COM SCI M148, Lecture 5:
Regularization

Prof. Nathan Szymanski
nszymanski@seas.ucla.edu
Department of Materials Science

UCLA

Samueli
School of Engineering

For fun – from the NY Times

A College-by-College Guide to Which Majors Pay Off — or Don't




By Arfa Momin and Ron Lieber

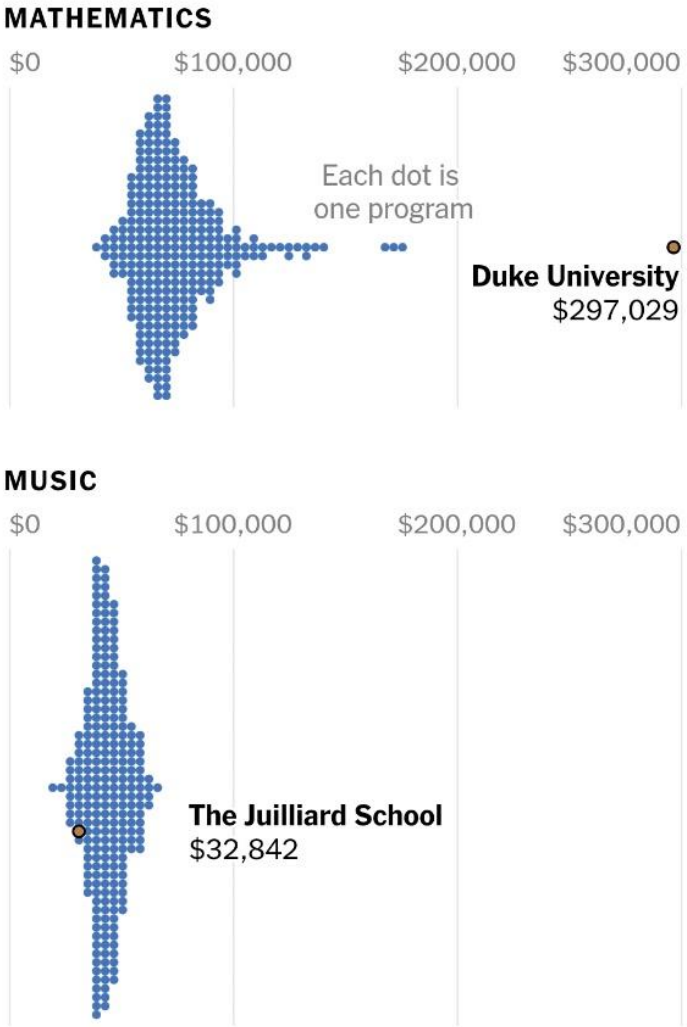
Arfa Momin finished her bachelor's degree in May. Ron Lieber is one-third finished with paying for his children's degrees.

Oct. 7, 2026

Note: Earnings represent the median income reported on I.R.S. tax records for students who graduated from a particular program in 2017–18 or 2018–19. A program is listed if there were more than 15 graduates who had received federal aid. Programs in Puerto Rico, Guam and the Northern Mariana Islands are excluded. Major classifications come from the National Center for Education Statistics. Source: The HEA Group.

All data represents 4 years after graduation

Here's how earnings for a handful of majors vary across most schools



The top of the earnings list is dominated by graduates of highly rejective schools — the Ivies and others, like M.I.T., Stanford and U.C.L.A. The majors that top the list are computer science, mathematics and electrical engineering.

Compare earnings for your own college list

Select a school

Pick one

Select a major

Pick one

UNIVERSITY	EARNINGS	MAJOR
All Institutions	$64,500	All Majors
University of California, Los Angeles	$216,722	Computer and Information Sciences, General

Recall: (Multiple) Linear Regression

$$\hat{y} = X\beta$$

Response vector:

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \dots \\ \hat{y}_n \end{bmatrix}$$

The values we are trying to predict

Each row is one sample

Recall: (Multiple) Linear Regression

$$\hat{y} = X\beta$$

Response vector:

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \dots \\ \hat{y}_n \end{bmatrix}$$

The values we are trying to predict

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

Feature matrix

x_{ij} represents observed input feature values

Row (i) = one sample

Column (j) = one feature

Each row is one sample

*Can include many **interaction terms** and **higher-order terms***

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_1 x_2 + \beta_3 x_1^2 \dots$$

I will generally use "feature" to refer to any one of these terms

Recall: (Multiple) Linear Regression

$$\hat{y} = X\beta$$

Response vector:

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \dots \\ \hat{y}_n \end{bmatrix}$$

The values we are trying to predict

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

Feature matrix

*Can include many **interaction terms** and **higher-order terms***

Each row is one sample

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_1 x_2 + \beta_3 x_1^2 \dots$$

Coefficient vector:

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{bmatrix}$$

Each term gets a weight

$$\beta = (X^T X)^{-1} X^T y$$

Can solve exactly!

Which terms to include? How many?

$$\hat{y} = X\beta$$

Response vector:

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \dots \\ \hat{y}_n \end{bmatrix}$$

The values we are trying to predict

Each row is one sample

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

Feature matrix

*Can include many **interaction terms** and **higher-order terms***

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_1 x_2 + \beta_3 x_1^2 \dots$$

Coefficient vector:

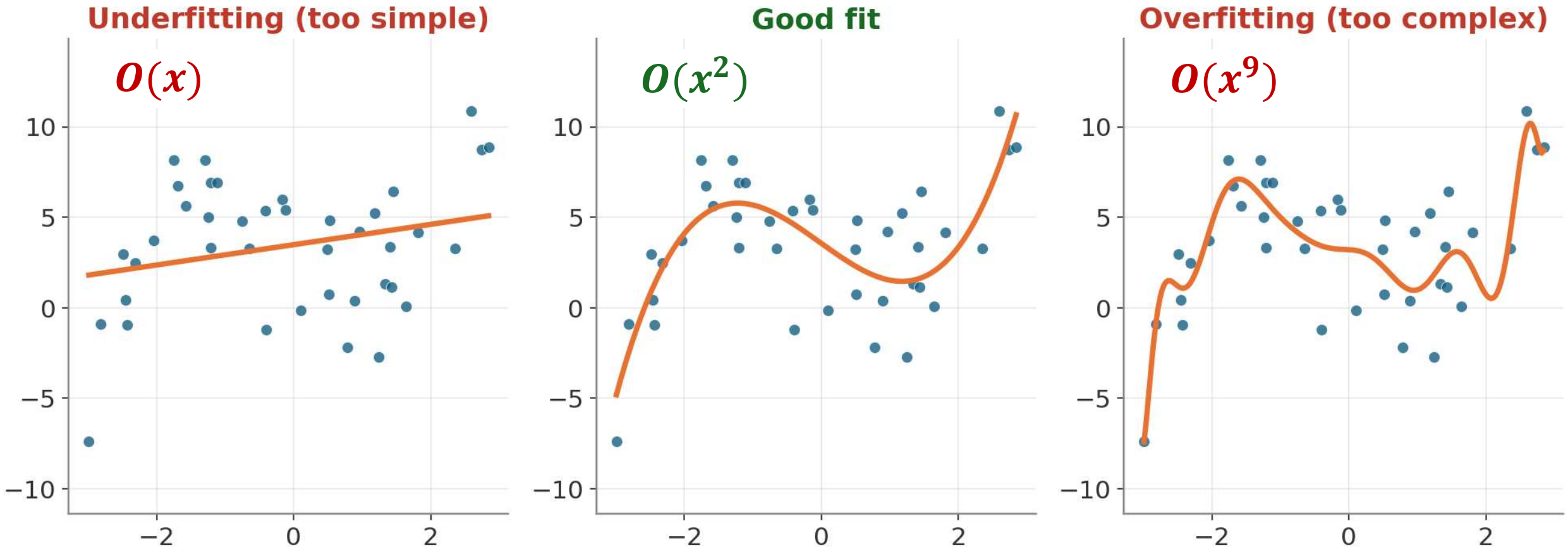
$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{bmatrix}$$

Each term gets a weight

$$\beta = (X^T X)^{-1} X^T y$$

Can solve exactly!

Recall: Overfitting *vs.* Underfitting in 1D



Too simple (low-degree) misses the trend; too complex (high-degree) memorizes the noise

Recall: Deciding model complexity

How to decide which terms to include in your multiple linear regression?

1) Use domain knowledge

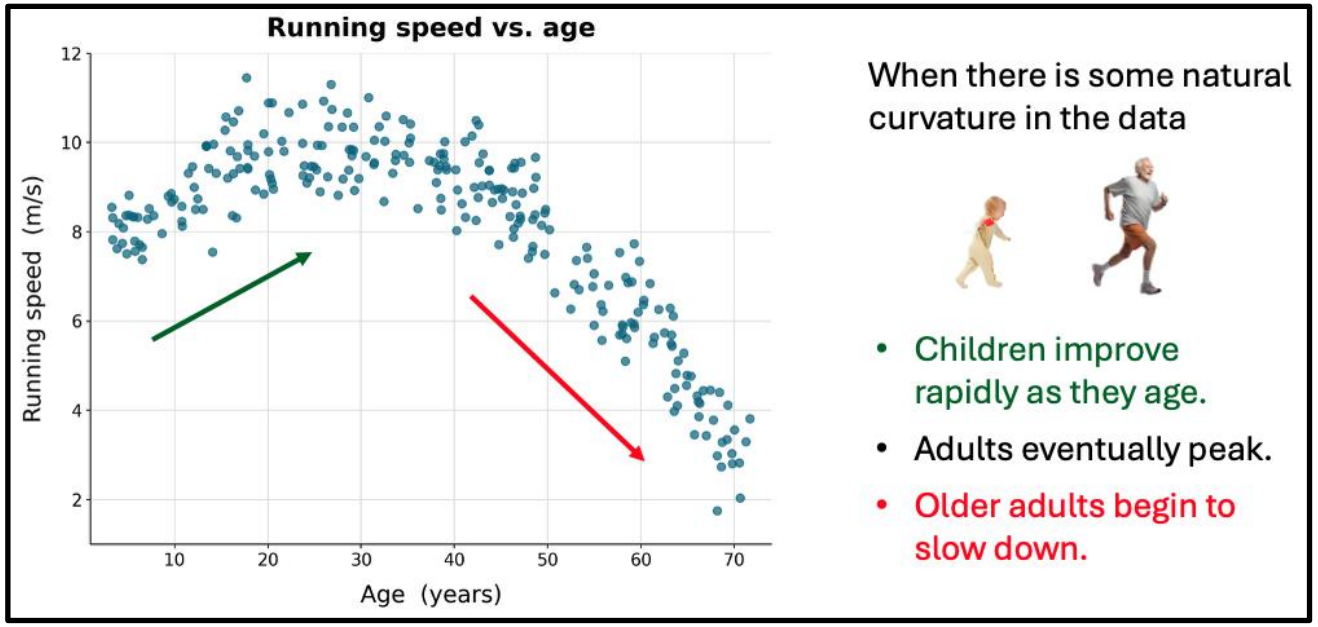
"These are the variables I think will be important" $(x_1, x_2, \ldots, x_n)$

"These pairs of variables will be correlated" $(x_1x_2, x_3x_4, \ldots)$

"These variables will have a non-linear effect" $(x_1^2, x_3^4, \ldots)$







Recall our running example from last time. Sometimes, it is clear which terms you should include.



But often, it is not… What if we don't know this in advance?

Recall: Deciding model complexity

How to decide which terms to include in your multiple linear regression?

1) **Use domain knowledge**

   "These are the variables I think will be important" $(x_1, x_2, \ldots, x_n)$

   "These pairs of variables will be correlated" $(x_1x_2, x_3x_4, \ldots)$

   "These variables will have a non-linear effect" $(x_1^2, x_3^4, \ldots)$



2) **We could try adding terms and evaluate on a hold-out test set**

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2$$

$$\hat{y} = \beta_0 + \beta_2 x_1 x_2^2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2$$

Quickly becomes intractable when we have lots of variables!!!

Today: Regularization

With **regularization**, we can include many candidate terms and let the model determine which ones ***are important.***

To generalize well to *new* data

Remember that minimizing error on the training set will *always* favor increased model complexity…
Regularization fights against this

2) We could try adding terms and evaluate on a hold-out test set

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2$$

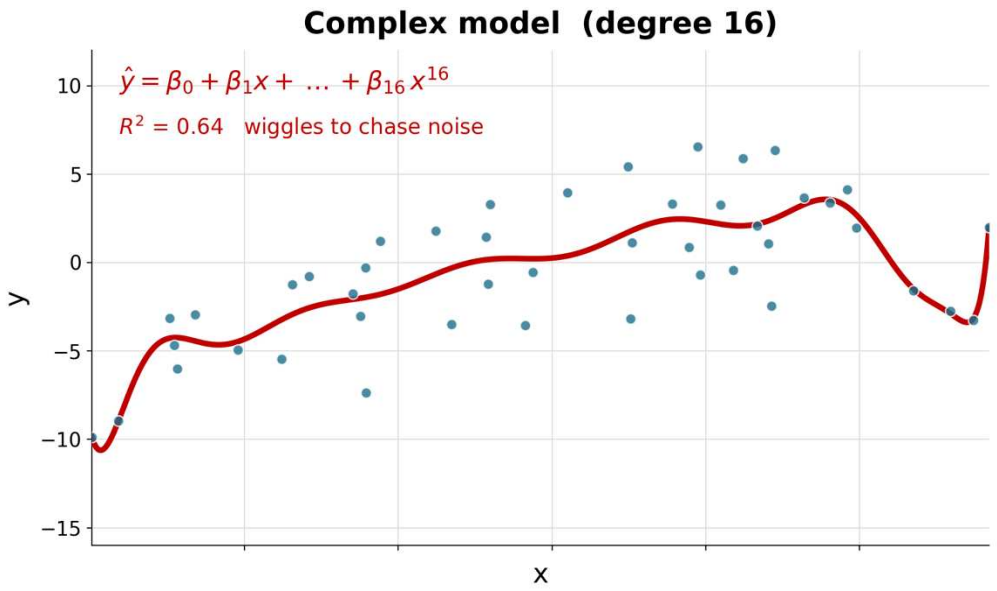
$$\hat{y} = \beta_0 + \beta_2 x_1 x_2^2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2$$

Quickly becomes intractable when we have lots of variables!!!

Illustrating regularization

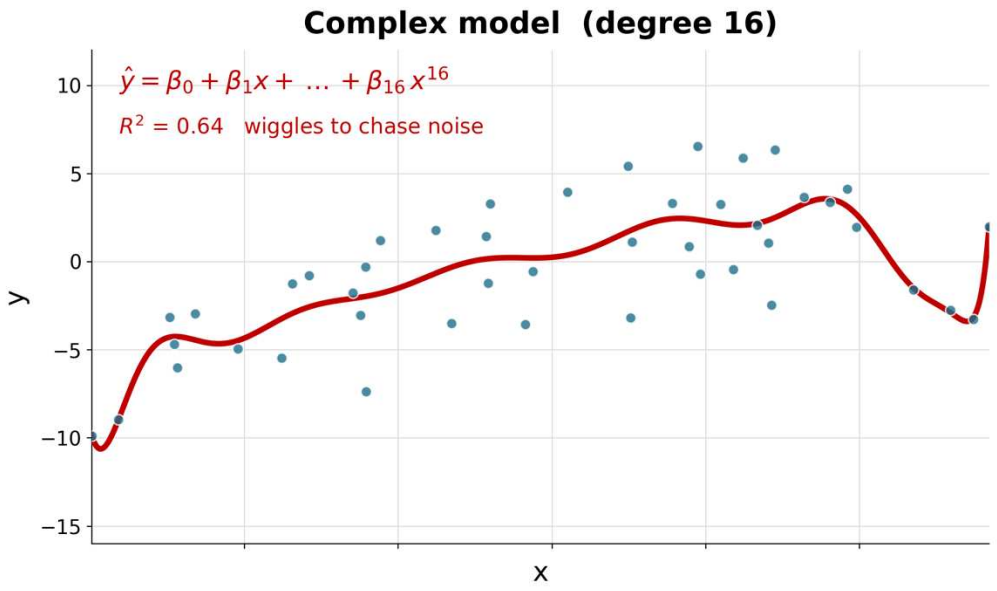


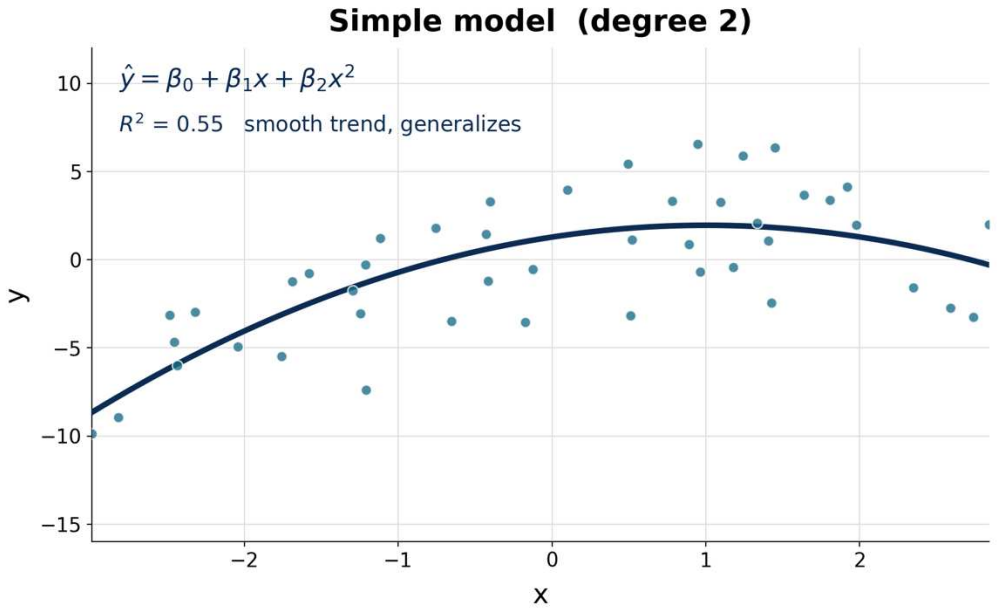
Suppose we fit a highly flexible model to a small, noisy dataset:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{16} x^{16}$$

→ Overfit to the noise

Illustrating regularization





Suppose we fit a highly flexible model to a small, noisy dataset:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{16} x^{16}$$

→ Overfit to the noise

What if we instead fit a simpler model?

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2$$

→ **Better captures the general trend**

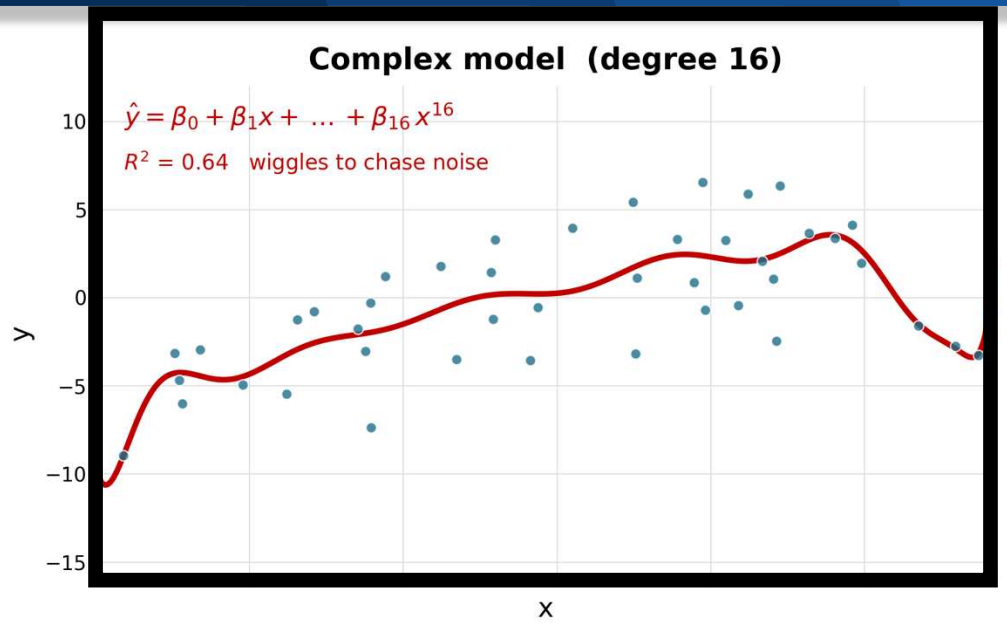
This could be rewritten as:

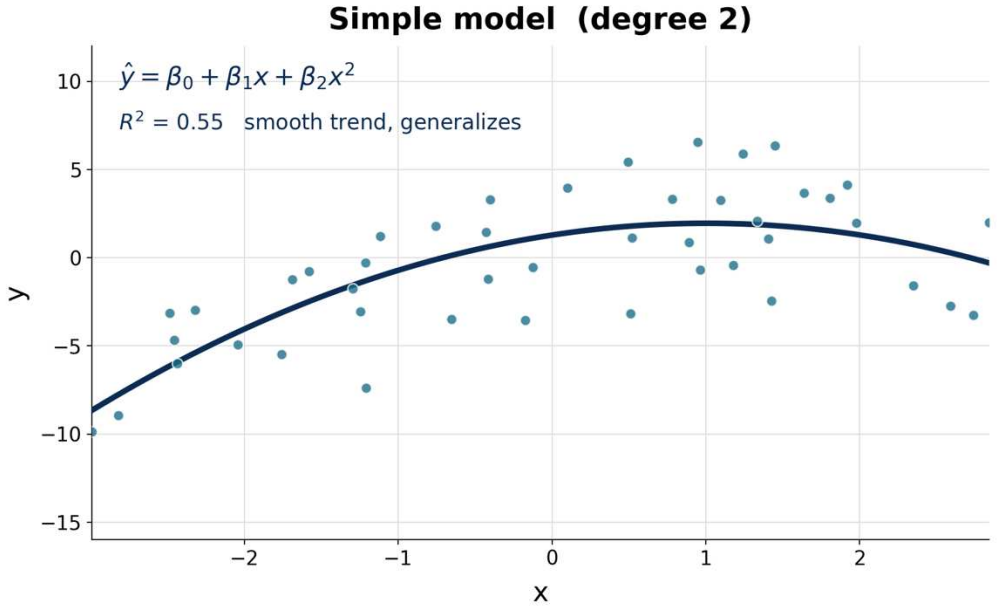
$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{16} x^{16}$$

Where $\underbrace{\beta_3 \approx \beta_4 \approx \cdots \approx \beta_{16} \approx 0}$

Can we learn which coefficients should be zero?

Illustrating regularization





But the model will *always* choose $\beta_i \neq 0$ if it leads to reduced error on the training set…

Even though it might raise error on the test set

Unless…We can somehow ***penalize*** $\beta_i \neq 0$ so the model uses coefficients only if they are absolutely necessary → **regularization**

This could be rewritten as:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{16} x^{16}$$

Where $\beta_3 \approx \beta_4 \approx \cdots \approx \beta_{16} \approx 0$

Can we learn which coefficients should be zero?

Regularization tries to keep β_i small

…By penalizing model complexity in the loss – not just plain old MSE anymore!

Modifying the loss function

Recall that we derived $\beta = (X^T X)^{-1} X^T y$ **by minimizing the MSE loss:**

$\mathcal{L}$ **indirectly depends on** β_i

$e_i = y_i - \hat{y}_i(\beta_i)$ ← *where* $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} |e_i|^2$

But the loss does not "care" if β_i **is big or small.**
It will just choose whatever β_i **minimizes the MSE.**

Can we modify the loss make it care?

Modifying the loss function

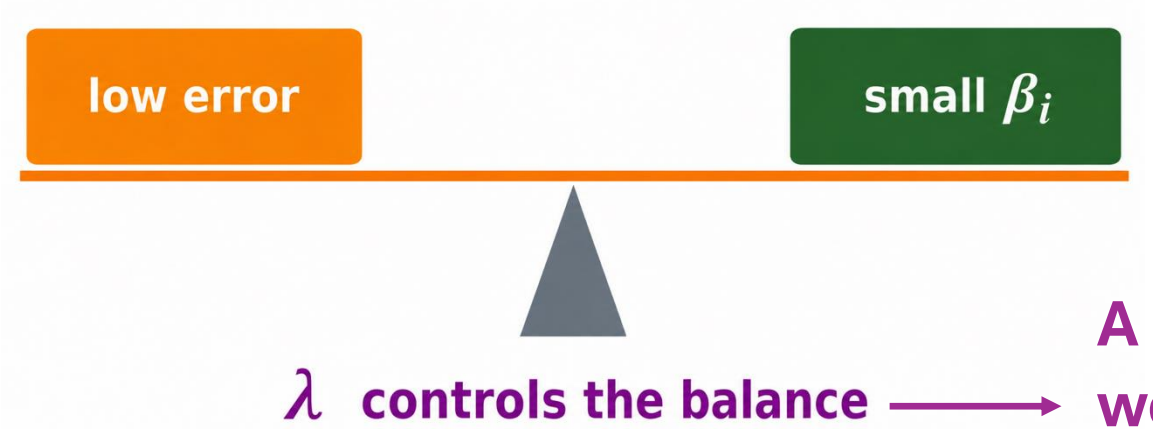
Recall that we derived $\beta = (X^T X)^{-1} X^T y$ by minimizing the MSE loss:

$\mathcal{L}$ indirectly depends on β_i

$e_i = y_i - \hat{y}_i(\beta_i)$ ← where $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} |e_i|^2$

How about we instead minimize: $\mathrm{MSE}(\beta_i) + \lambda R(\beta_i)$ → **A term that penalizes large β_i**

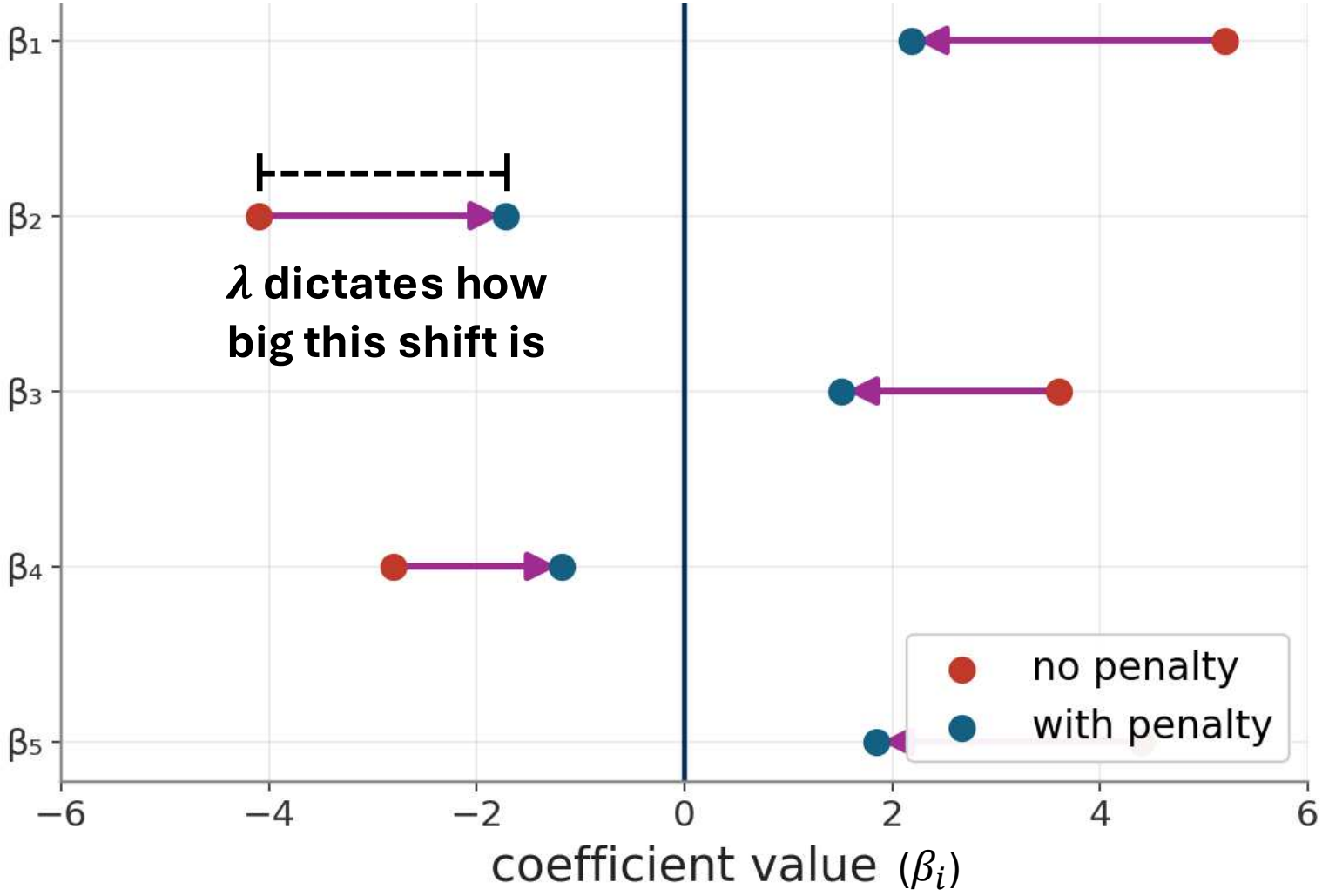
By minimizing the sum, we balance **fit** *vs.* **simplicity**



λ **controls the balance** → **A hyperparameter we choose**

Regularization shrinks β_i toward zero

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5$$



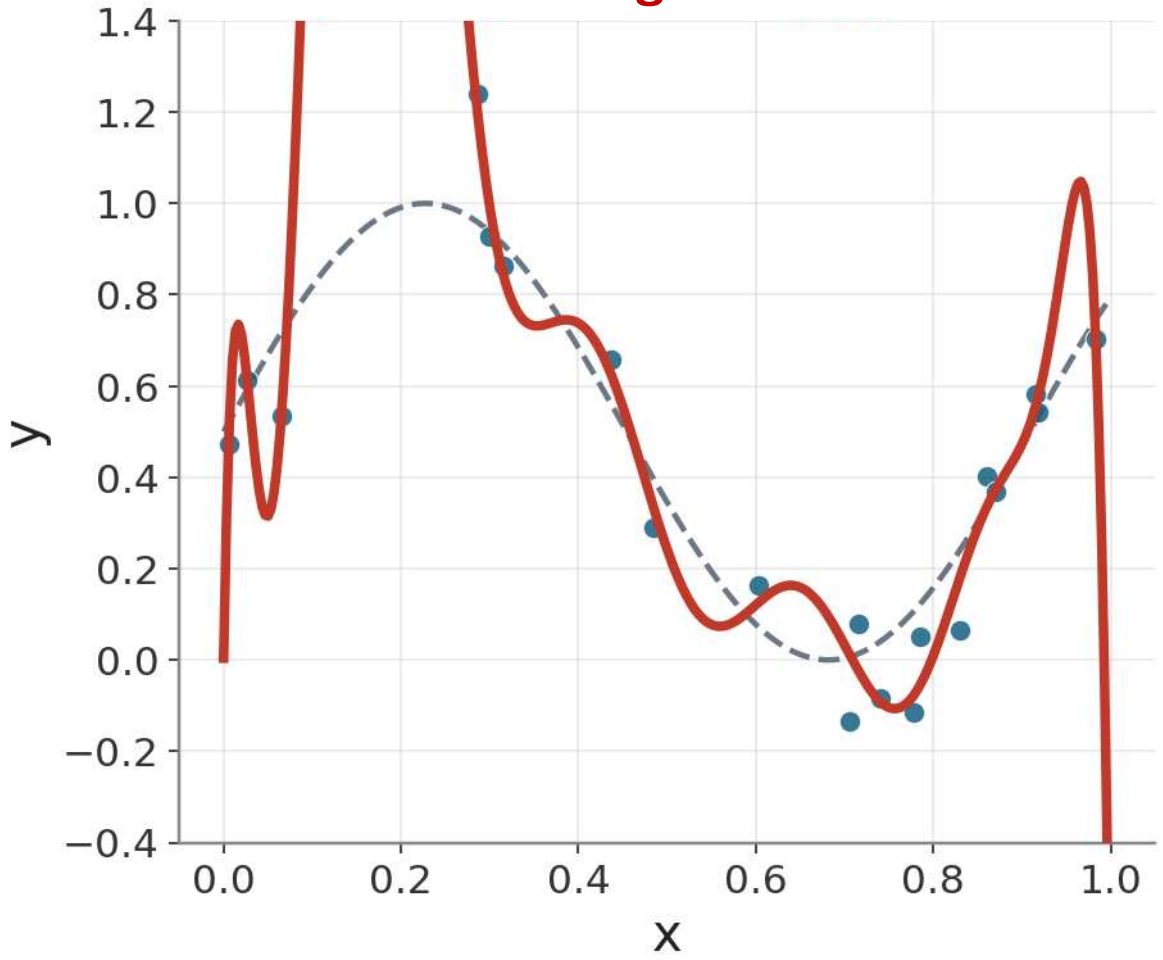
A ***hyper*parameter** is a parameter *we choose* before training the model, rather than learned from the data

The hyperparameter λ dictates how small we make β_i

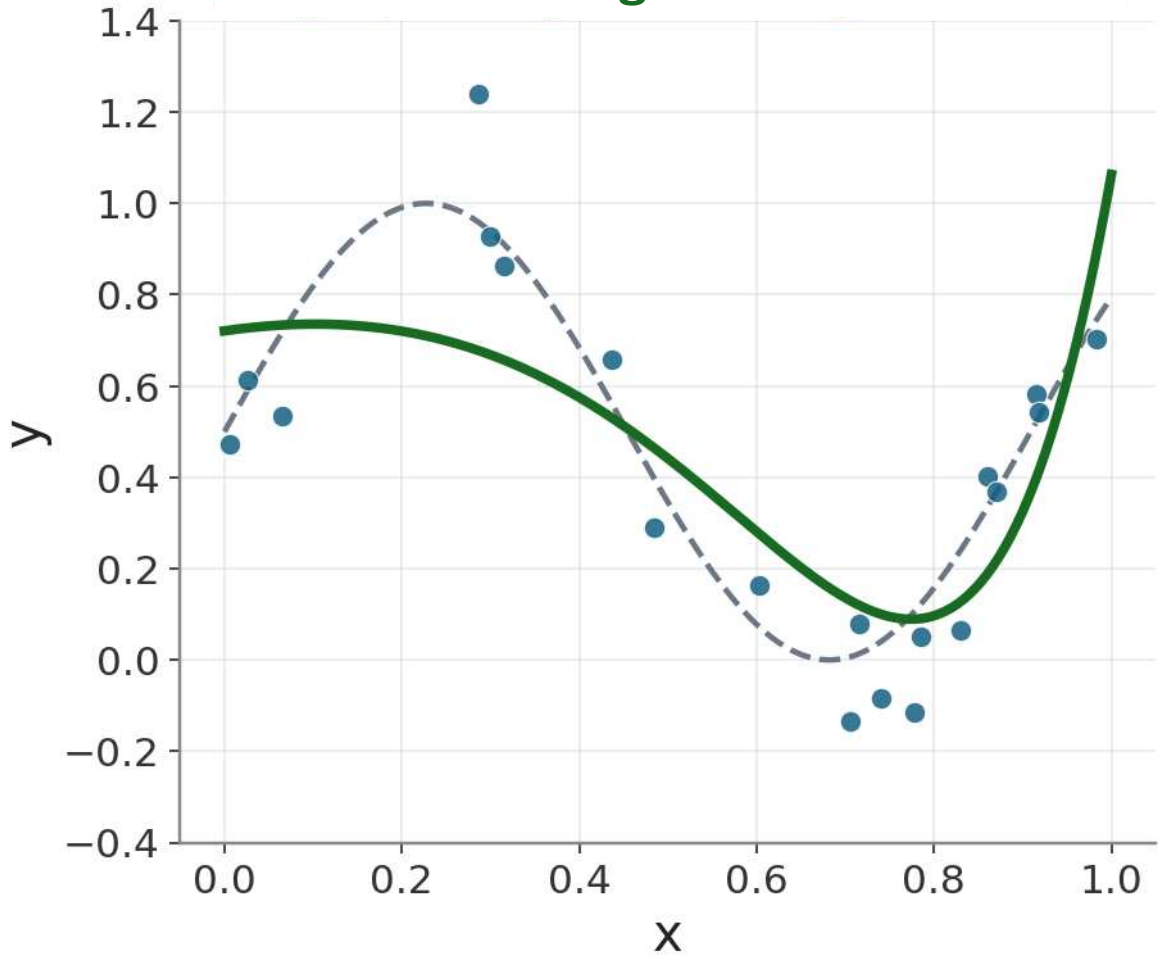
We'll discuss how to choose λ later…

The end-result is a smoother, simpler fit!

Before regularization



After regularization



Same degree-12 starting model. Regularization turns a “wiggly” curve into a smooth one

How to define the regularization term?

Remember:
1) We **minimize error**
2) We want **small $\boldsymbol{\beta_i}$**

$$\mathcal{L} = \mathbf{MSE}(\boldsymbol{\beta_i}) + \boldsymbol{\lambda} \boldsymbol{R}(\boldsymbol{\beta_i})$$

L_2-norm

L_1-norm

$$\boldsymbol{R}(\boldsymbol{\beta_i}) = \sum_i \boldsymbol{\beta_i^2}$$

Ridge regression

$$\boldsymbol{R}(\boldsymbol{\beta_i}) = \sum_i |\boldsymbol{\beta_i}|$$

Lasso regression

Any questions before we hop into the details?

Ridge Regression (L_2 Regularization)

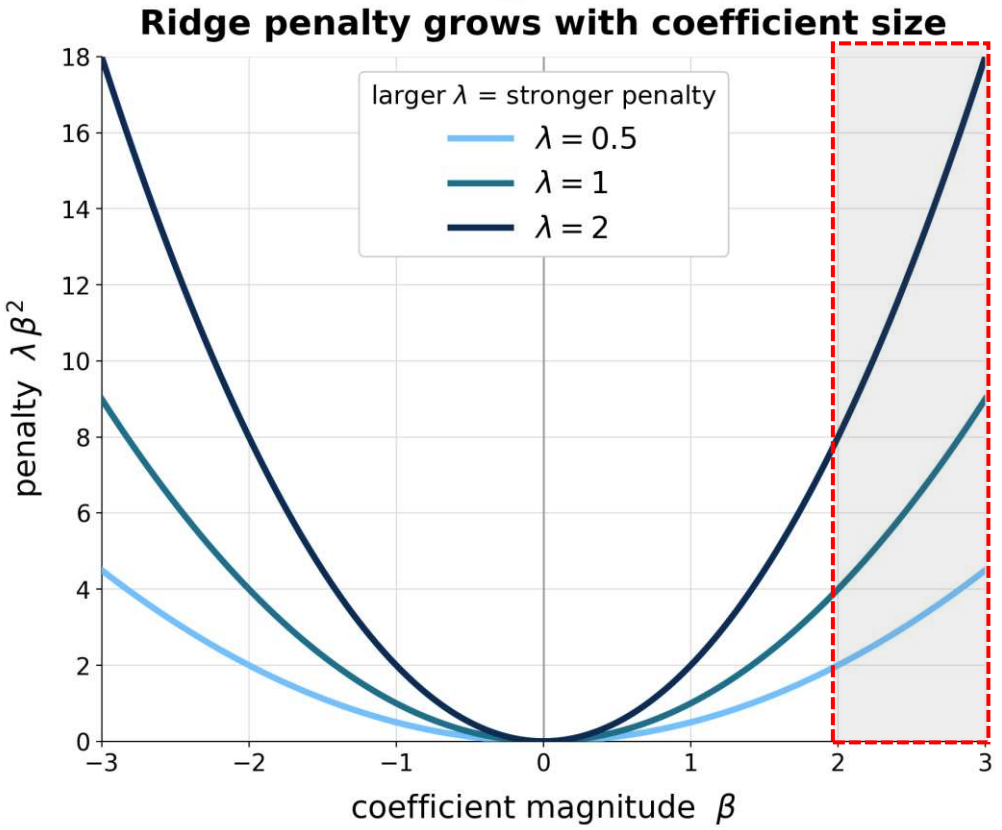
$$R(\beta_i) = \sum_i \beta_i^2$$

Ridge regression: Penalize the sum of squares

Remember:
$\hat{y}_i$ depends on β_i
so we can't just
make all $\beta_i = 0$

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|^2 + \lambda \sum_{i=1}^{n} \beta_i^2$$

- λ controls how much penalty is imposed
- Because it's the **square** of β_i, extra large coefficients are doubly penalized!

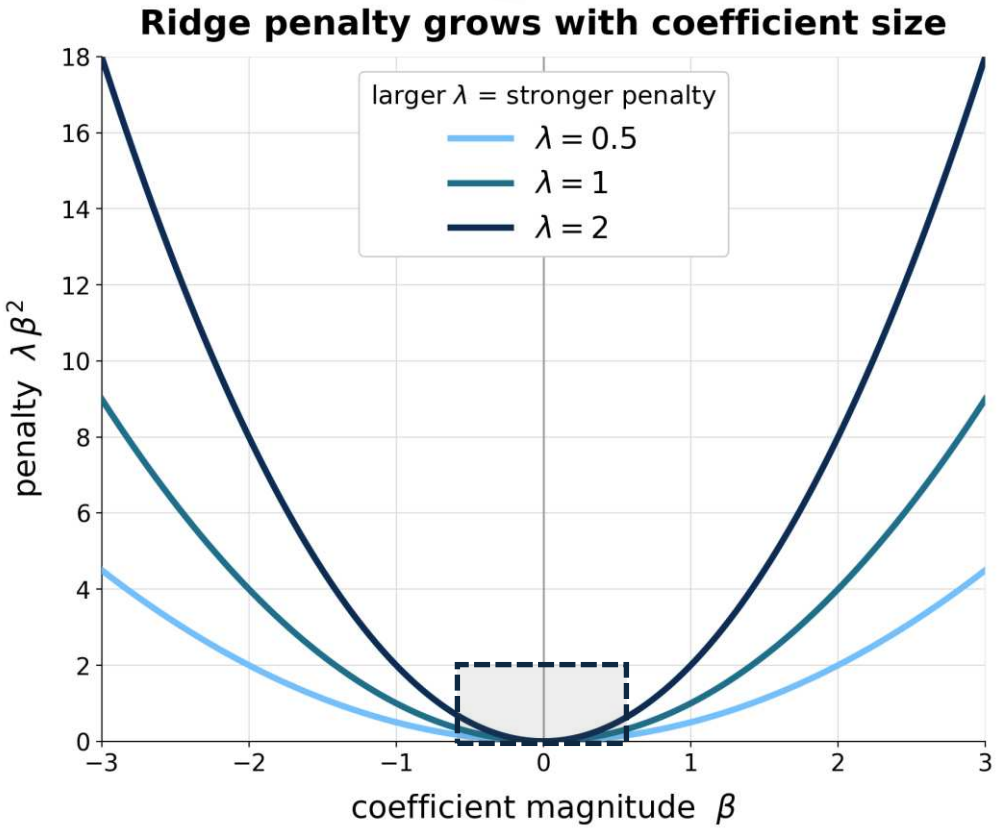


Ridge regression: Penalize the sum of squares

Remember:
$\hat{y}_i$ depends on β_i
so we can't just
make all $\beta_i = 0$

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|^2 + \lambda \sum_{i=1}^{n} \beta_i^2$$

- λ controls how much penalty is imposed
- Because it's the **square** of β_i, extra large coefficients are doubly penalized!
- Notice the derivative is small near zero, so those coefficients are not penalized much **(ridge rarely sets $\beta_i = 0$)**
- On that note…the **differentiability of β_i^2** allows us to solve this exactly ☺



Ridge regression: Derivation

Loss written in matrix form: $\mathcal{L} = \|y - X\beta\|^2 + \lambda\|\beta\|^2$ $\quad \|\beta\|^2 = \beta^T\beta = \sum_{i=1}^{n} \beta_i^2$

Rewrite: $(y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta = y^Ty - y^TX\beta - \beta^TX^Ty + \beta^TX^TX\beta + \lambda\beta^T\beta$

This is the same math as last time, just now with an extra term

Ridge regression: Derivation

Loss written in matrix form: $\mathcal{L} = \|y - X\beta\|^2 + \lambda\|\beta\|^2$ $\quad \|\beta\|^2 = \beta^T\beta = \sum_{i=1}^{n} \beta_i^2$

Rewrite: $(y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta = y^Ty - y^TX\beta - \beta^TX^Ty + \beta^TX^TX\beta + \lambda\beta^T\beta$

Minimize: $\frac{\partial \mathcal{L}}{\partial \beta} = \frac{\partial}{\partial \beta}(y^Ty - 2\beta^TX^Ty + \beta^TX^TX\beta + \lambda\beta^T\beta) = -2X^Ty + 2X^TX\beta + 2\lambda\beta = 0$

Ridge regression: Derivation

Loss written in matrix form: $\mathcal{L} = \|y - X\beta\|^2 + \lambda\|\beta\|^2$ $\quad \|\beta\|^2 = \beta^T\beta = \sum_{i=1}^{n}\beta_i^2$

Rewrite: $(y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta = y^Ty - y^TX\beta - \beta^TX^Ty + \beta^TX^TX\beta + \lambda\beta^T\beta$

Minimize: $\frac{\partial\mathcal{L}}{\partial\beta} = \frac{\partial}{\partial\beta}(y^Ty - 2\beta^TX^Ty + \beta^TX^TX\beta + \lambda\beta^T\beta) = -2X^Ty + 2X^TX\beta + 2\lambda\beta = 0$

Solve for $\boldsymbol{\beta}$ by rearranging : $(X^TX + \lambda I)\mathbf{2\beta} = 2X^Ty$

Cancel the 2's and multiple by the inverse of the left term

$$\boldsymbol{\beta = (X^TX + \lambda I)^{-1}X^Ty}$$

Basically, just add λ to the diagonal of X^TX

Side note:
I is the identity matrix

2x2: $\lambda I = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}$

Ridge regression: Derivation

So…If we are given:

- Observed inputs (X)
- Observed outputs (y)

And we choose λ

Then we can solve for the regularized coefficients!

$$\boldsymbol{\beta} = \left(\boldsymbol{X^T X}\right)^{-1} \boldsymbol{X^T y}$$

Our usual equation

$$\boldsymbol{\beta} = \left(\boldsymbol{X^T X} + \boldsymbol{\lambda I}\right)^{-1} \boldsymbol{X^T y}$$

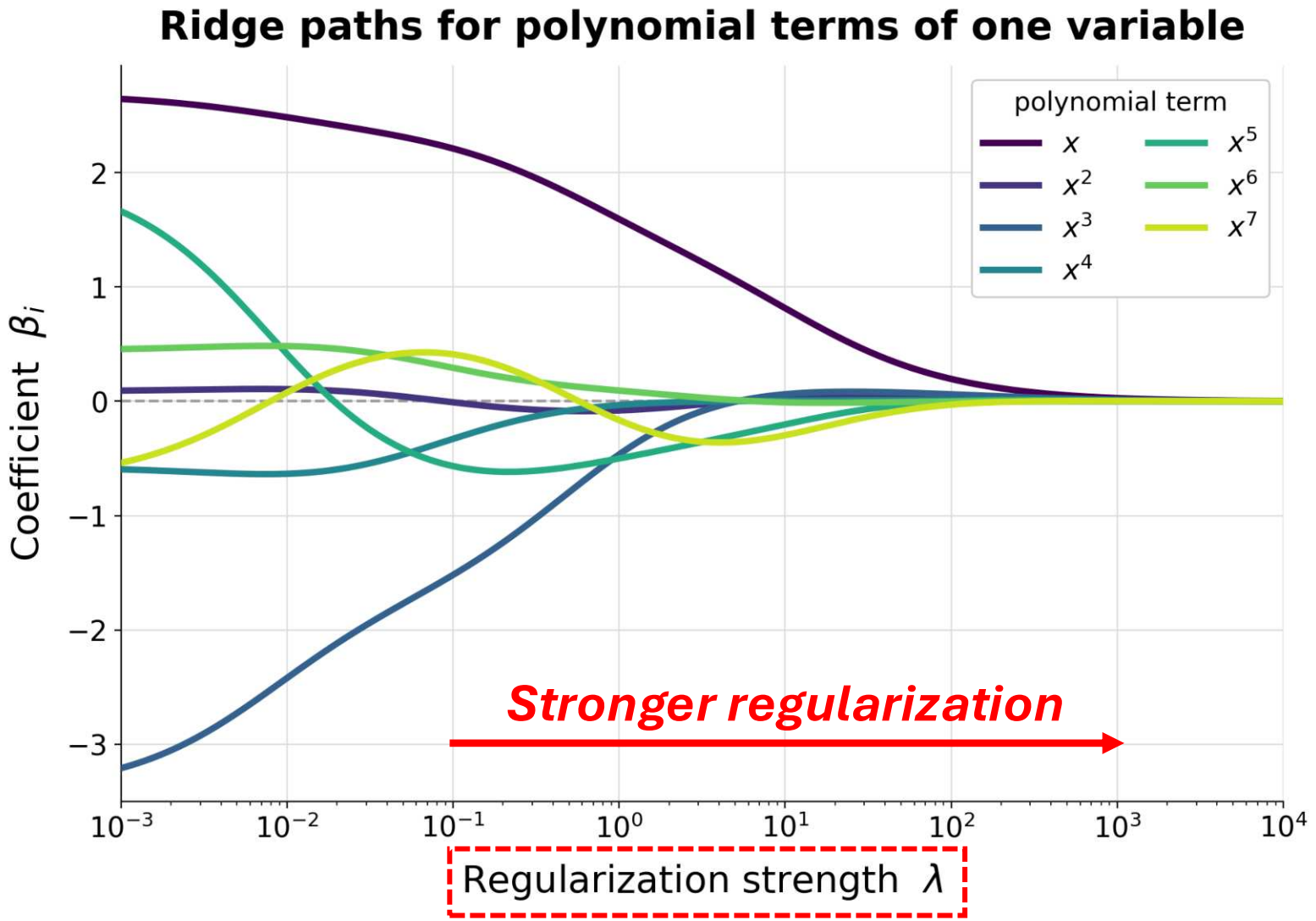
Basically, just add λ to the diagonal of $X^T X$

Looks very similar to the Normal Equation, but with an **extra term!**

Side note:
I is the identity matrix

2x2: $\lambda I = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}$

Ridge coefficients (usually) shrink but *never* hit zero



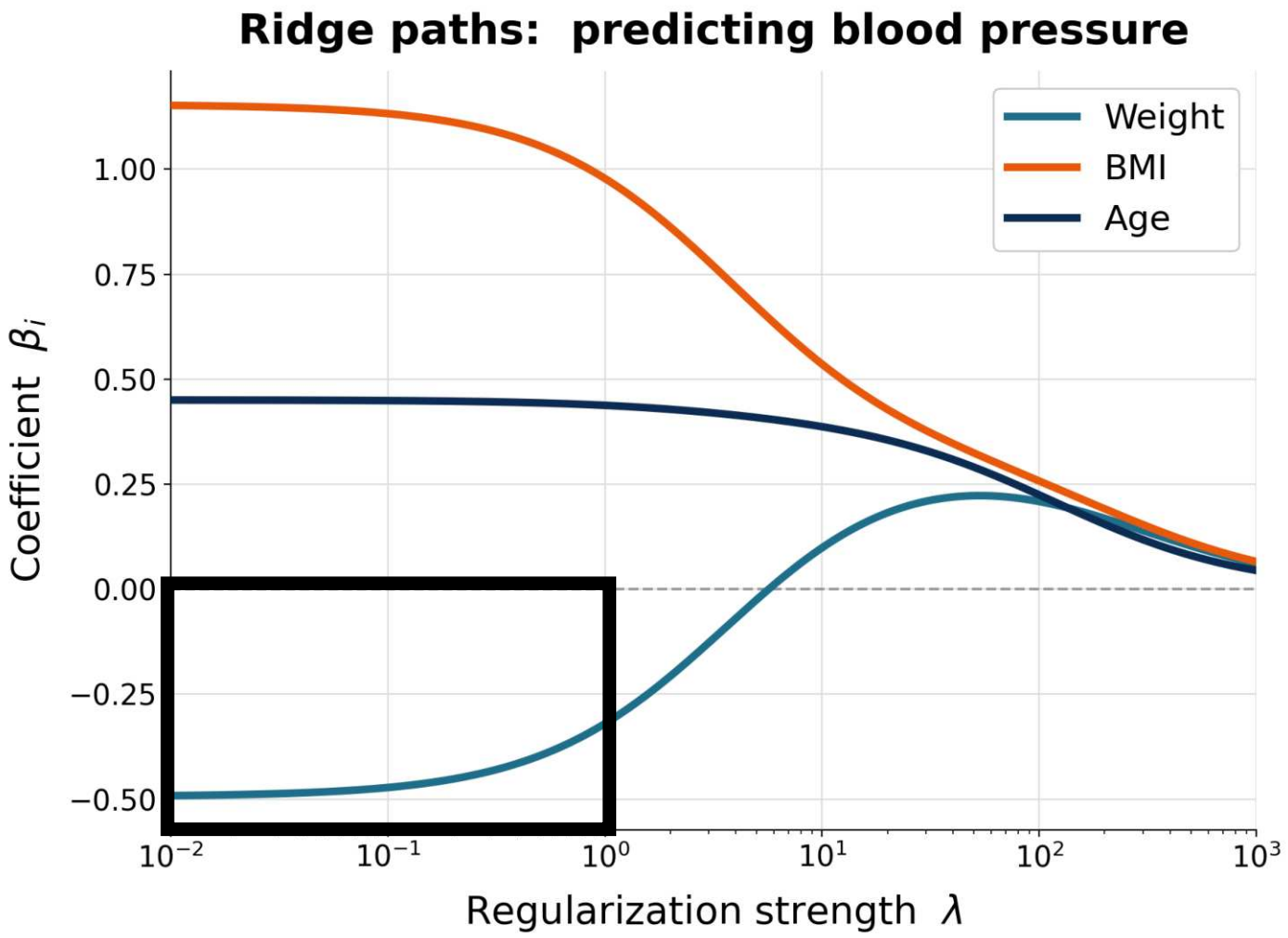
Consider a model with many terms depending on x^n:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5 + \beta_6 x^6 + \beta_7 x^7$$

$$\mathcal{L} = \|y - X\beta\|^2 + \lambda\|\beta\|^2$$

As we increase λ, *most* coefficients shrink...
But not all! The changes are non-monotonic due to "tradeoffs" between β_i

Tradeoffs between variables



Suppose for people of similar height, we are predicting blood pressure from **weight, BMI**, and age

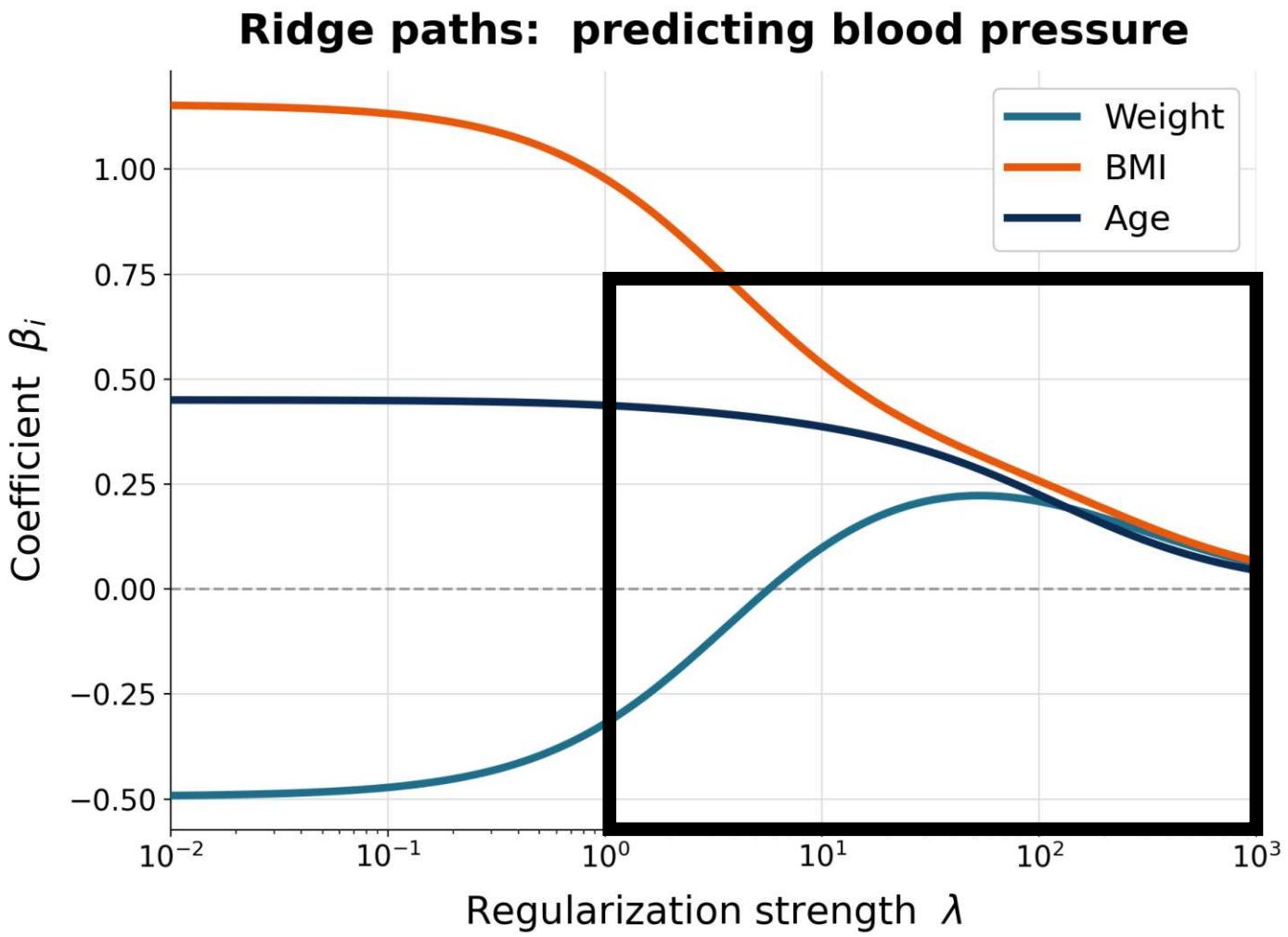
$$\boldsymbol{\beta_0} + \underbrace{\boldsymbol{\beta_1 x_{\text{weight}}} + \boldsymbol{\beta_2 x_{\text{BMI}}}} + \boldsymbol{\beta_3 x_{\text{age}}}$$

These two will be partially collinear

Without regularization (at small λ), notice that BMI gets a large β_1 but **weight** gets an (unphysical)* **negative $\boldsymbol{\beta_2}$** as it "fits the noise"

*Weight and blood pressure should **positively** correlate

Tradeoffs between variables



Suppose for people of similar height, we are predicting blood pressure from **weight, BMI**, and age

$$\beta_0 + \underbrace{\beta_1 x_{\text{weight}} + \beta_2 x_{\text{BMI}}} + \beta_3 x_{\text{age}}$$

These two will be partially collinear

***With* regularization** (at large λ), notice the **coefficients get more evenly distributed* and β_1 becomes positive**

*What if we want to force the model to choose one feature over another? (*e.g.*, drop BMI)

Lasso Regression (L1 Regularization)

But first, any questions on Ridge? We'll come back to it…

How to define the regularization term?

Remember:
1) We **minimize error**
2) We want **small β_i**

$$\mathcal{L} = \mathbf{MSE}(\beta_i) + \lambda R(\beta_i)$$

L_2-norm

L_1-norm

$$R(\beta_i) = \sum_i \beta_i^2$$

Ridge regression

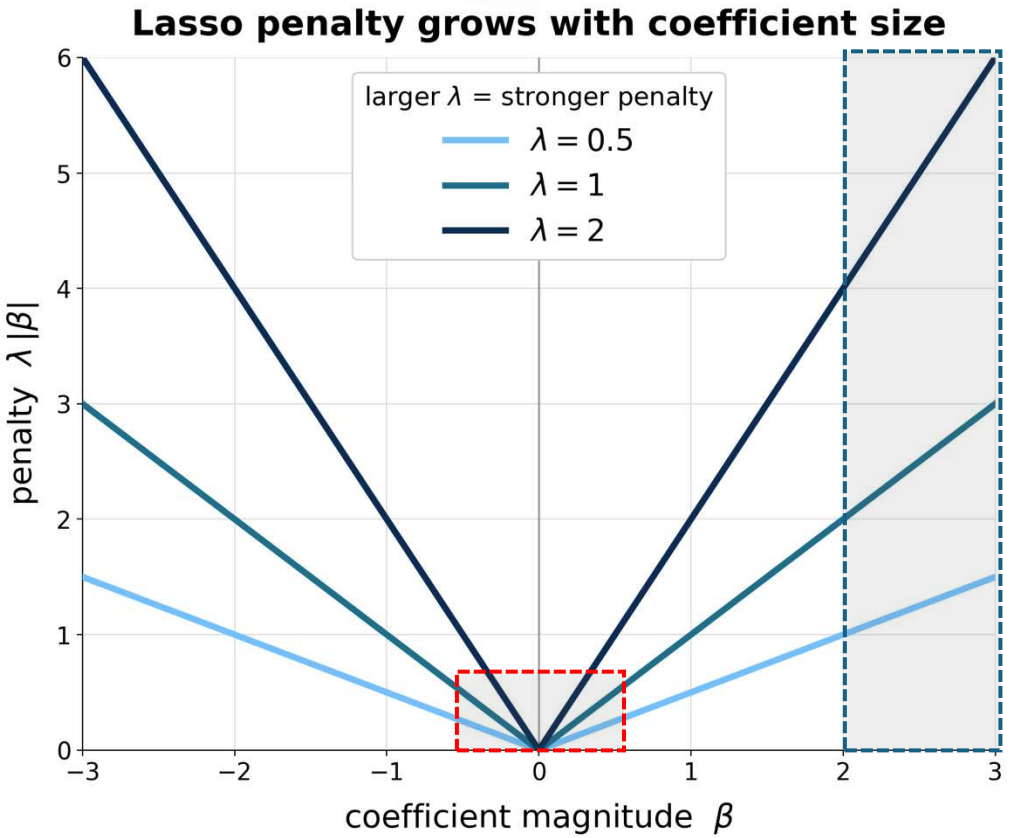
$$R(\beta_i) = \sum_i |\beta_i|$$

Lasso regression

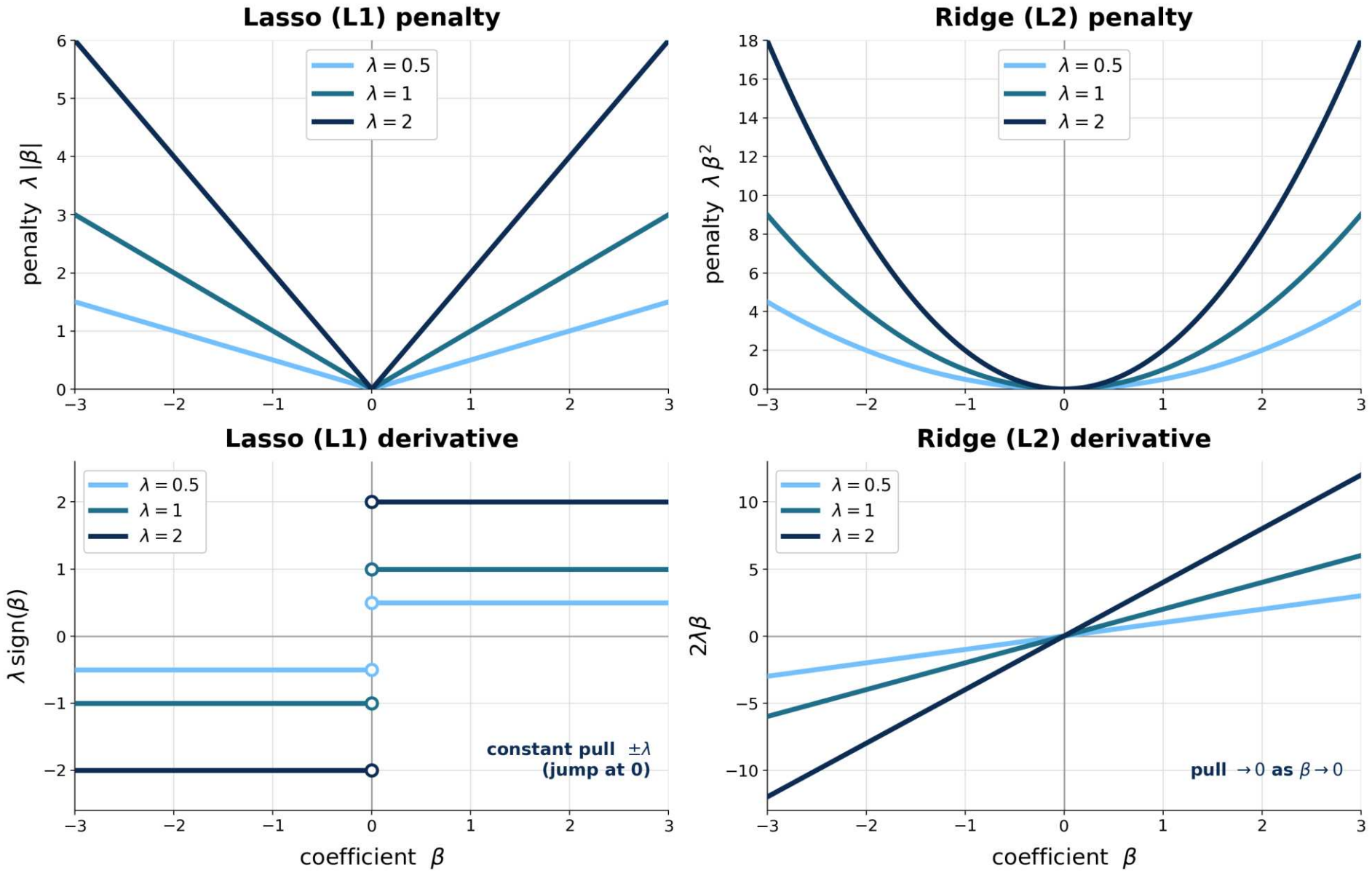
Lasso regression: Penalize absolute values

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 + \lambda\sum_{i=1}^{n}|\beta_i|$$

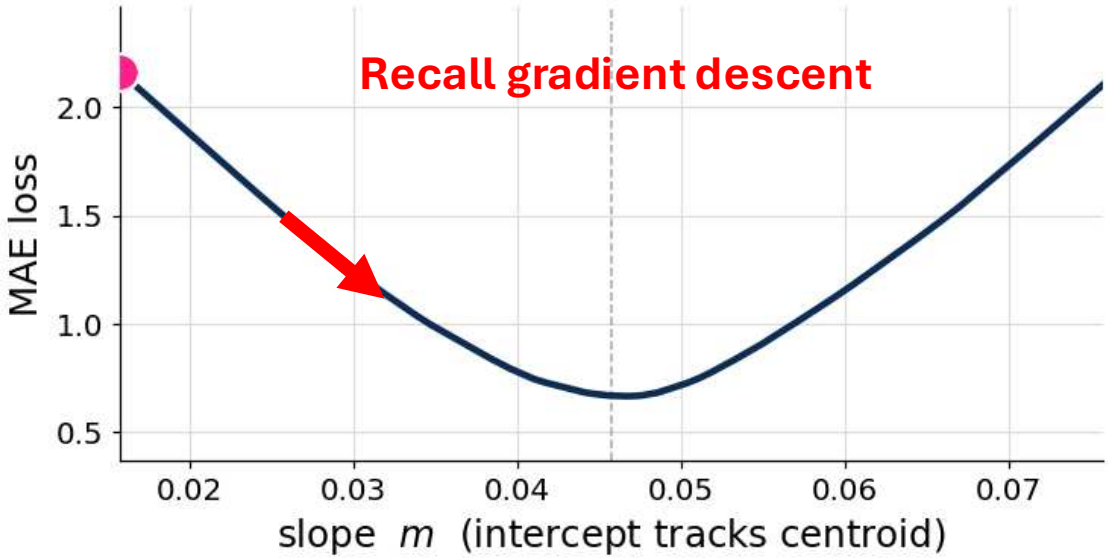
- λ still controls how much penalty is imposed
- Large β_i values are still penalized, though now in a **linear** fashion (not squared)
- Notice the derivative remains large even at values near zero, so **minimizing $\mathcal{L}$ will drive β_i exactly to zero!**

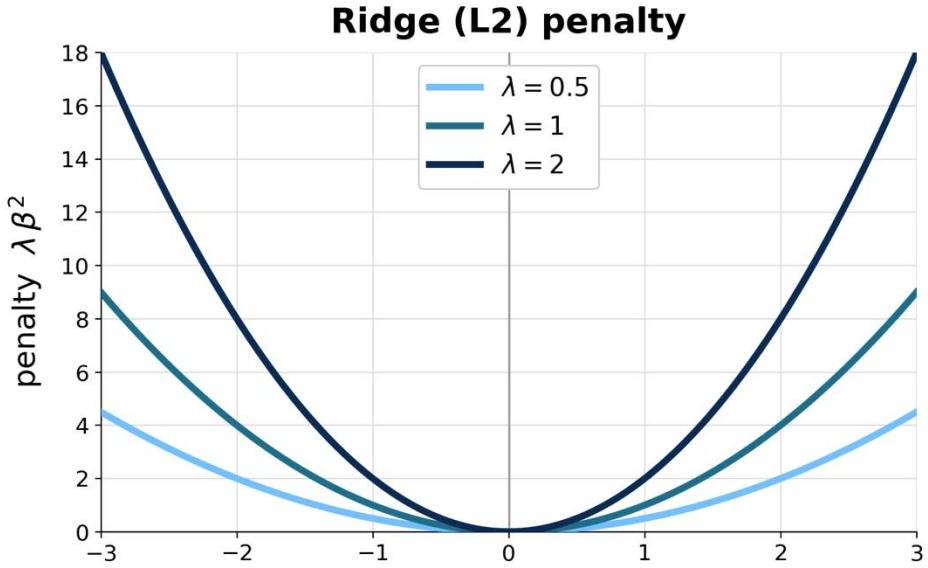


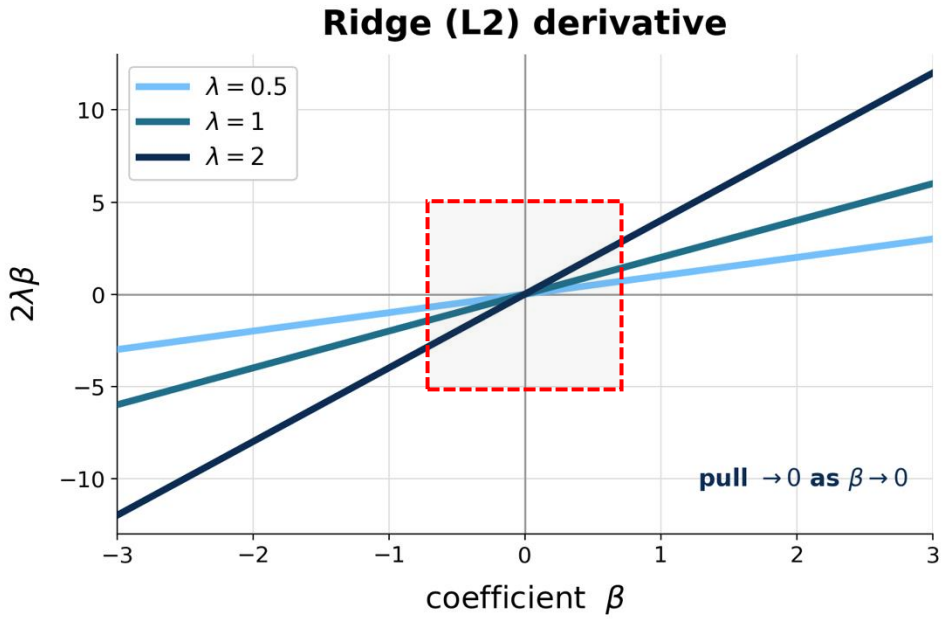
Lasso *vs.* Ridge regression: Derivatives



Lasso *vs.* Ridge regression: Derivatives

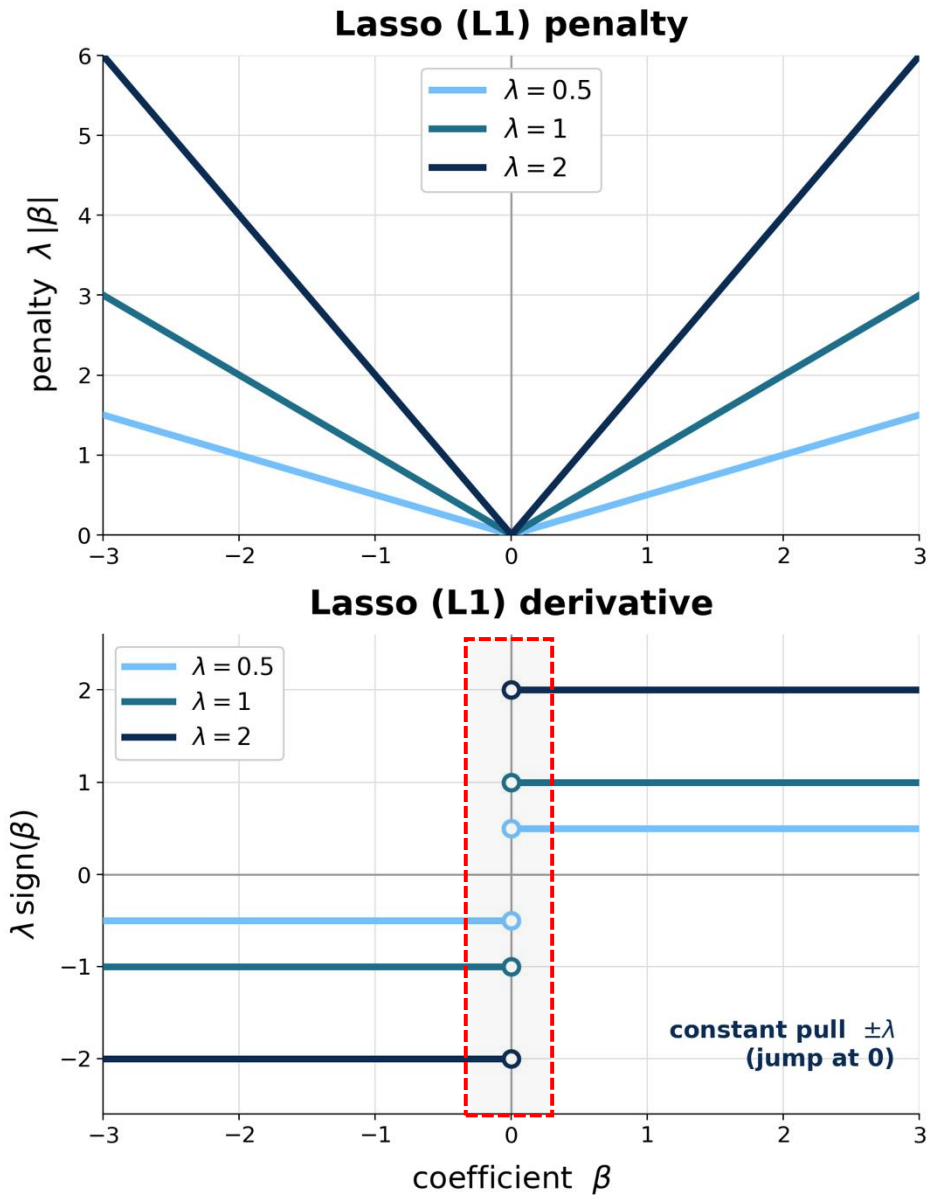






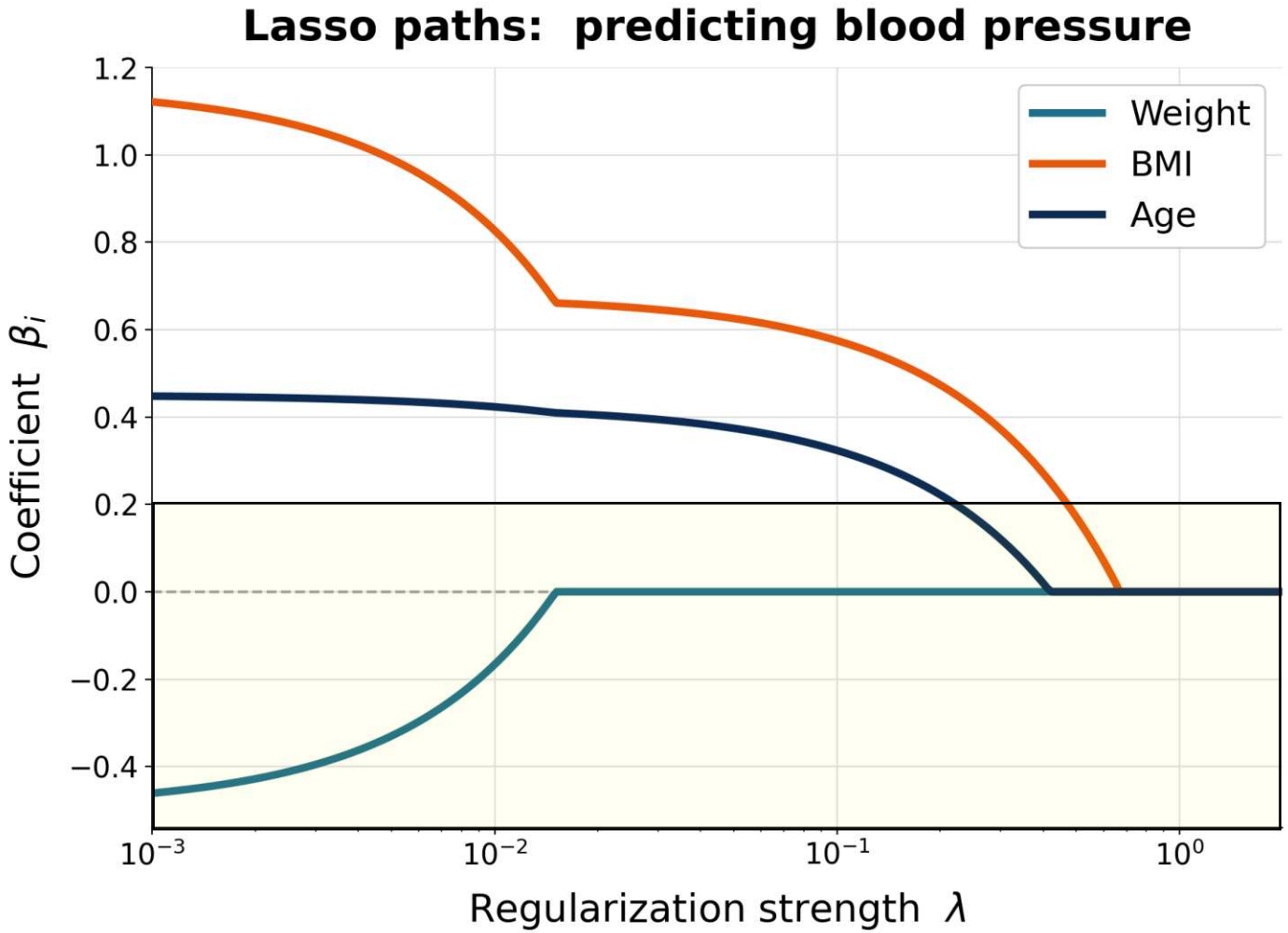
- Think of the derivative as a "driving force" to shrink β
- **With Ridge regression, that driving force goes to zero as $\beta \to 0$**

Lasso *vs.* Ridge regression: Derivatives



- **With Lasso, the driving force remains constant as $\beta \to 0$. So, "unimportant" features will be dropped entirely! This leads to *sparsity* in the predictors.**
- **However…Notice the discontinuity at $\beta = 0$.**
  This is not differentiable! We will deal with this later…

Examples of Lasso regression



Suppose for people of similar height, we are predicting blood pressure from **weight, BMI**, and age

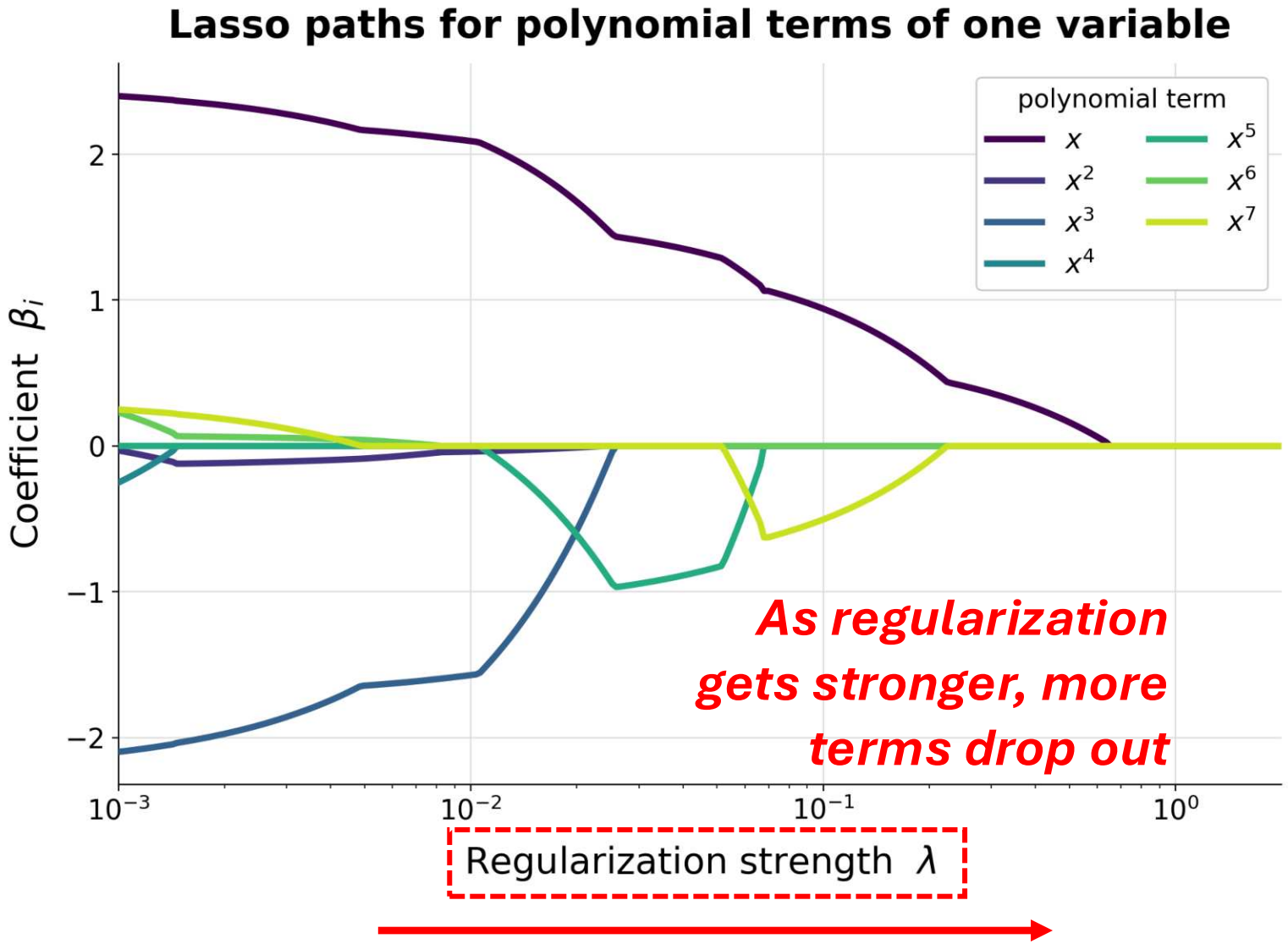
$$\boldsymbol{\beta_0} + \underbrace{\boldsymbol{\beta_1 x_{\text{weight}}} + \boldsymbol{\beta_2 x_{\text{BMI}}}} + \boldsymbol{\beta_3 x_{\text{age}}}$$

These two will be partially collinear

Notice that with Lasso regression, **we learn to drop the redundant variable (weight) by setting $\boldsymbol{\beta_1 = 0}$**

(once λ gets large enough)

Examples of Lasso regression



Consider a model with many terms depending on x^n:

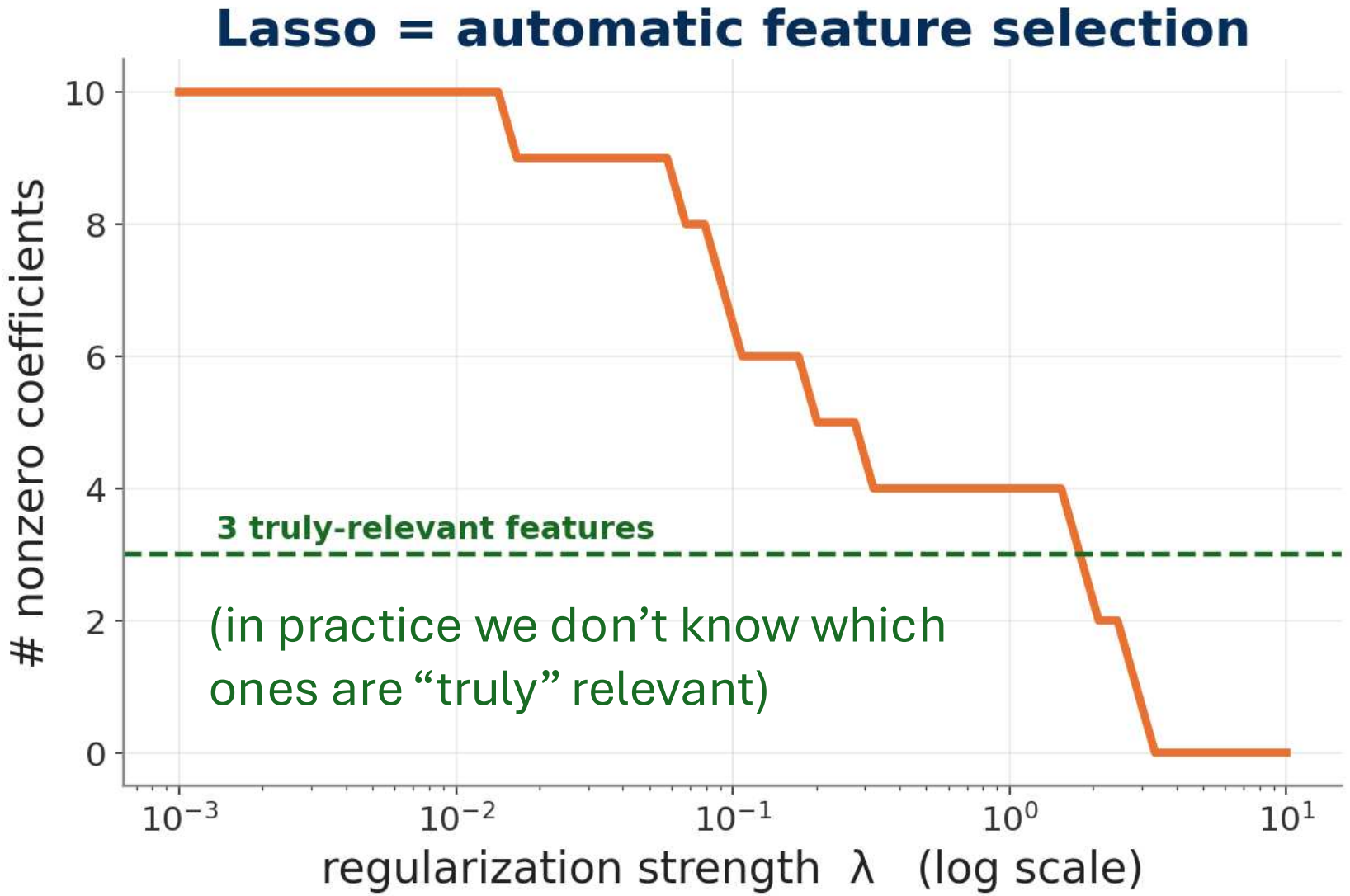
$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5 + \beta_6 x^6 + \beta_7 x^7$$

$$\mathcal{L} = \|y - X\beta\|^2 + \lambda\|\beta\|$$

Keep in mind that changes in β_i can still be non-monotonic when using Lasso regression.

It can "change its mind" on which terms are important.

Lasso regression → *sparsity*

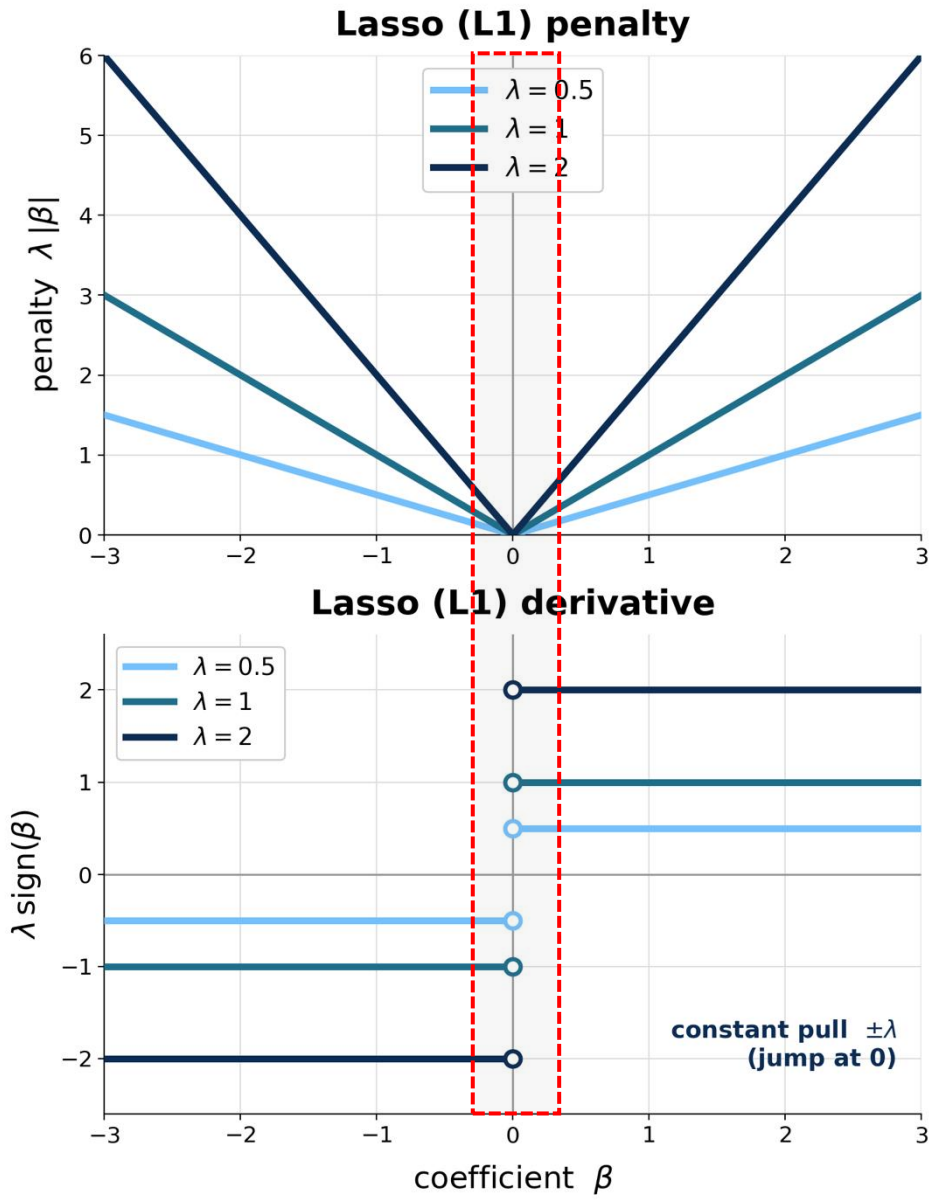


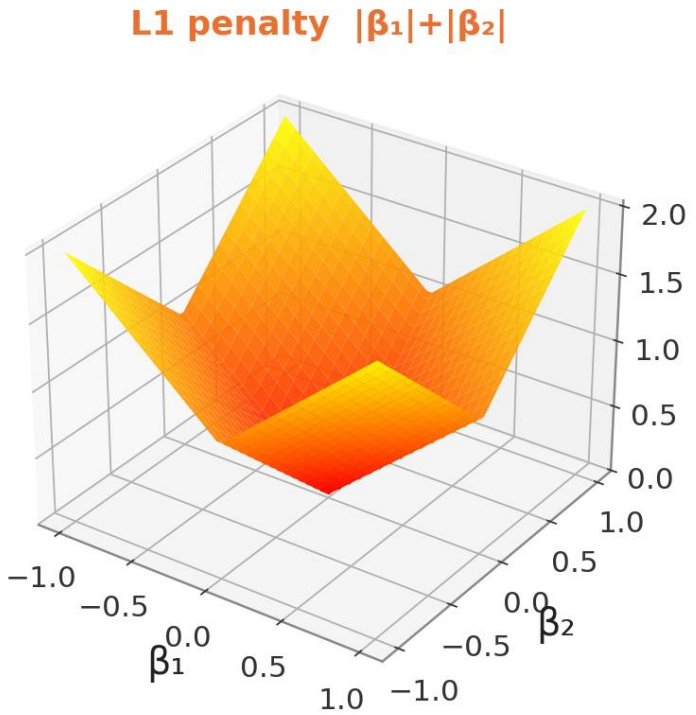
As we increase λ in Lasso, more features are dropped; only the ones deemed most important are kept.

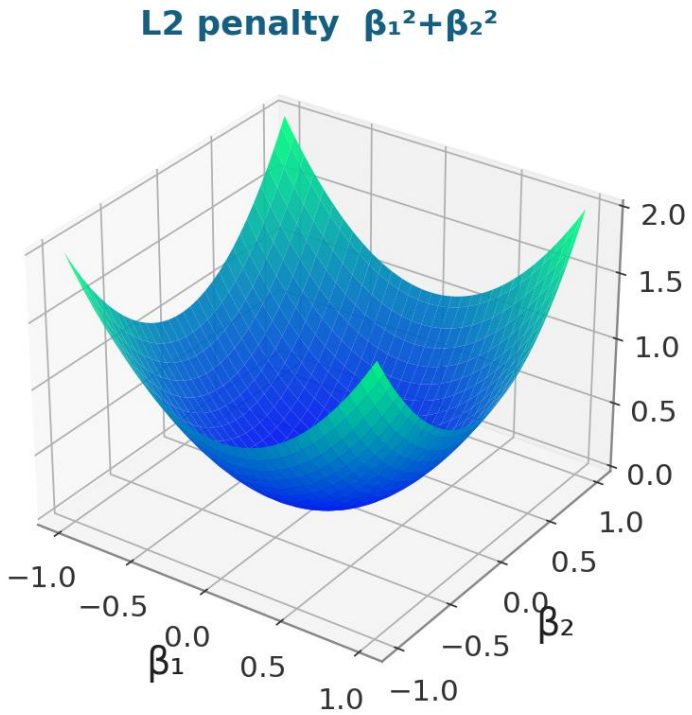
This is great because it leads to **"simpler" models** with much **better interpretability** than Ridge regression.

But there is a tradeoff…
Lack of differentiability!

Lasso has no closed-form analytical solution







With Ridge, we derive an exact solution:

$$\boldsymbol{\beta} = \left(X^T X + \lambda I\right)^{-1} X^T y$$

With Lasso, we cannot solve β exactly ☹

Lasso can be solved using gradient descent

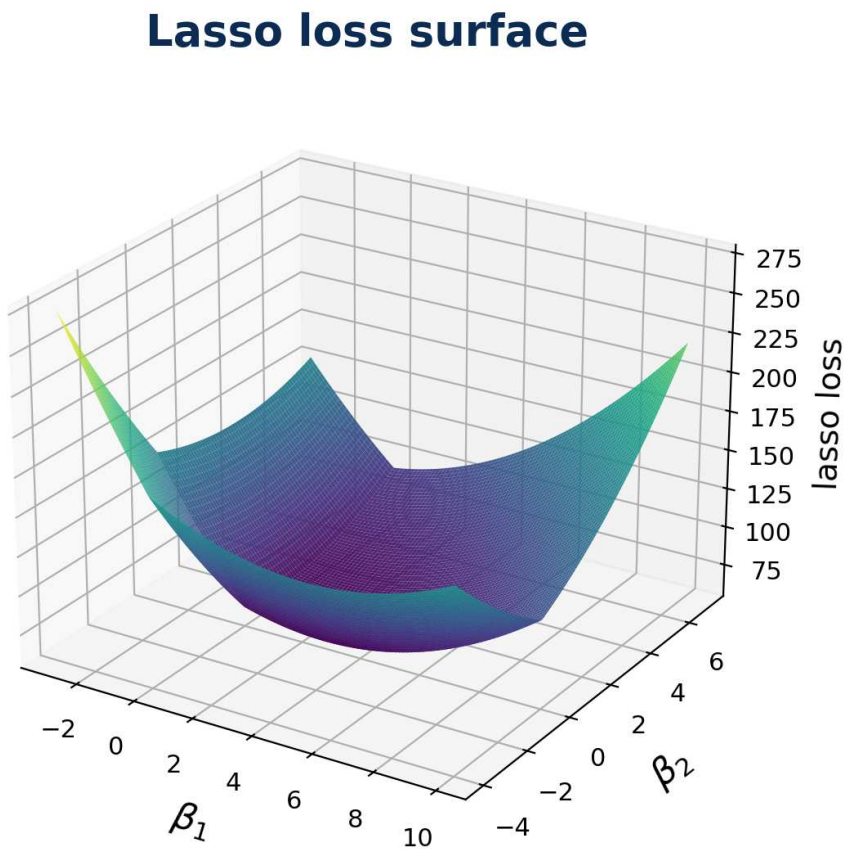
Recall Gradient Descent: $\beta_i' = \beta_i^0 - \eta \left(\frac{d\mathcal{L}}{d\beta_i}\right)\Big|_{\beta_i=\beta_i^0}$ **where** η **is the learning rate**

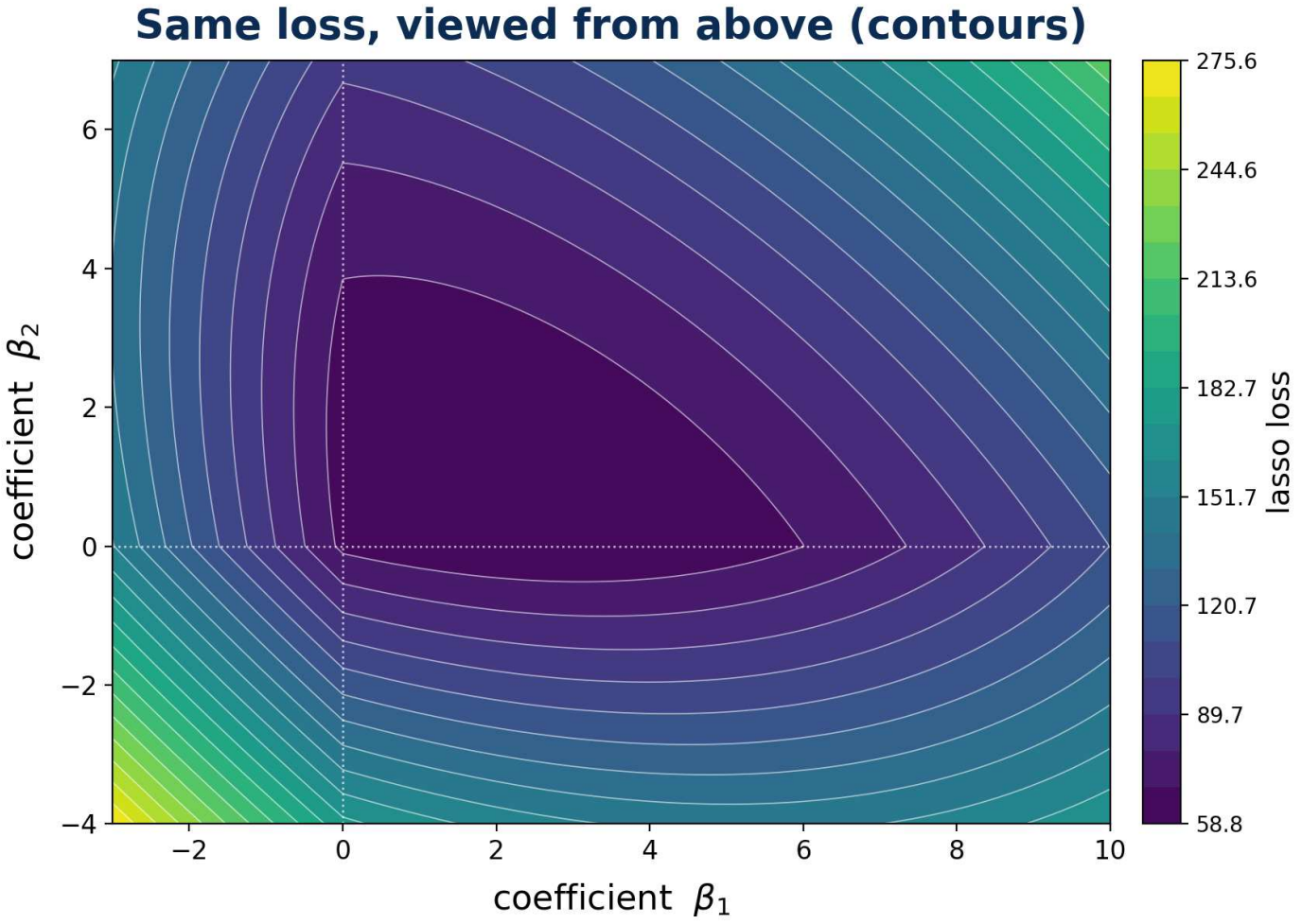


$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 + \lambda\sum_{i=1}^{n}|\beta_i|$$ can be differentiated locally

Lasso can be solved using gradient descent

Fortunately, the loss landscape is still convex for Lasso regression, so gradient descent will converge to the global minimum



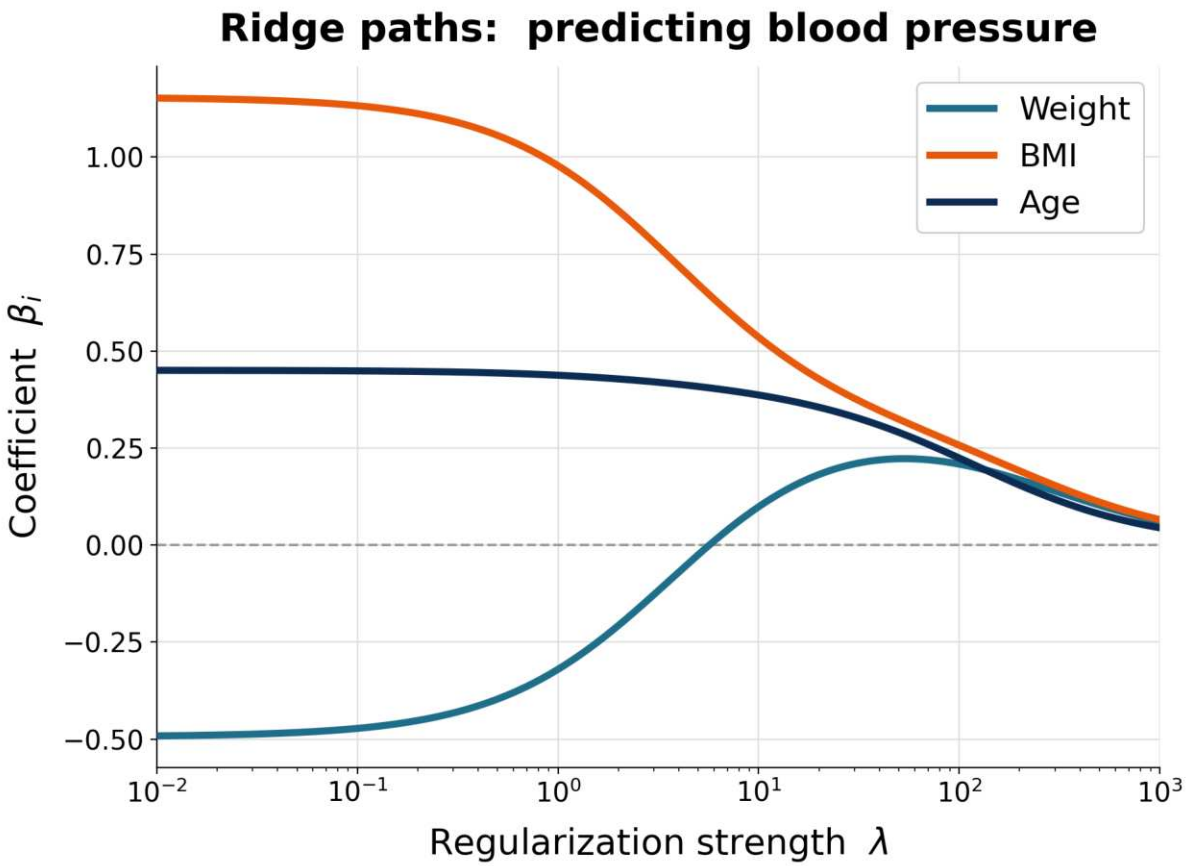


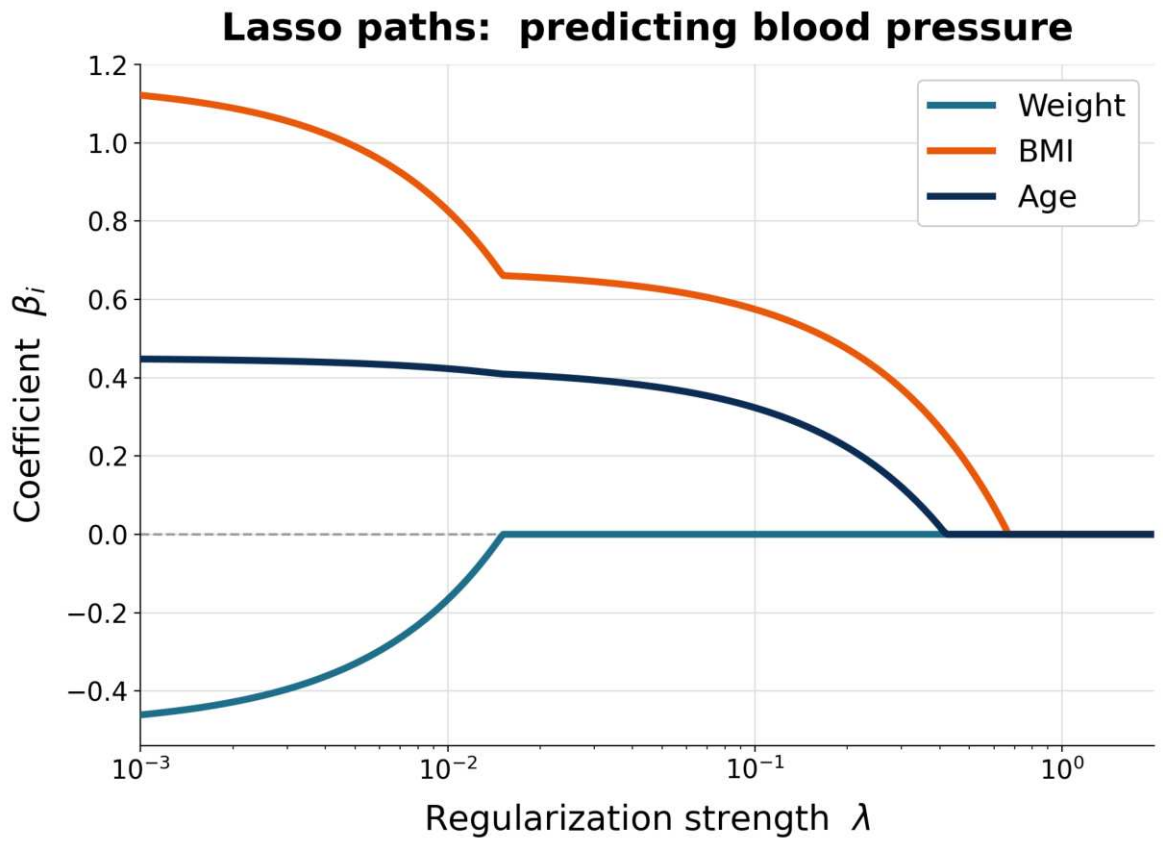
Questions?

Ridge *vs.* Lasso

When to use one over the other?

Same data, two different penalties





Ridge penalizes the largest coefficients, instead "spreading" out β_i while keeping *all* features

Lasso drops features one-by-one, setting $\beta_i = 0$ when features have little influence over data

Which one should you choose?

…Choose…

Ridge (L2) if you…

- Believe ***all* features** are important and should contribute to the model
- Want **minimal test error**
  - Because L2 keeps more features, it can (usually) achieve lower error
- Want fast & easy computation
  - β solved analytically

Lasso (L1) if you…

- Believe there are redundant features that could be removed
- Want a highly ***interpretable* model** based on a sparse feature set
- Don't mind more involved computations – note this is less of a concern with modern computers
  - β solved *via* Gradient Descent

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 + \lambda\sum_{i=1}^{n}|\beta_i|$$

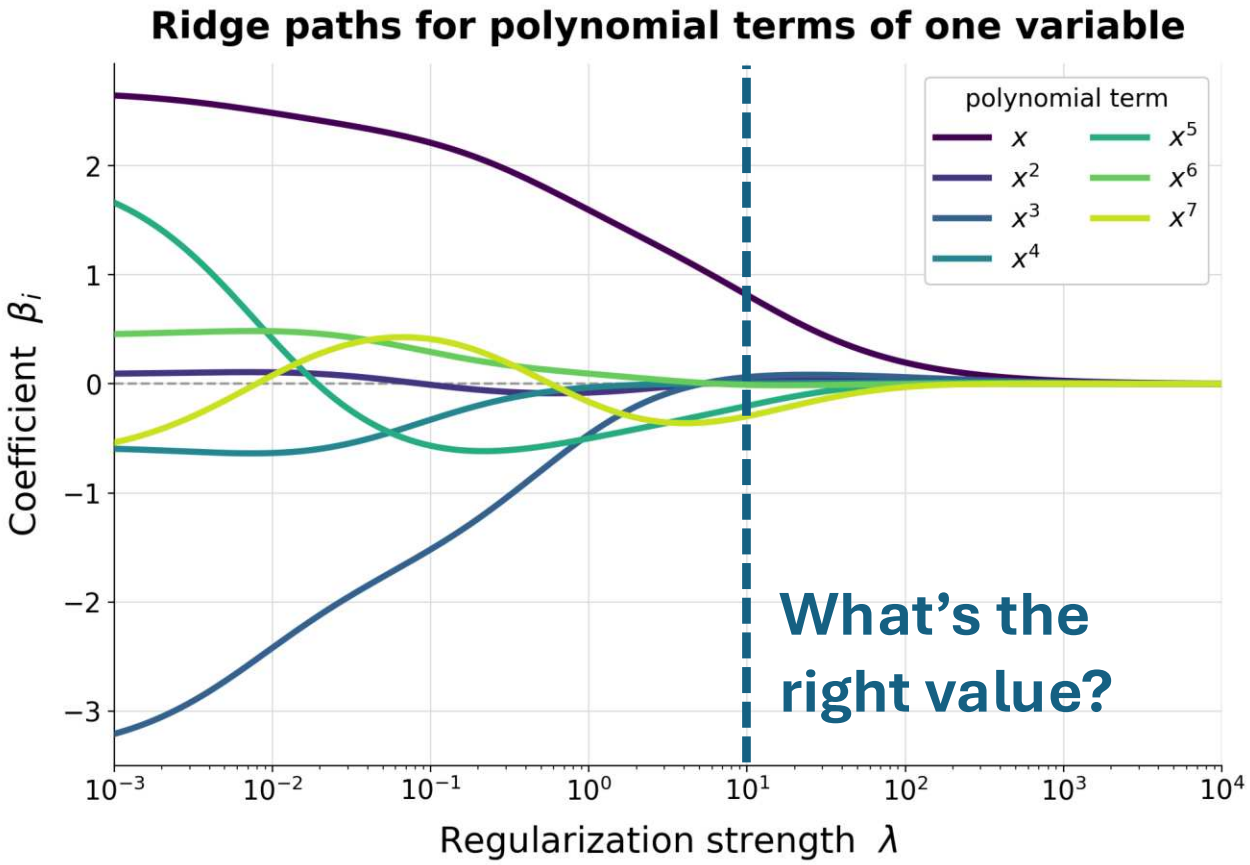
$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 + \lambda\sum_{i=1}^{n}\beta_i^2$$

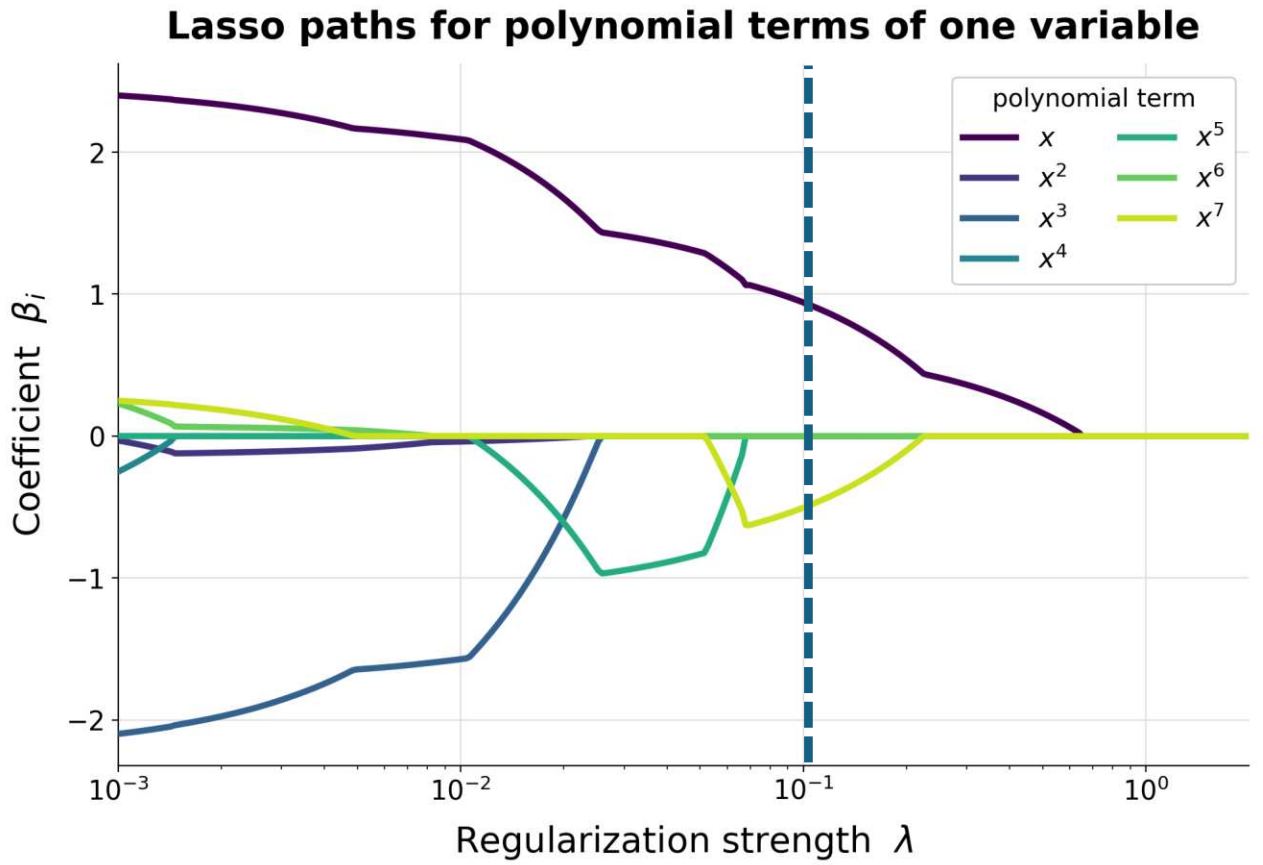
How to set the λ hyperparameter

*Remember…this controls the **regularization strength***

How λ affects the coefficients β_i

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5 + \beta_6 x^6 + \beta_7 x^7$$

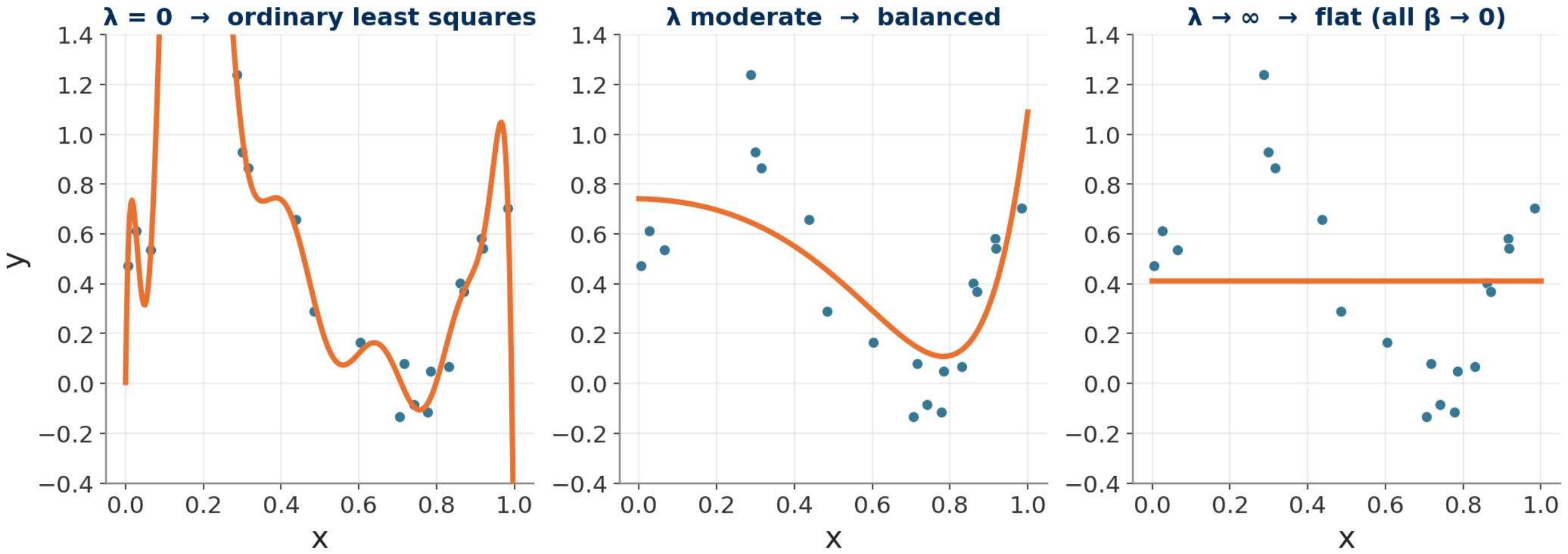




Larger $\lambda \rightarrow$ smaller β_i (Ridge) and fewer predictors kept (Lasso)

How λ affects the fit

Think of λ as a dial for the "smoothness" of our model

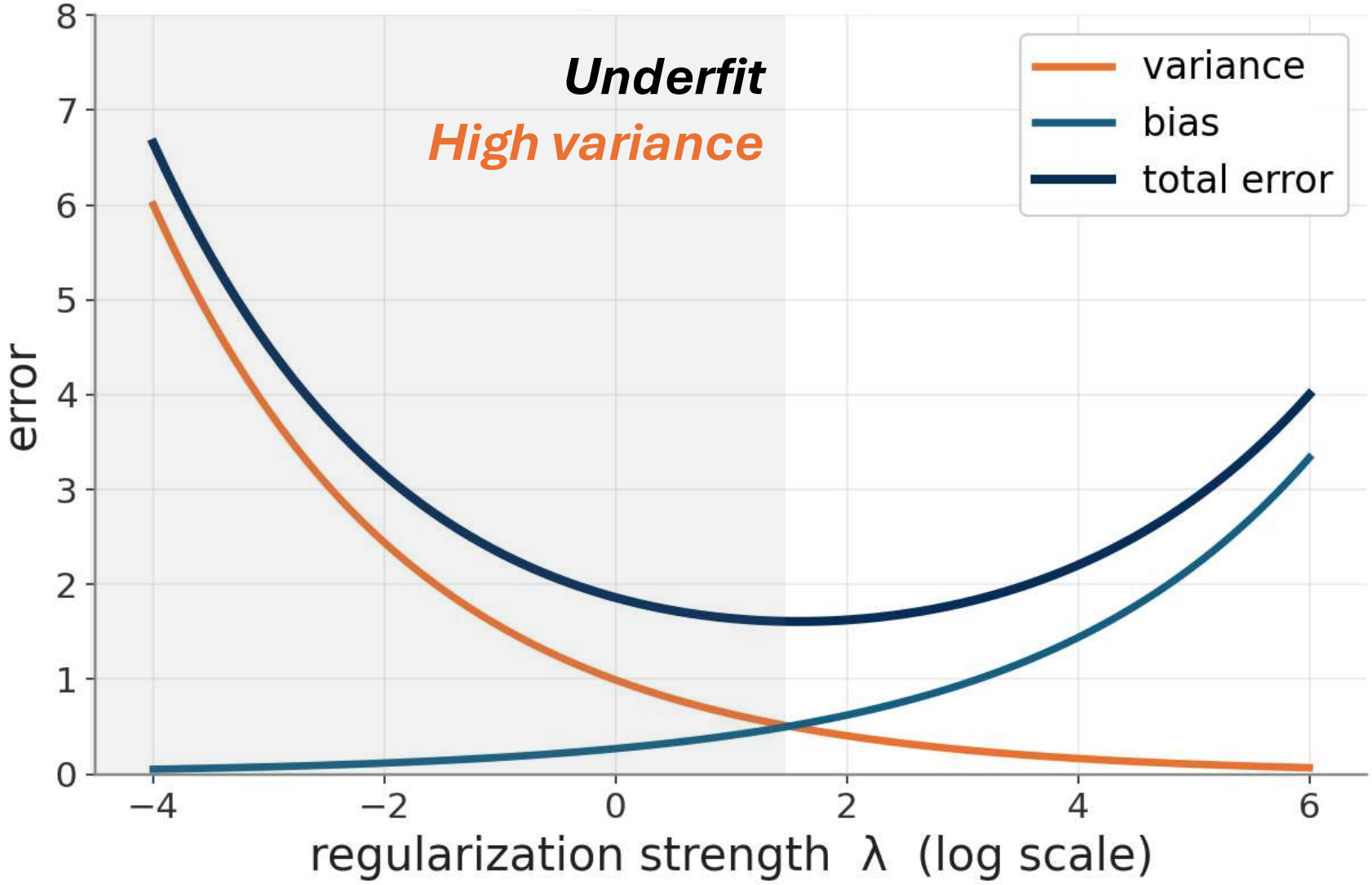


Overfitting → **Underfitting**

λ too small

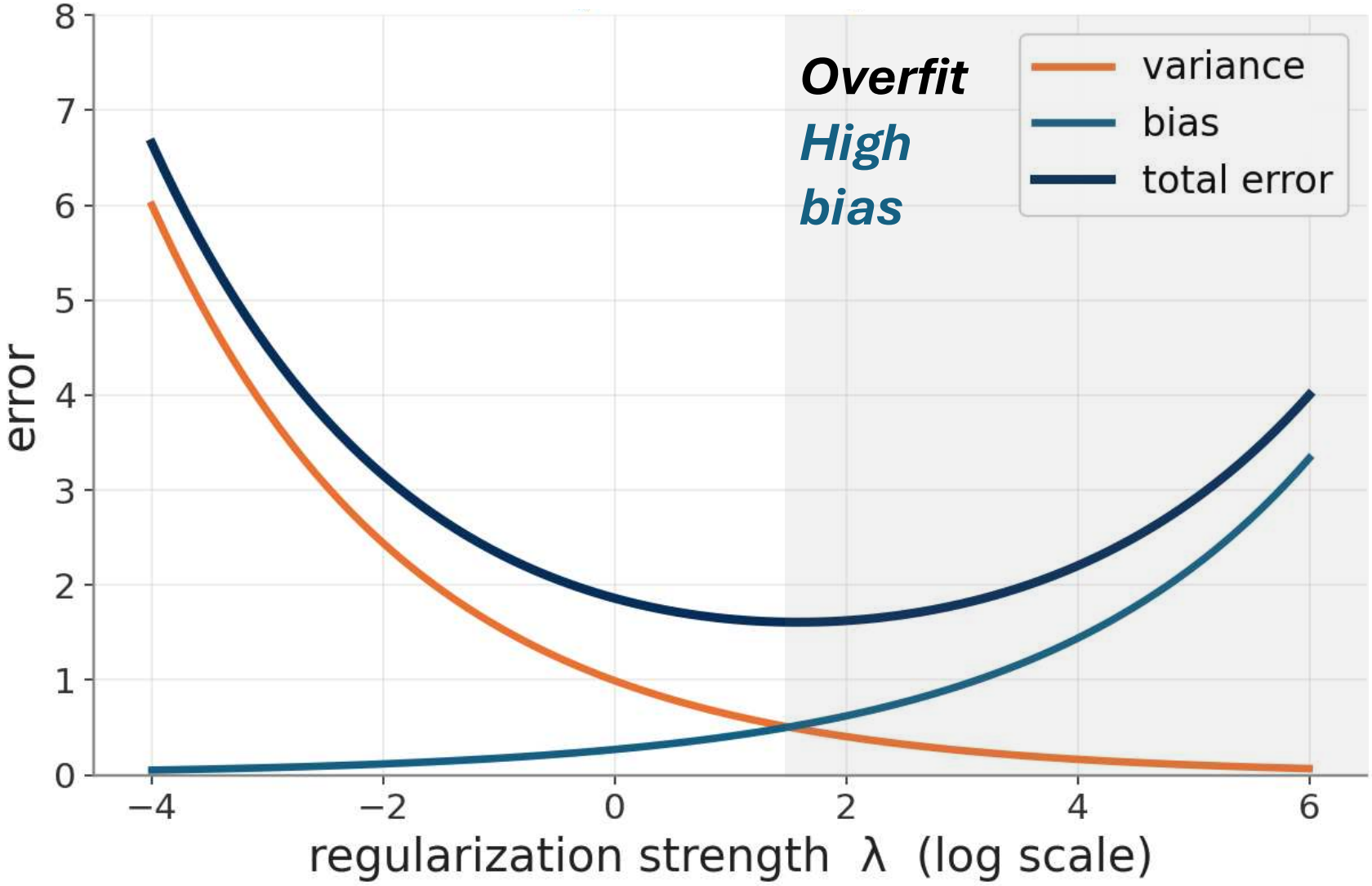
λ too large

Larger λ leads to more bias, less variance



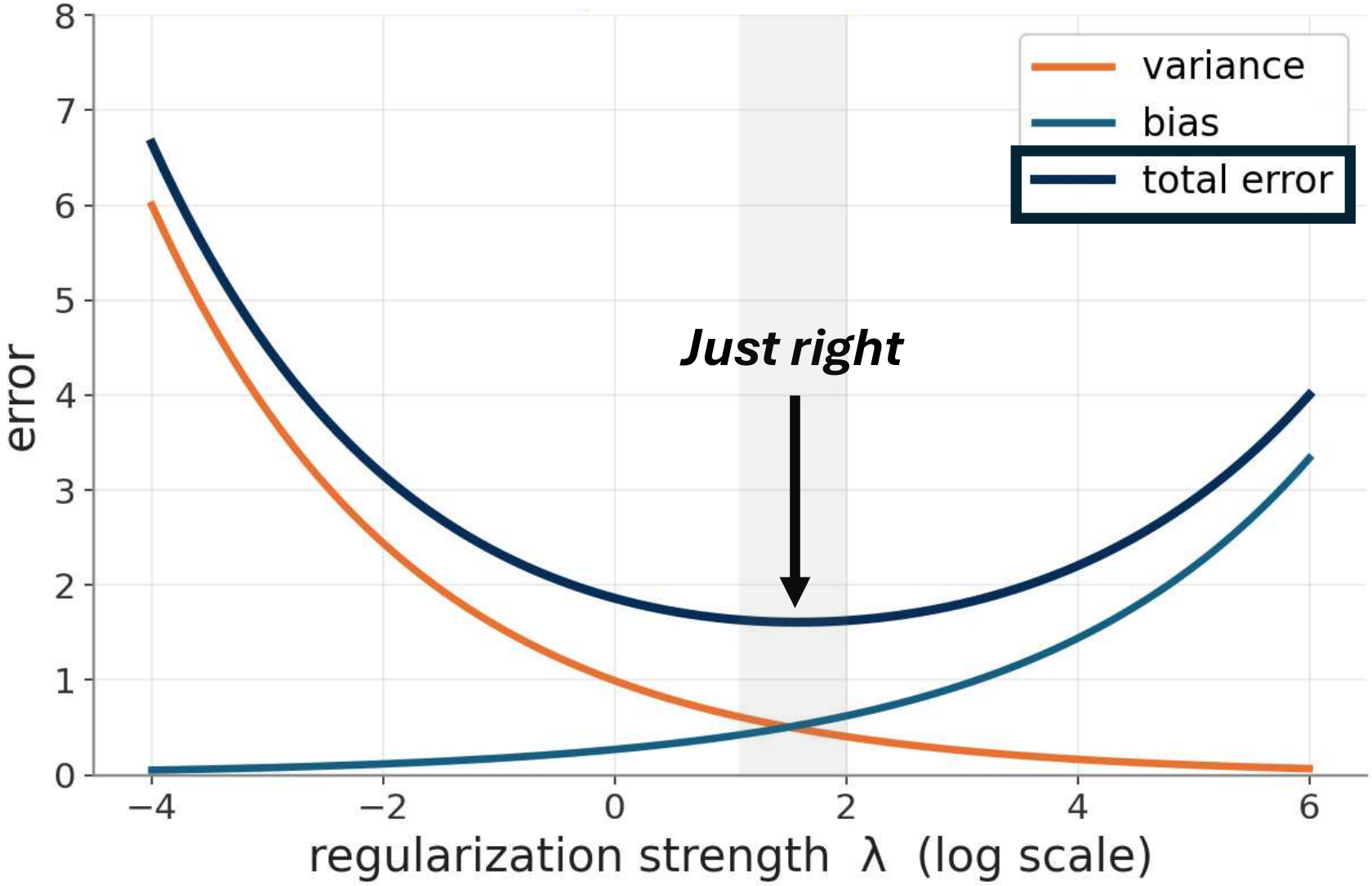
When λ is too small, the model will fluctuate wildly depending on which data you sample, because it can respond through **large changes in β_i** → **high variance**

Larger λ leads to more bias, less variance



When λ is too large, you are introducing **systematic error** by forcing the model to be overly simplistic → **high bias**

Larger λ leads to more bias, less variance



But on which dataset do we evaluate the **"total error"**

When λ is just right, you strike a **balance between model flexibility and generalizability**

Training, validation, and test data

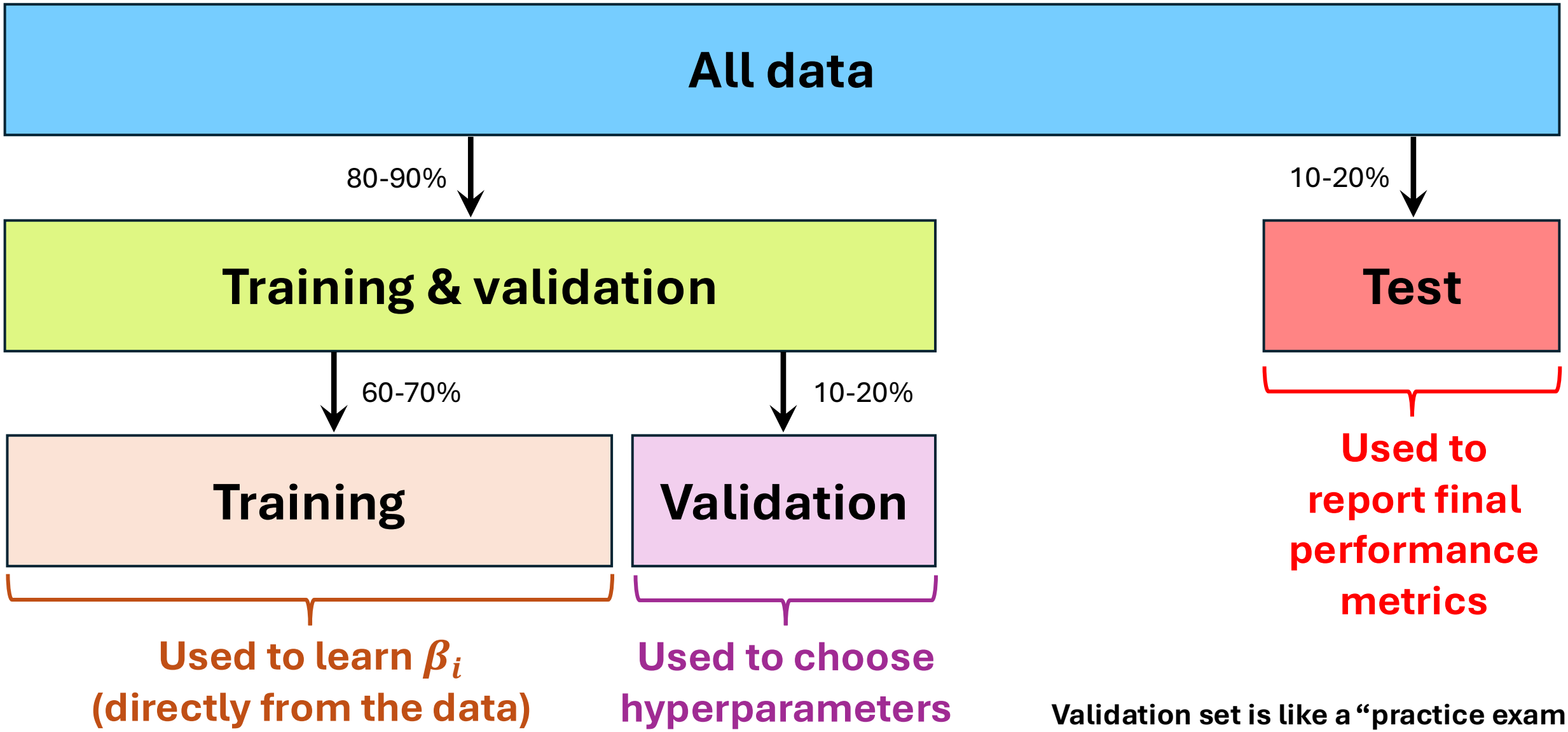


Model learns the task

This is the data we want our model to perform well on…**But tuning λ (or any hyperparameter) on the test set would be "cheating"**
Then it wouldn't be *new* data anymore!

How good is the model on "new" data?

Training, validation, and test data



Validation set is like a "practice exam"

Step-by-step: Choosing the right λ

1) Split your dataset into training (60%), validation (20%), and test (20%)

- Usually this is done randomly but watch out for data leakage (*e.g.*, duplicates)

Step-by-step: Choosing the right λ

1) Split your dataset into training (60%), validation (20%), and test (20%)
   - Usually this is done randomly but watch out for data leakage (e.g., duplicates)

2) Decide **which λ values** to test – then for each, fit β_i on the training set
   - Use $\beta = (X^T X + \lambda I)^{-1} X^T y$ for Ridge or gradient descent for Lasso

Step-by-step: Choosing the right λ

1) Split your dataset into training (60%), validation (20%), and test (20%)
   - Usually this is done randomly but watch out for data leakage (e.g., duplicates)

2) Decide which λ values to test – then for each, fit β_i on the training set
   - Use $\beta = (X^T X + \lambda I)^{-1} X^T y$ for Ridge or gradient descent for Lasso

3) Using that trained model, **compute MSE on the validation set for each λ**

Step-by-step: Choosing the right λ

1) Split your dataset into training (60%), validation (20%), and test (20%)
   - Usually this is done randomly but watch out for data leakage (e.g., duplicates)

2) Decide which λ values to test – then for each, fit β_i on the training set
   - Use $\beta = (X^T X + \lambda I)^{-1} X^T y$ for Ridge or gradient descent for Lasso

3) Using that trained model, compute MSE on the validation set for each λ

4) Choose the λ with the smallest validation MSE

Step-by-step: Choosing the right λ

1) Split your dataset into training (60%), validation (20%), and test (20%)
   - Usually this is done randomly but watch out for data leakage (e.g., duplicates)

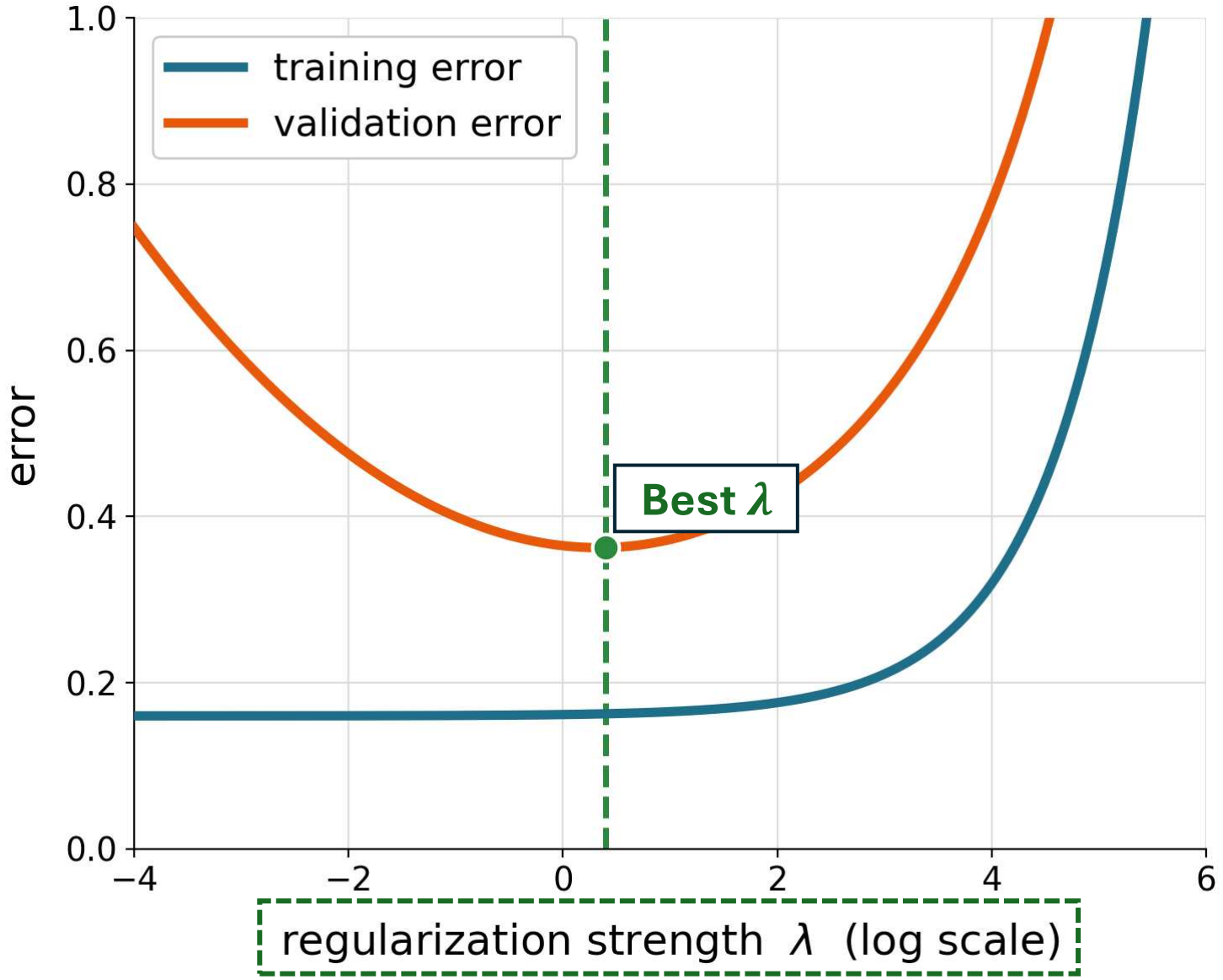
2) Decide which λ values to test – then for each, fit β_i on the training set
   - Use $\beta = (X^T X + \lambda I)^{-1} X^T y$ for Ridge or gradient descent for Lasso

3) Using that trained model, compute MSE on the validation set for each λ

4) Choose the λ with the smallest validation MSE

5) Refit on training + validation **using this λ**, then **report MSE on the test data**
   - Only evaluate on the test data once. No cheating!

Training and validation loss *vs.* λ



- Keep in mind that **adding regularization (increasing λ) will *always* increase error on the training set**
- That's okay – **we care about minimizing error on the validation set**, which (hopefully) translates to minimal error on *new* data

Some caveats and applications of regularized regression

Standardization matters a lot here!

Suppose we are building a model to predict car price based on its **mileage** and **number of previous owners**



Number of previous owners: 1-5



Mileage: anywhere from 0 to 300,000 miles

$$\hat{y}_{\text{price}} = \beta_0 + \beta_1 x_{\text{owners}} + \beta_2 x_{\text{mileage}}$$

Always standardize

It never hurts!

β_1 will need to be huge in comparison to β_2 to compensate for the different input magnitude, and **regularization will penalize this!**

Don’t penalize the intercept (β_0)

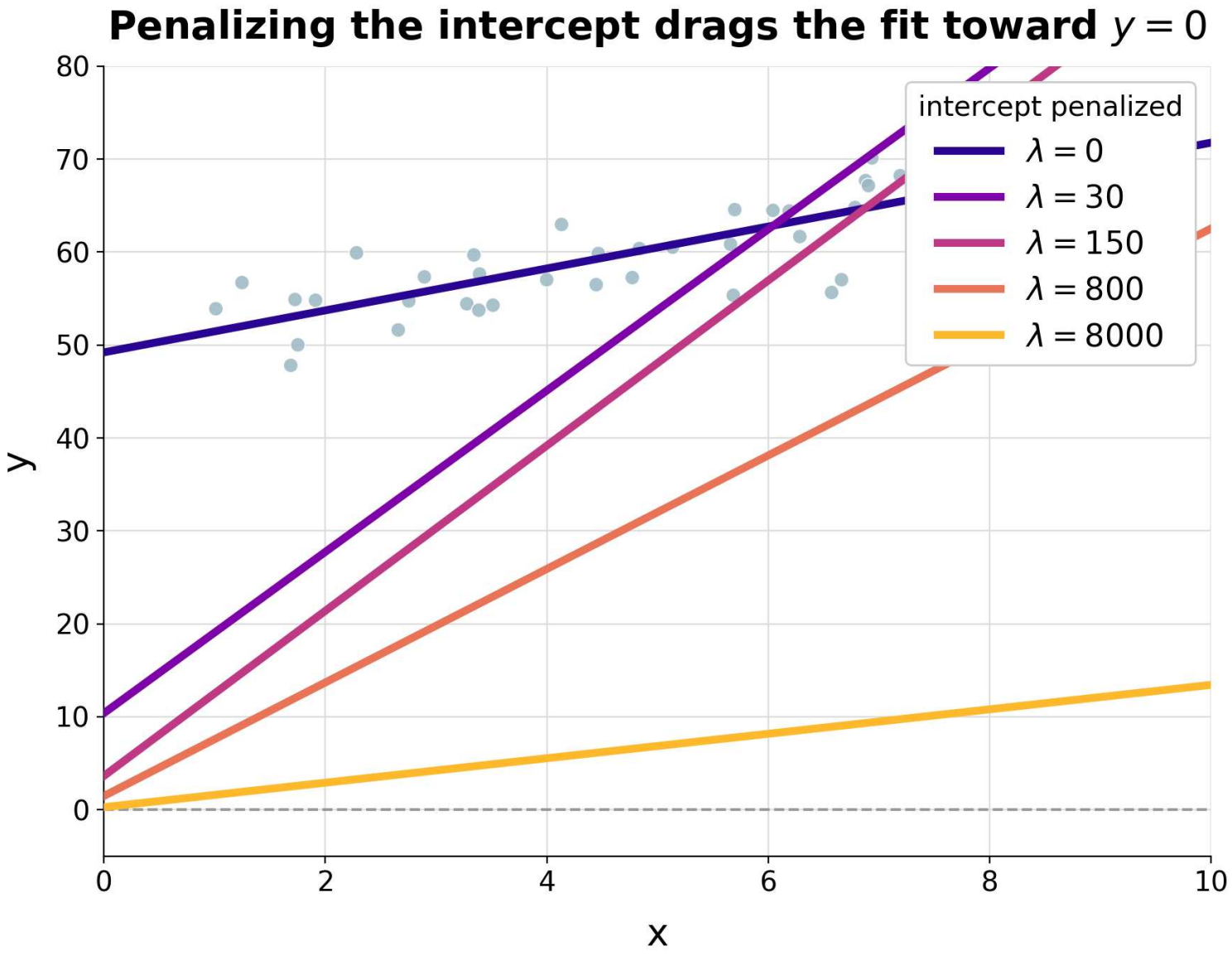
$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 + \lambda\sum_{i=1}^{n}|\beta_i|$$

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 + \lambda\sum_{i=1}^{n}\beta_i^2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots$$

- Notice **our summation begins from $i = 1$ not $i = 0$**
- We penalize coefficients that represent slopes ($\beta_1, \beta_2, \ldots$) ***not* the intercept (β_0)**
- **β_0 simply represents the baseline of the data** – it has no relation to the model complexity, which is what we actually want to penalize
- Including β_0 in the regularization would only shift/bias our data toward zero

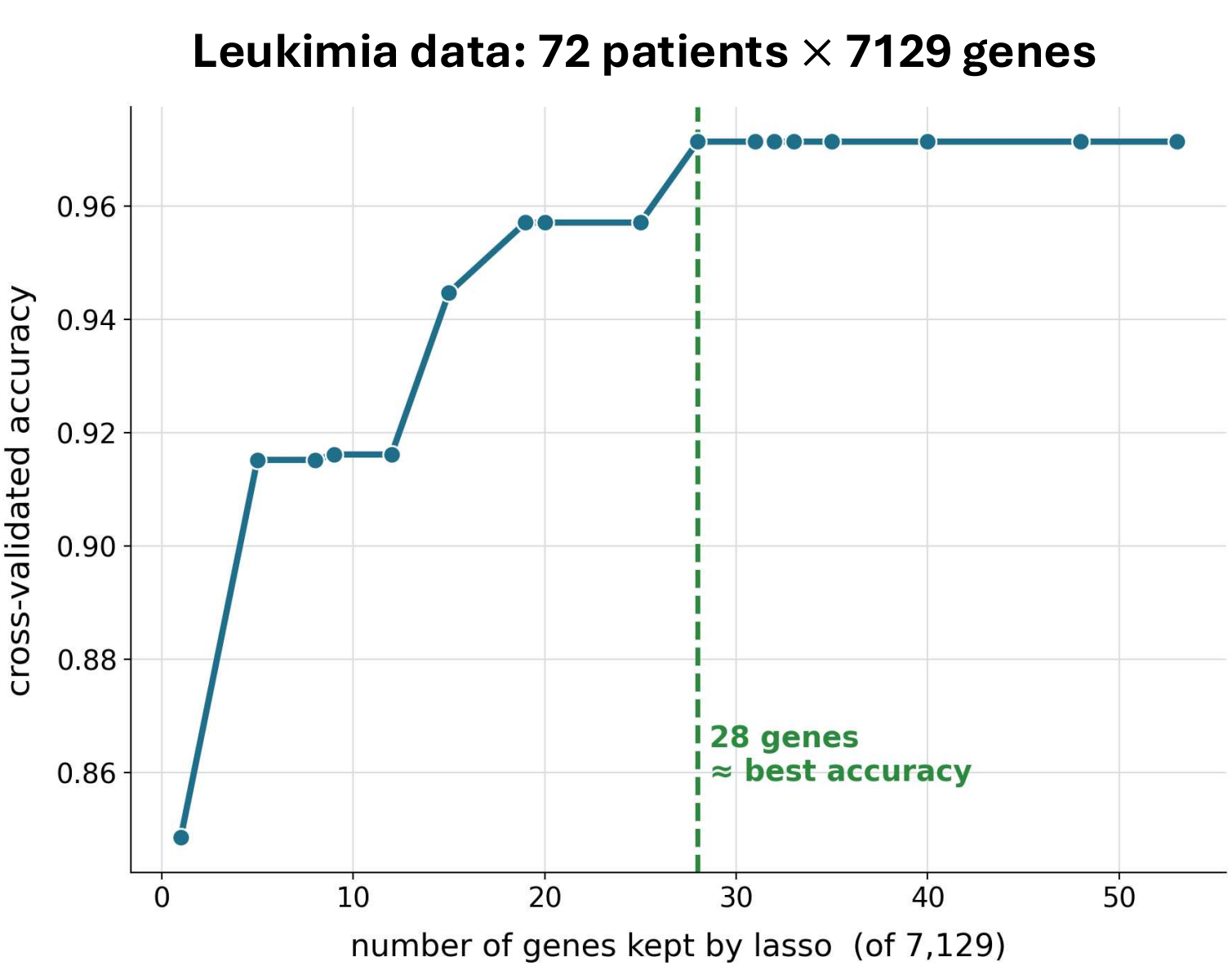
Don’t penalize the intercept (β_0)

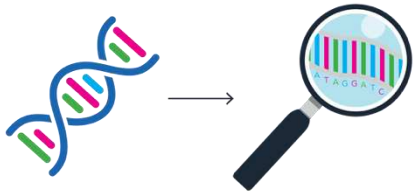


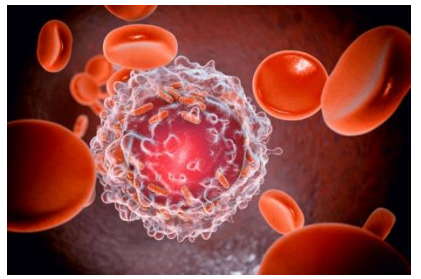
This is what happens if we *do* penalize the intercept…
Not good!

Note that most Python packages should handle this for you automatically when doing Ridge and Lasso regression

Some real applications of regularization







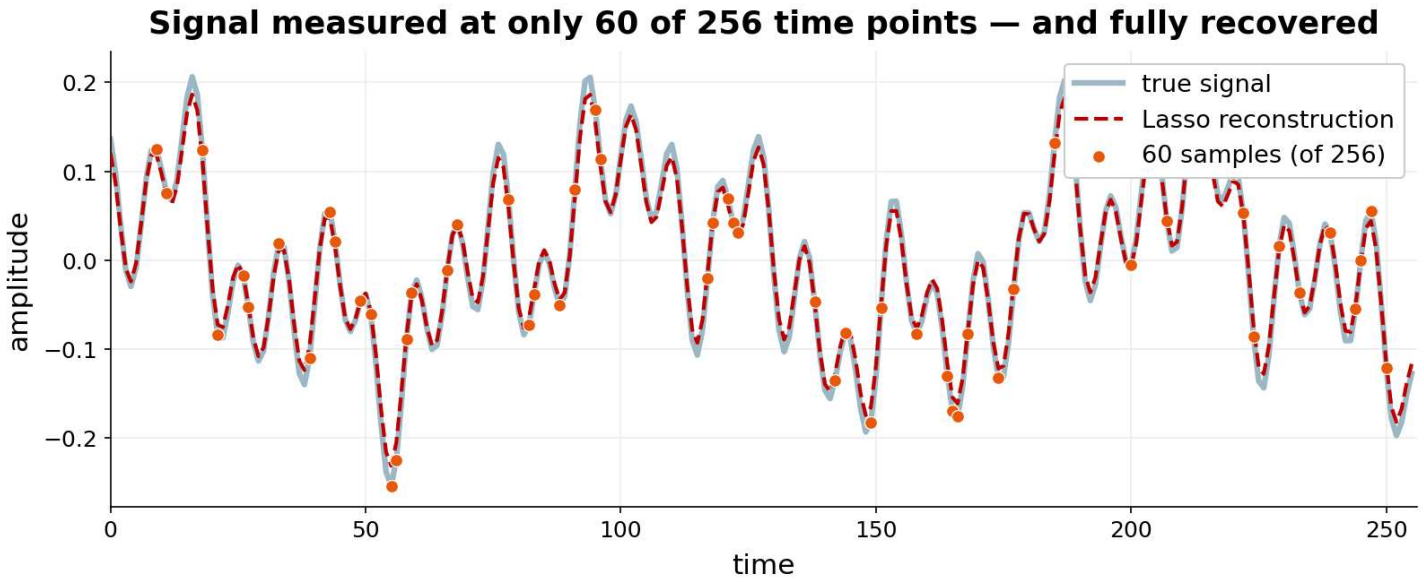
- Data from 72 patients, **7129 genes each**

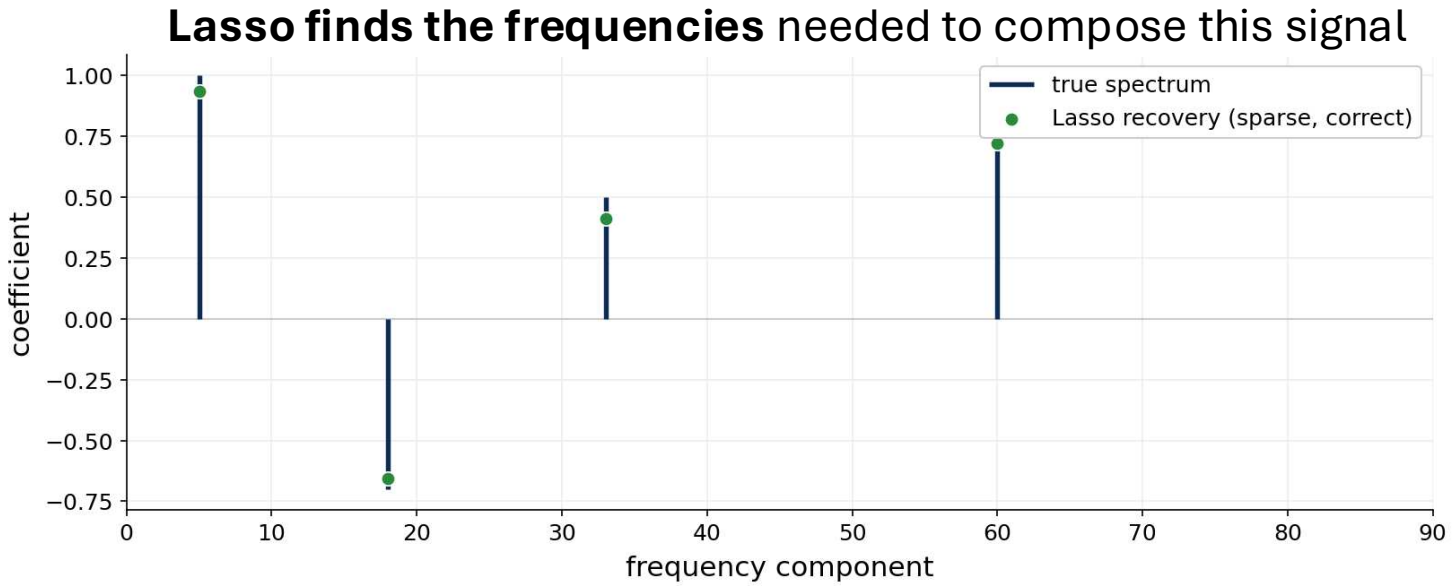
  Many features to choose from…

- Lasso found that **just 28 of these genes** could give highly accurate predictions of leukemia subtype

Note: it doesn't mean these genes *cause* the leukemia, just correlate

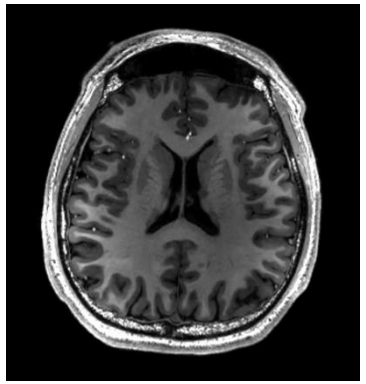
Some real applications of regularization





Compressed Sensing:

- Describe complex signals using frequencies as the predictors
- From limited inputs (t) we build a large set of features $\cos(f_i t)$
- Lasso learns that we can predict the entire signal from a relatively small subset of frequencies (f_i)



Commonly used to reduce scan time in MRI

Questions?

What to know for HW + Exams

- Know when to use **Ridge *vs.* Lasso**
- Understand the effect of the **regularization strength (λ)**
- Know the role of the **validation set** in choosing the "right" λ
- Given some dataset and a chosen λ, **calculate $\boldsymbol{\beta}_i$**