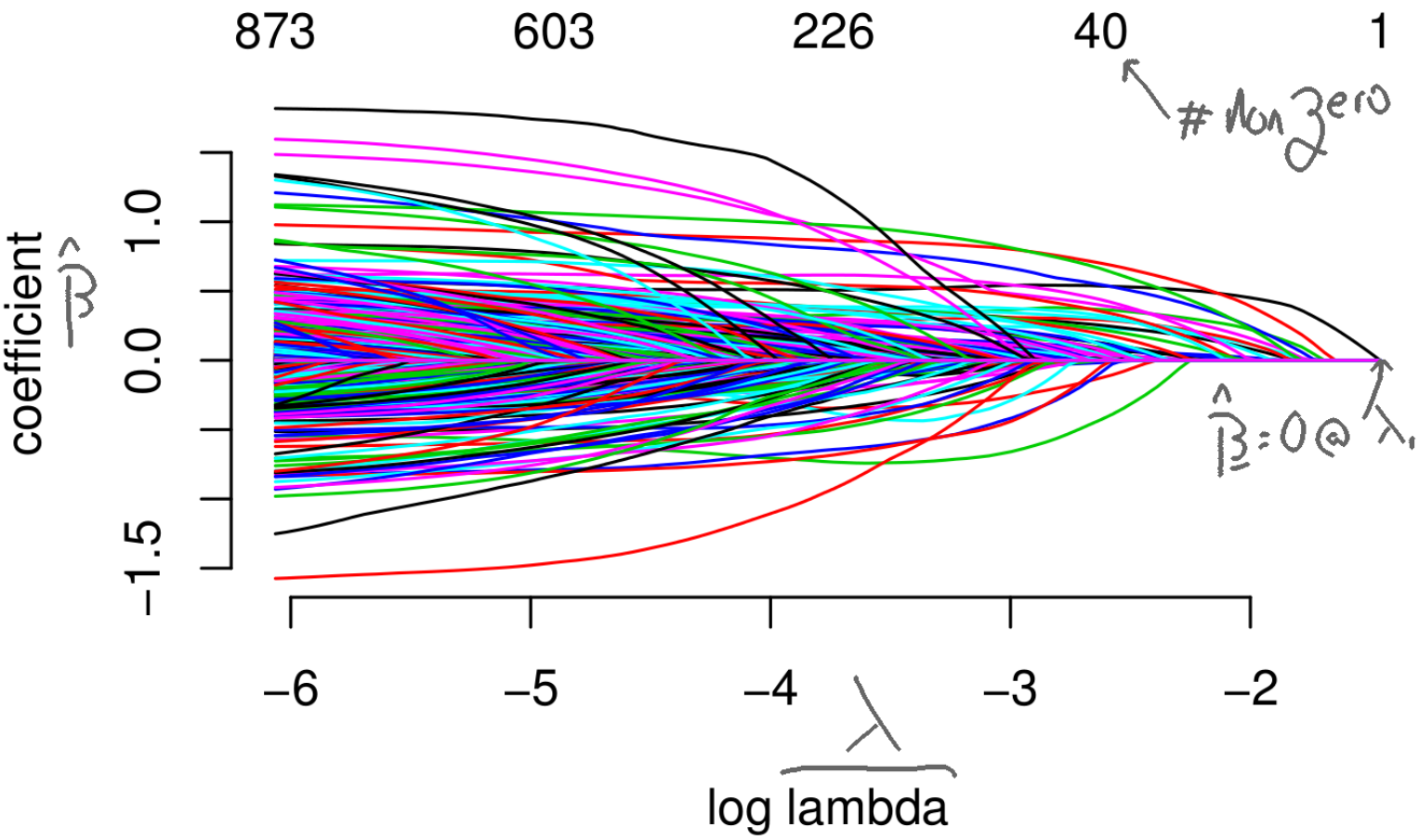
Start with big λ_1 so big that $\hat{\boldsymbol{\beta}} = \mathbf{0}$.

For $t = 2 \dots T$: update $\hat{\boldsymbol{\beta}}$ to be optimal under $\lambda_t < \lambda_{t-1}$.

Since estimated $\hat{\boldsymbol{\beta}}$ changes smoothly along this path:

- It's fast! Each update is easy.
- It's stable: optimal λ_t may change a bit from sample to sample, but that won't affect the model much.

The whole enterprise is easiest to understand visually.



The algorithm moves *right to left*.

The y-axis is $\hat{\boldsymbol{\beta}}$ (each line a different $\hat{\beta}_j$) as a function of λ_t.

Example: web browser data

The previous plot is household log-online-spend regressed onto % of time spent on various websites (each β_j a different site).

Browsing and purchasing behavior for households:

I've extracted 2006 data for the 1000 most heavily trafficked websites and for 10,000 households that spent at least 1$.

Why do we care? Predict consumption from browser history.

faculty.chicagobooth.edu/matt.taddy/teaching/comscore.R

There are many packages for fitting lasso regressions in R.

`glmnet` is most common. `gamlr` is my contribution.
These two are very similar, and they share syntax.

Big difference is what they do beyond a simple lasso:.

- `glmnet` does an 'elastic net': $c(\beta) = |\beta| + \nu\beta^2$.
- `gamlr` does a 'gamma lasso': $c(\beta) \approx \log(\nu + |\beta|)$.

Both use the `Matrix` library representation for sparse matrices.

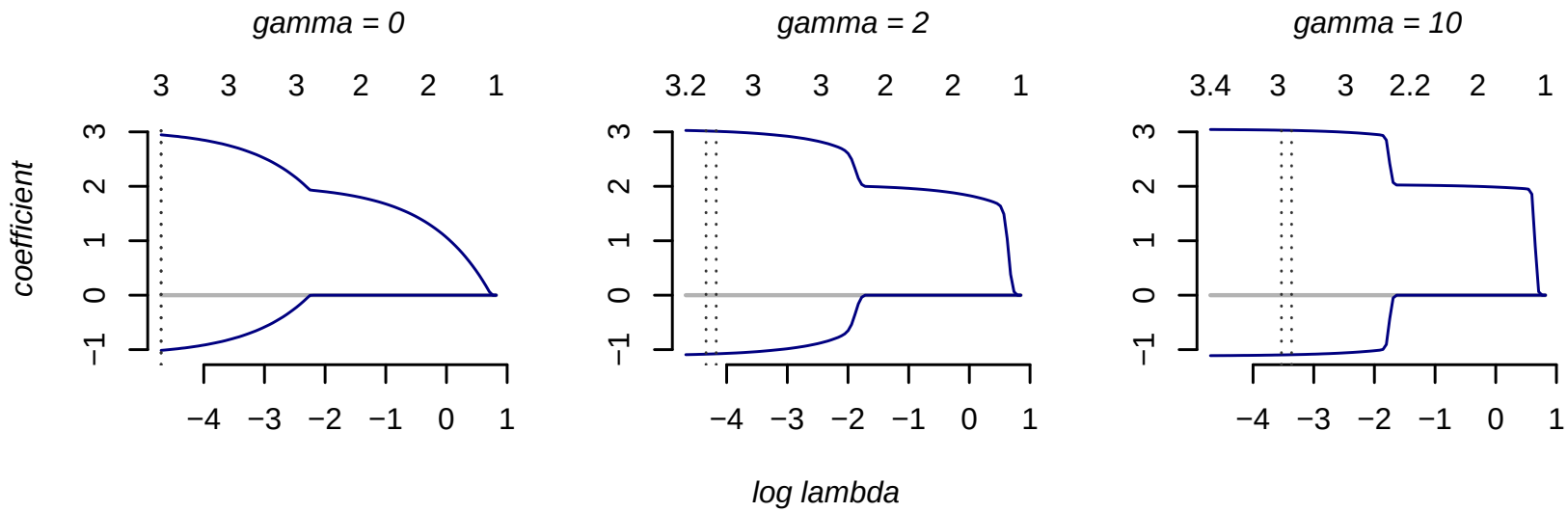
A little bit more info on `gamlr`:

it actually applies a weighted L_1 penalty: $\lambda \sum_j \omega_j |\beta_j|$.

These weights adapt along the regularization path as a function of the previous segment's solution

$$\omega_j^t = \left(1 + \gamma |\hat{\beta}_j^{t-1}|\right)^{-1} \quad j = 1 \dots p \tag{1}$$

This lets you measure large signals with less bias.



See *Paths of One-Step Estimators* on my website.

Regularization and Selection

The lasso minimizes $-\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta}) + \lambda \sum_j |\beta_j|$.

This 'sparse regularization' auto-selects the variables.

Sound too good to be true? You need to choose λ.

Think of $\lambda > 0$ as a signal-to-noise filter: *like squelch on a radio.*

We'll use cross validation or information criteria to choose.

Path algorithms are key to the whole framework:

⋆ They let us quickly enumerate a set of candidate models.

⋆ This set is stable, so selected 'best' is probably pretty good.

A recipe for model selection.

1. Find a manageable set of candidate models (i.e., such that fitting all models is fast).
2. Choose amongst these candidates the one with best predictive performance *on unseen data*.

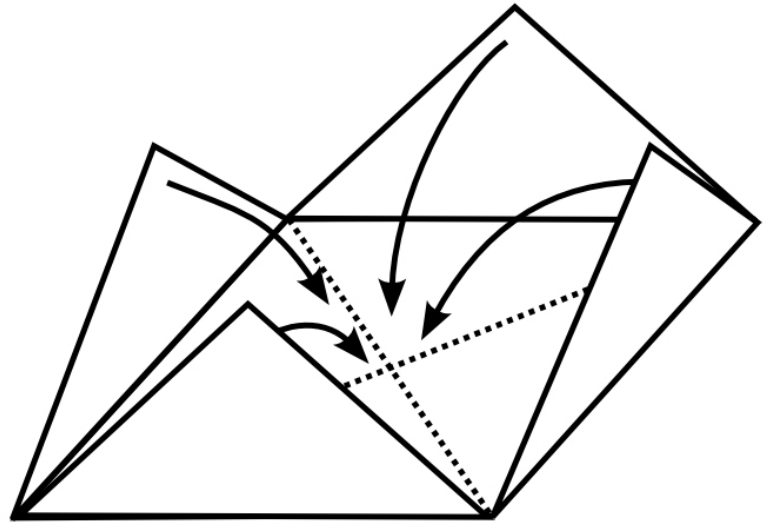
1. is what the lasso paths provide.
2. Seems impossible! But it's not ...

First, define predictive performance via 'deviance'.

Then, we need to *estimate* deviance for a fitted model applied to *new independent observations* from the true data distribution.

K-fold Cross Validation

One option is to just take
repeated random samples.
It is better to 'fold' your data.



- Sample a random ordering of the data
  (important to avoid order dependence)
- Split the data into K folds: 1st 100/K%, 2nd 100/K%, etc.
- Cycle through K CV iterations with a single fold left-out.

Be careful to not cheat: for example, with the FDR cut model we've already used the full n observations to select the 25 strongest variables. It is not surprising they do well 'OOS'.
The rules: if you do something to the data, do it *inside* CV.

CV Lasso

The lasso path algorithm minimizes $-\frac{2}{n}\log \text{LHD}(\boldsymbol{\beta}) + \lambda_t \sum_j |\beta_j|$ over the sequence of penalty weights $\lambda_1 > \lambda_2 \ldots > \lambda_T$.

This gives us a path of T fitted coefficient vectors, $\hat{\boldsymbol{\beta}}_1 \ldots \hat{\boldsymbol{\beta}}_T$, each defining deviance for new data: $-\log \text{p}(\mathbf{y}^{new} \mid \mathbf{X}^{new}\hat{\boldsymbol{\beta}}_t)$.

Set a sequence of penalties $\lambda_1 \ldots \lambda_T$.
Then, for each of $k = 1 \ldots K$ folds,

- Fit the path $\hat{\boldsymbol{\beta}}^k_1 \ldots \hat{\boldsymbol{\beta}}^k_T$ on all data *except* fold k.
- Get fitted deviance *on left-out data*: $-\log \text{p}(\mathbf{y}^k \mid \mathbf{X}^k\hat{\boldsymbol{\beta}}_t)$.

This gives us K draws of OOS deviance for each λ_t.

Finally, use the results to choose the 'best' $\hat{\lambda}$, then re-fit the model to *all of the data* by minimizing $-\frac{2}{n}\log \text{LHD}(\boldsymbol{\beta}) + \hat{\lambda} \sum_j |\beta_j|$.

CV Lasso

Both `gamlr` and `glmnet` have functions to wrap this all up.
The syntax is the same; just preface with `cv.`.

```
cv.spender <- cv.gamlr(xweb, log(yspend))
```

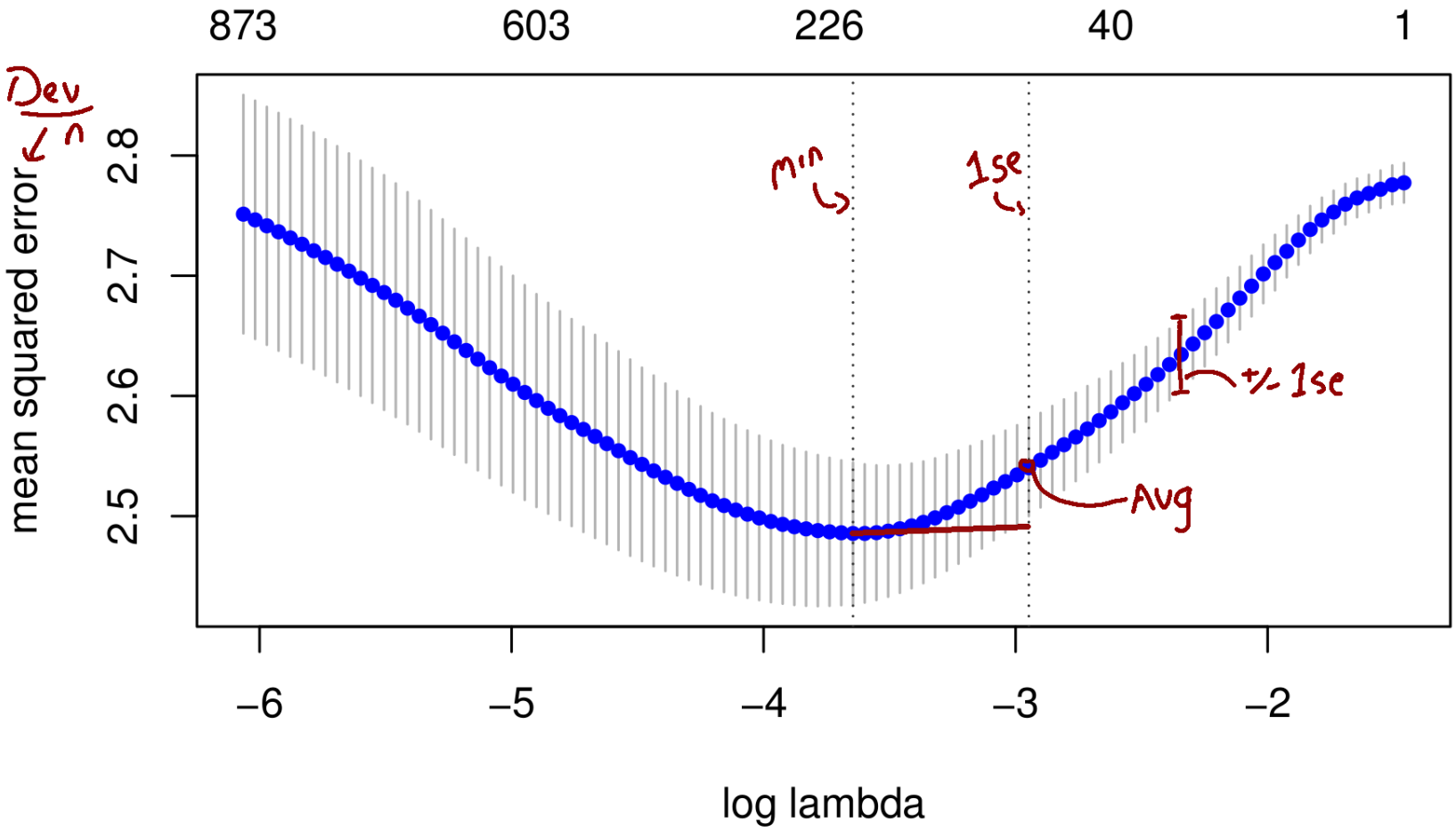
Then, `coef(cv.spender)` gives you $\hat{\beta}_t$ at the 'best' λ_t

- `select="min"` gives λ_t with *min average OOS deviance*.
- `select="1se"` defines best as *biggest* λ_t *with average OOS deviance no more than 1SD away from the minimum.*

`1se` is default, and balances prediction against false discovery.
`min` is purely focused on predictive performance.

CV Lasso

Again, the routine is most easily understood visually.



Both selection rules are good; `1se` has extra bias for simplicity.

Problems with Cross Validation

It is time consuming: When estimation is not instant, fitting K times can become unfeasible even K in 5-10.

It can be unstable: imagine doing CV on many different samples. There can be large variability on the model chosen.

Still, some form of CV is used in most DM applications.

Alternatives to CV: **Information Criteria**

Many 'Information Criteria' out there: AICc, AIC, BIC, ...
These approximate distance between a model and 'the truth'.
You can apply them by choosing the model with minimum IC.

Most common is Akaike's AIC $=$ Deviance $+ 2df$.

$df =$ 'degrees of freedom' used in your model fit.
For lasso and MLE, this is just the # of nonzero $\hat{\beta}_j$.

AIC overfits in high dimensions

The AIC is atually estimating OOS deviance: what your deviance would be on another *independent* sample of size n.

IS deviance is too small, since the model is tuned to this data.
Some deep theory shows that IS - OOS deviance $\approx 2df$.

$$\Rightarrow \text{AIC} \approx \text{OOS deviance.}$$

Its common to claim this approx (i.e., AIC) is good for 'big n'.
Actually, its only good for big n/df.

In Big Data, df (# parameters) can be huge. Often $df \approx n$.
In this case the AIC will be a bad approximation: it overfits!

AIC corrected: AICc

AIC approximates OOS deviance, but does a bad job for big df.

In linear regression an improved approx to OOS deviance is

$$\text{AICc} = \text{Deviance} + 2df\,\mathbb{E}\left[\frac{\sigma^2}{\hat{\sigma}^2}\right] = \text{Deviance} + 2df\,\frac{n}{n - df - 1}$$

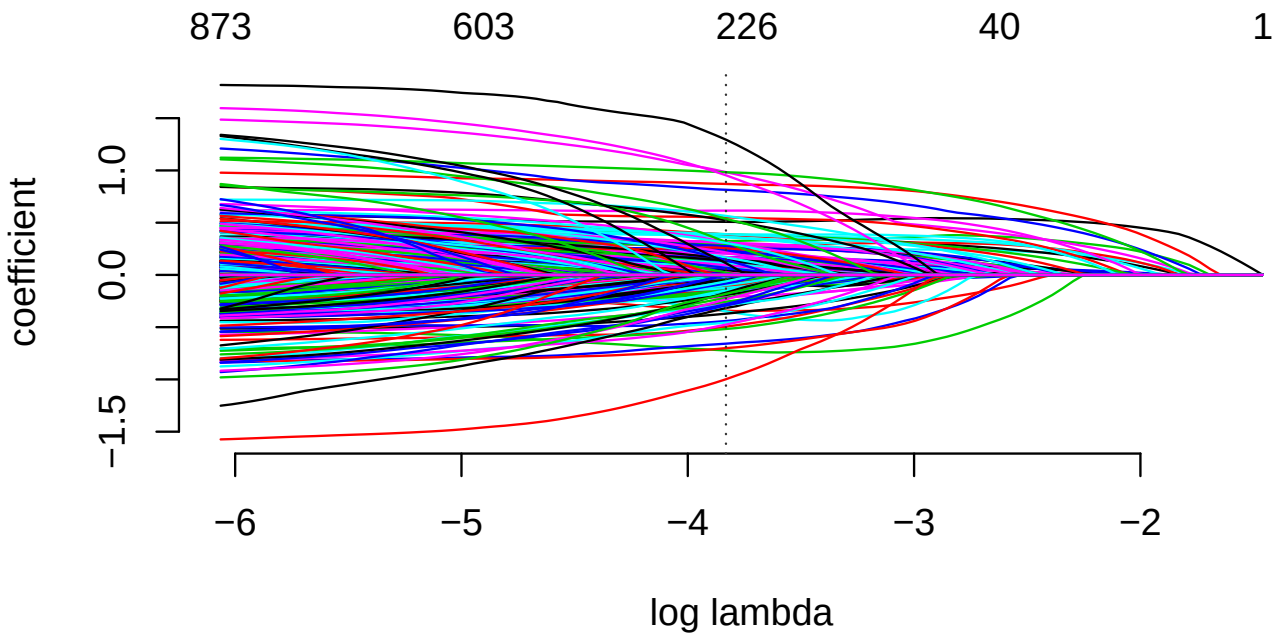
This is the corrected AIC, or AICc.

It also works nicely in logistic regression, or for any `glm`.

Notice that for big n/df, AICc $\approx$ AIC. So *always* use AICc.

gamlr uses AICc

It's marked on the path plot

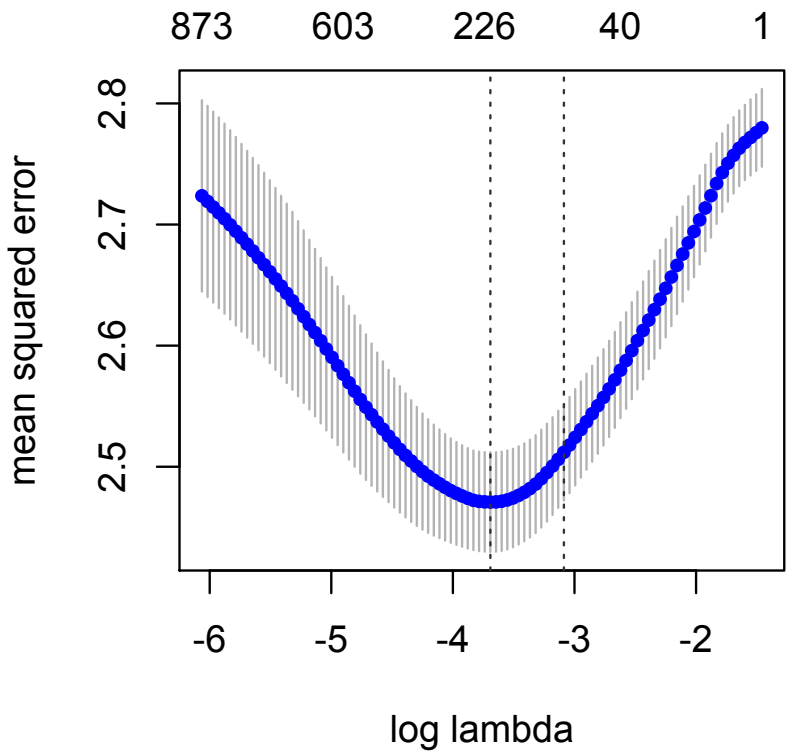


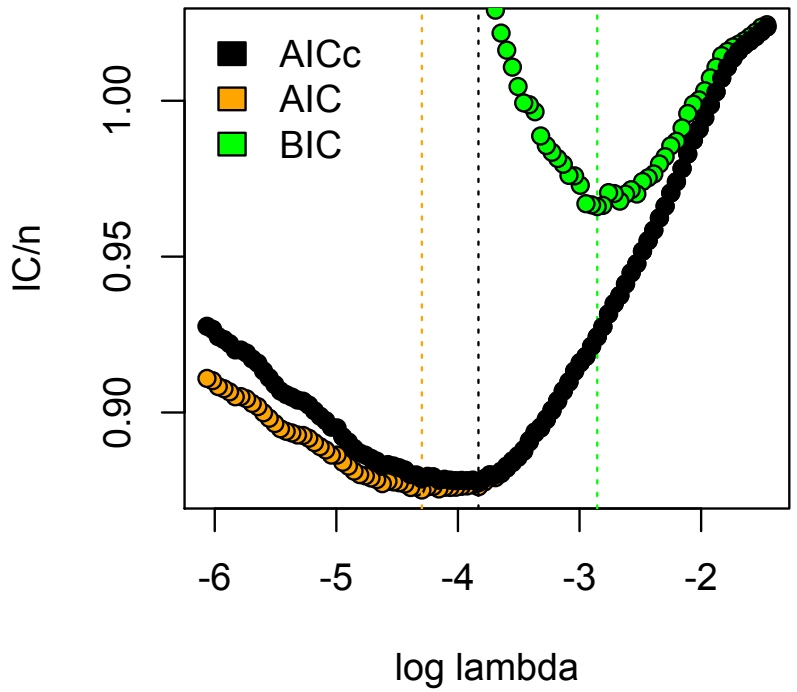
And it is the default for `coef.gamlr`

```
B <- coef(spender)[-1,]
B[c(which.min(B),which.max(B))]
```

```
    cursormania.com shopyourbargain.com
          -0.998143             1.294246
```

IC and CV on the Comscore Data

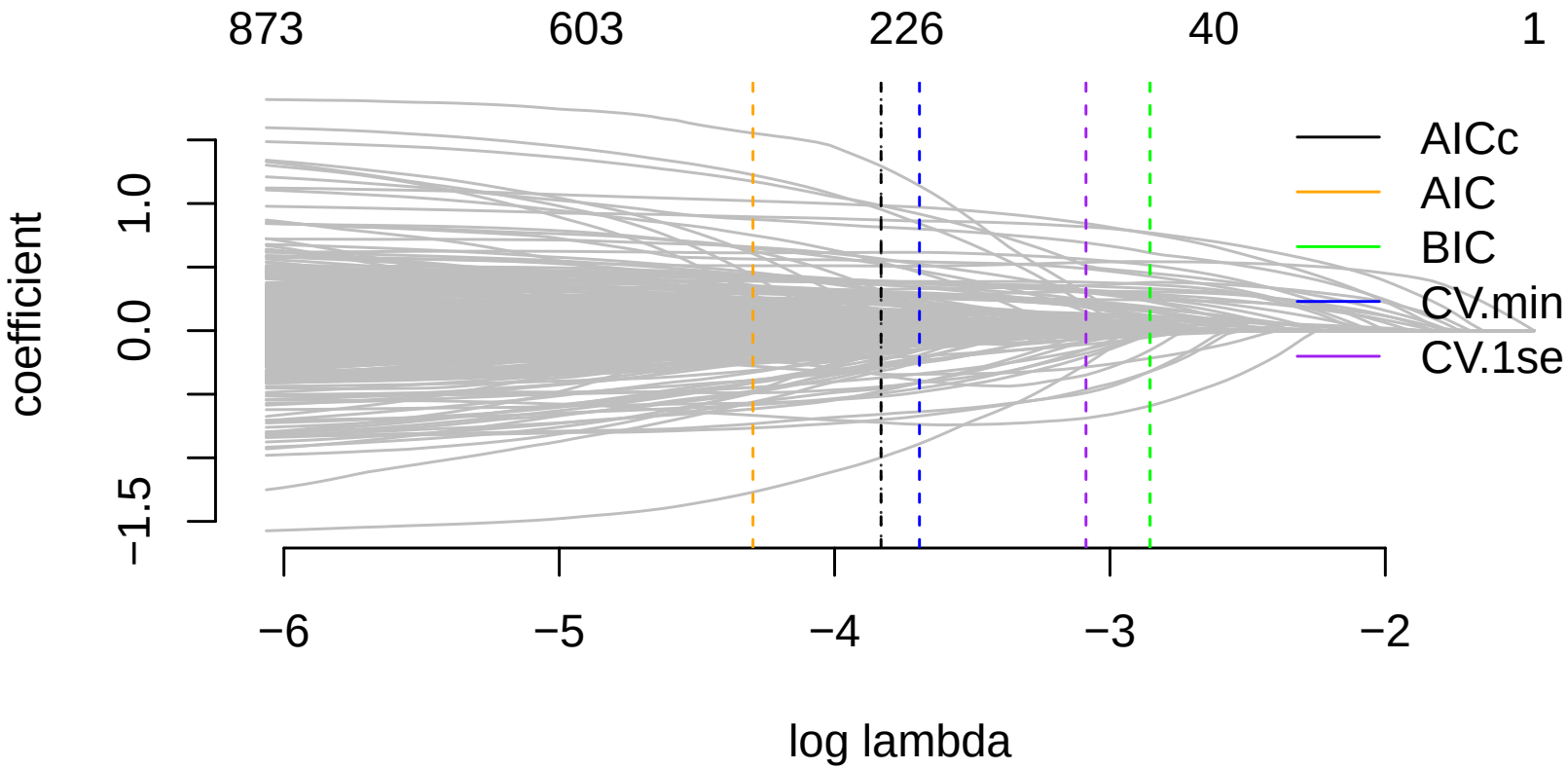




The take home message: AICc curve looks like CV curve.

In practice, BIC works more like the 1se CV rule.
But with big n it chooses too simple models (it underfits).

IC and CV on the Comscore Data



With all of these selection rules, you get a range of answers.

If you have time, do CV. But AICc is fast and stable.

If you are worried about false discovery, tend towards BIC/1se.

Causal Inference

We've used *prediction* as a basis for model building:

choose a fitting routine to do the best job forecasting y at new $\mathbf{x}$ drawn from the same distribution as data sample $\mathbf{X}$.

This exactly the question tackled by CV and AICc.

Now, we'll try to estimate the effect of a special covariate:

Treatment 'd', which we can change *independently* from $\mathbf{x}$.

That is: we want to know the causal *treatment effect* (TE).

For example,

- $d = 1$ if you get the drug, $d = 0$ for placebo (control).
- d is some *policy tool*, like interest rates or product price.

Observational Studies

Our treatment effect (TE) model looks like

$$\mathbb{E}[y|d, \mathbf{x}] = \alpha + d\gamma + \mathbf{x}'\boldsymbol{\beta}$$

and we'll want to interpret the estimated *treatment effect* $\hat{\gamma}$ as change in y when d moves independent of all other influencers.

This is easy to measure in a fully randomized experiment, because d is independent by design: we sample it randomly.

Without an experiment, we need to control for (i.e., remove from $\hat{\gamma}$) all influences on y which are correlated with d.

These other influencers are called confounders.
We control for them by including them in regression.

How does controlling for confounders work?

With $\mathbf{x}$ in the regression model, inference for γ is measured from the effect of *the bit of* $\mathbf{d}$ *that is not predictable by* $\mathbf{x}$.

e.g., say $d(\mathbf{x}) = \mathbf{x}'\boldsymbol{\tau} + \nu$, where ν is random noise (residual).

$$\begin{aligned} \text{Then:} \quad \mathbb{E}[y|\mathbf{x}, d] &= d\gamma + \mathbf{x}'\boldsymbol{\beta} \\ &= (\mathbf{x}'\boldsymbol{\tau} + \nu)\gamma + \mathbf{x}'\boldsymbol{\beta} \\ &= \nu\gamma + \mathbf{x}'(\gamma\boldsymbol{\tau} + \boldsymbol{\beta}) \approx \nu\hat{\gamma} + \mathbf{x}'\hat{\boldsymbol{\beta}} \end{aligned}$$

So $\hat{\gamma}$ is *identified* as the effect of ν, the independent part of d.

This type of controlling is simple with low-D $\mathbf{x}$: just fit the MLE regression and your standard errors on $\hat{\gamma}$ should be correct.
But this fails if you have too many controls.

Multicollinearity and the lasso

Can't we just throw everything in a lasso?

Not exactly.

Even if all possible influencers are in $\mathbf{x}$, the lasso won't always choose the right ones to remove confounding effects.

In the previous slide's example, we could have also written $\mathbf{x} = \boldsymbol{\varphi} d + \upsilon$, so that $\mathbf{x}$ is now a function of d.

$$\text{Then:} \quad \mathbb{E}[y|\mathbf{x}, d] = d\gamma + \mathbf{x}'\boldsymbol{\beta} = d(\gamma + \boldsymbol{\varphi}'\boldsymbol{\beta}) + \upsilon\boldsymbol{\beta}$$

Since the lasso makes you pay a price for every extra nonzero coefficient, it'll choose to just collapse the effects of $\mathbf{x}$ ($\boldsymbol{\beta}$) into $\hat{\gamma}$ (unless υ has a big enough effect to warrant the extra cost.)

Causal Lassos

Recall how we started today: we want to reduce dimension in the directions that are relevant to our decision making.

The problem is that $d = d(\mathbf{x}) + \nu$, and we want the effect of ν.

So estimate $\hat{d}(\mathbf{x})$ directly and include it in the regression!

Any left over effect of d will be attributable to $d - \hat{d}(\mathbf{x}) = \nu$.

$$\mathbb{E}[y|\mathbf{x}] = (\hat{d}(\mathbf{x}) + \nu)\gamma + \hat{d}(\mathbf{x})\delta + \mathbf{x}'\boldsymbol{\beta} = \nu\gamma + \hat{d}(\mathbf{x})(\gamma + \delta) + \mathbf{x}'\boldsymbol{\beta}$$

Controlling for $\hat{d}(\mathbf{x})$ in regression is equivalent
to estimating $\hat{\gamma}$ as the effect of ν: *the independent part of d*.

A Treatment Effects Lasso

Two stages:

1. Estimate $\hat{d}(\mathbf{x})$ with lasso regression of d on $\mathbf{x}$.
2. Do a lasso of y on $[d, \hat{d}(\mathbf{x}), \mathbf{x}]$, with $\hat{d}(\mathbf{x})$ *unpenalized*.

Including $\hat{d}$ unpenalized in [2] ensures that confounder effects on d have been removed: thus $\hat{\gamma}$ measures the effect of ν.

In [2], we can apply our usual AICc lasso to see what else in $\mathbf{x}$ effects y and, most importantly, if $\hat{\gamma} \neq 0$.

We've replaced causal estimation with two prediction problems. And prediction is something we're really good at, even in HD.

In the end, we're asking: is ν *useful* for predicting y?

Freakonomics: Abortion and Crime

Donahoe and Levitt (DL) argue a controversial thesis:

easier access to abortion
causes decreased crime.

Proposed mechanism holds that birth is postponed until the mother is more ready.

They assume stable family $\Rightarrow$ better upbringing $\Rightarrow$ less crime.

There's obviously no experiment here.
How have they controlled for confounders?

Crime $\sim$ Abortion regression

The treatment variable d is by-state abortion rate, and for response we look at $y =$ murder rate.

DL *control* for bunch of state-specific *confounders*: income, poverty, child tax credits, weapons laws, beer consumption...

They also include state effects (factor 's') and a time trend (numeric 't") to control for missed confounders.

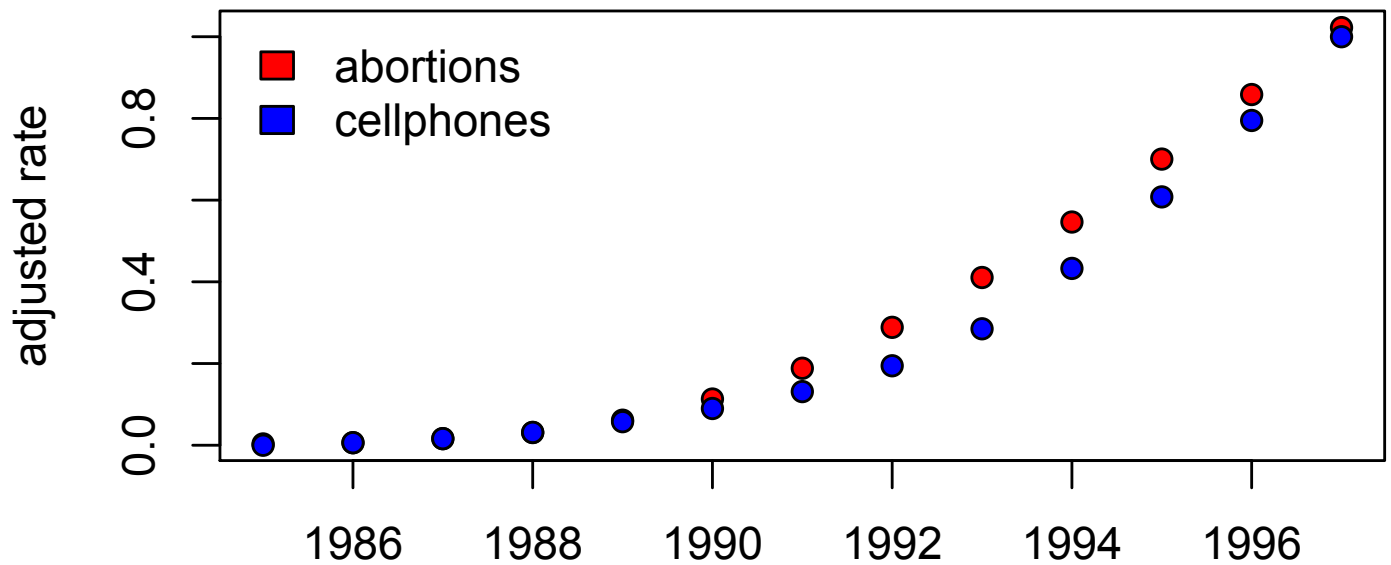
```
> orig = glm(y ~ d + t + s + ., data=controls)
> summary(orig)$coef['d',]
     Estimate    Std. Error       t value      Pr(>|t|)
-2.098119e-01  4.109177e-02 -5.105936e+00  4.505925e-07
```

Abortion has a very significant effect! Skeptical? You should be.

Alternative story: Cellphones and Murder

Technology has contributed to lower murder rates, and we'll add cellphone subscribers as a variable to control for tech progress.

e.g., Cellphones lead to faster ambulances, our jobs are gentile so we're less rough, more communication increases empathy, or we don't interact in-person because we don't have to.



Abortion and cellphones move together...

```
> tech = glm(y ~ phone + t + s + ., data=controls)
> summary(tech)$coef[c('phone'),]
     Estimate    Std. Error       t value      Pr(>|t|)
-3.662545e-01  6.779352e-02 -5.402500e+00  9.699480e-08
```

Cellphones have an even more significant effect!

It took me about 10 minutes to dream up a causal variable and grab data off of wikipedia.

What is happening is that murder decreased quadratically, and we have no controls that also moved this way.

How can we be sure the abortion effect is not just a stand-in for another cause that changed quadratically over the years?

To *control* for such confounders, we just add t^2 to the model.

We should also allow confounder effects to interact with each other (e.g., different `gun` effect for high `beer`) and with time.

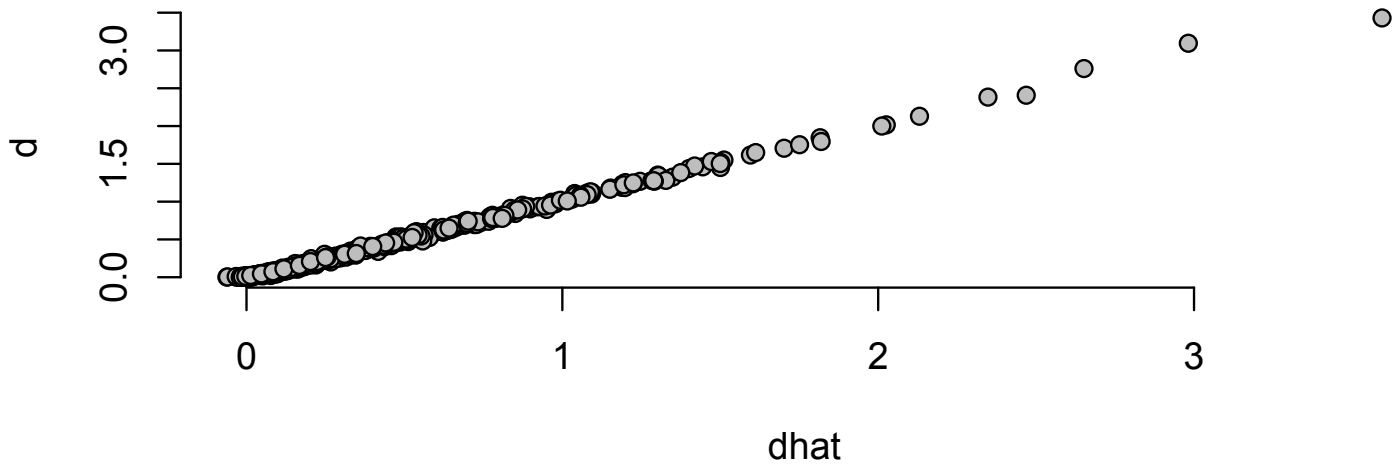
But now we've added so many variables that there is not enough data to say anything.

```
> dim(model.matrix(y ~ d + (s + .^2)*(t+t2), data=cntrls))
 [1] 624 280
```

We can't expect to fully control for every crazy story.
We'd like to select amongst HD controls those we really need.

$\Rightarrow$ Use the double lassos to first control, then estimate.

AICc lasso selects *d* as highly predicted by **x**.



There's almost no *independent* movement of abortion rates to measure as effecting crime (it's not much of an experiment).

Sure enough, if you include `dhat` in lasso regression then AICc says there is no residual effect for *d*.

```
## free=2 here leaves dhat unpenalized
> causal <- gamlr(cBind(d,dhat,x),y,free=2)
> coef(causal)["d",]
[1] 0
```