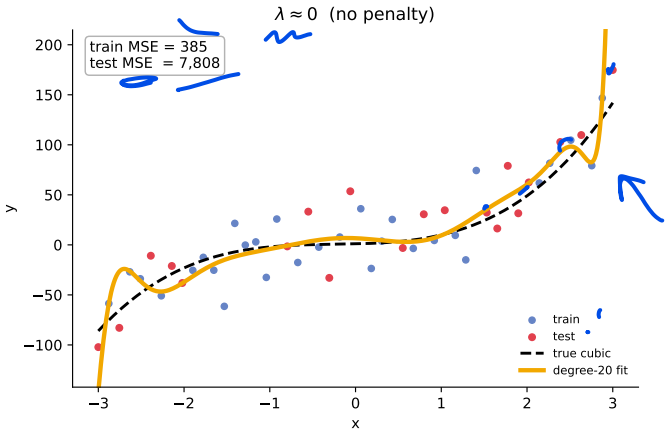
One knob controls fit vs. memorization

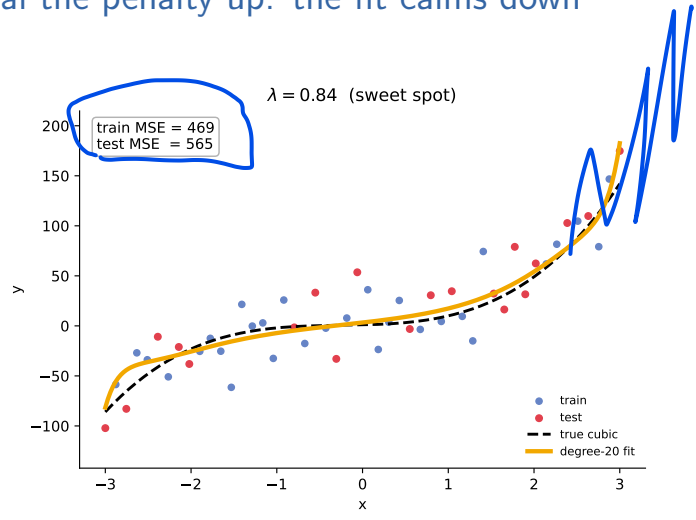


A **degree-20 polynomial** fit to a noisy cubic, with **no penalty** ($\lambda \approx 0$).

It threads the training points but **thrashes** between them.

train MSE $= 385$ but **test MSE** $= 7808$.
Classic overfitting: memorized the noise.

Dial the penalty up: the fit calms down

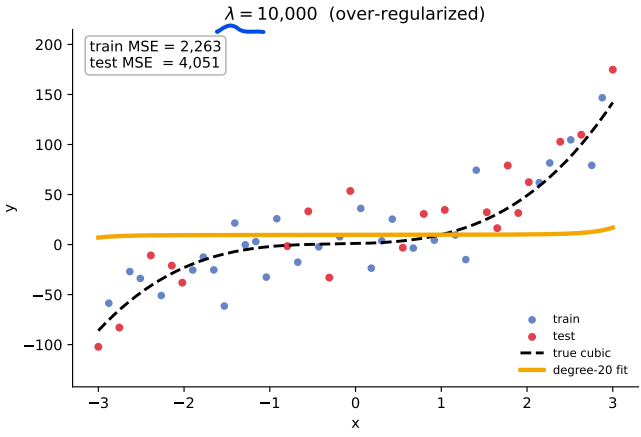


Same model class, same data. We just added a penalty $\lambda = 0.84$ that **discourages large coefficients**.

The curve now **hugs the true cubic**.

train MSE = 469 (up a bit) **test MSE** = 565 (down 14×).

Crank it too far: now we underfit



With $\lambda = 10{,}000$ the penalty dominates: coefficients are crushed toward 0 and the fit goes **nearly flat**.

Both errors climb (train 2263, test 4051).

Regularization is a continuous knob. Too little ⇒ memorize noise. Too much ⇒ underfit. There is a **sweet spot** in between — so how do we find it?

Outline

L_1, L_2

What is regularization?

Methods that inject an **inductive bias** (typically "low complexity") into the model to reduce overfitting and improve the bias-variance tradeoff.

- **Explicit** regularization: add a complexity penalty to ERM (L_1, L_2).
- **Implicit** regularization: early stopping, data augmentation, dropout, ensembling.
- **Structured** regularization: let the penalty use *known structure* among the weights — e.g., in a polynomial, penalize *high-degree* coefficients harder than low-degree ones, encoding a preference for smooth, low-degree behaviour.

This lecture focuses on **explicit** L_1 and L_2 penalties on the coefficients — exactly the knob we just turned.

Regularized empirical risk minimization

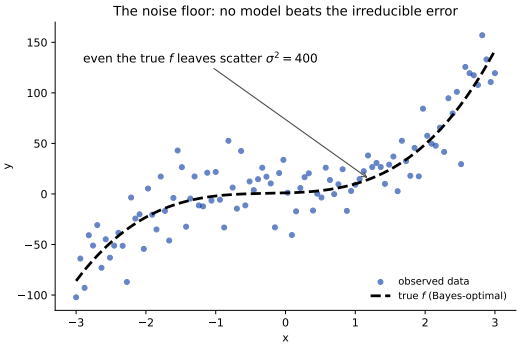
Trade off two competing goals in a **single objective**:

$$\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \underbrace{\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}_{\text{fit the data}} + \lambda \underbrace{J(\boldsymbol{\theta})}_{\text{stay simple}}$$

- $J(\boldsymbol{\theta})$: **complexity penalty** / regularizer (how "big" the model is)
- $\lambda \geq 0$: **regularization strength** — the knob from the cold open
- $\lambda = 0$: plain ERM (overfit panel); $\lambda \to \infty$: model as "simple" as possible (underfit panel)
- Choosing λ is a tuning problem — usually via **cross-validation** (last week's lecture)

That penalty term **raises bias** and **lowers variance**. To see why that is a trade worth making, we need a vocabulary for the two errors at play.

The noise floor (Bayes risk): what no model can beat



Decompose expected test error:

$$\text{MSE} = \underbrace{\text{Bias}^2 + \text{Var}}_{\text{we can shrink this}} + \underbrace{\sigma^2}_{\text{noise floor}}$$

- The **Bayes risk** is σ^2: the error left even for an oracle that *knew* the true f (the **Bayes-optimal** predictor).
- It is **irreducible** — pure noise in the data, nothing to learn.
- Here $\sigma = 20 \Rightarrow \sigma^2 \approx 400$. Our sweet-spot fit hit 565 — close to the floor!

Regularization closes the gap **above** the floor — never below it.

Bias and variance, intuitively

Two sources of *reducible* error when an estimated model differs from the truth:

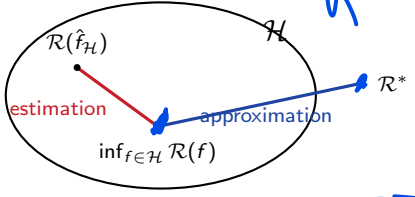
- **Bias**: how far the *average* estimate is from the truth
  - High bias = model class too restrictive to capture truth (*underfit* — our flat panel)
- **Variance**: how much the estimate *wiggles* across different training sets
  - High variance = model class so flexible it chases noise (*overfit* — our thrashing panel)

Regularization **raises bias a little** to **cut variance a lot**. The very same trade has another, more precise name — shown next.

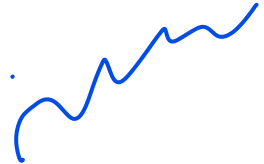
Estimation and approximation error

We want $\hat{f}_{\mathcal{H}}$ to have risk close to the **Bayes risk** $\mathcal{R}^*$ (the noise floor). Split that gap — the *excess risk* — into two pieces:

$$\underbrace{\mathcal{R}(\hat{f}_{\mathcal{H}}) - \mathcal{R}^*}_{\text{excess risk}} = \underbrace{\left[\mathcal{R}(\hat{f}_{\mathcal{H}}) - \inf_{f \in \mathcal{H}} \mathcal{R}(f)\right]}_{\text{estimation error}} + \underbrace{\left[\inf_{f \in \mathcal{H}} \mathcal{R}(f) - \mathcal{R}^*\right]}_{\text{approximation error}}$$



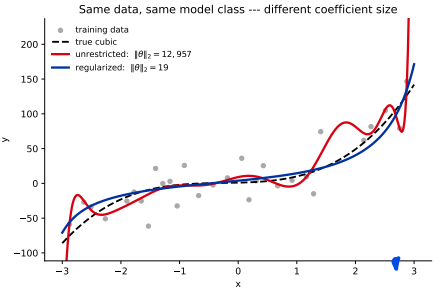
- **Estimation error**: we fit $\hat{f}_{\mathcal{H}}$ by ERM on *finite* data, so we never quite reach the best $f \in \mathcal{H}$ — this is the *variance* side.
- **Approximation error**: $\mathcal{H}$ usually doesn't even contain the Bayes-optimal f^* — this is the *bias* side, fixed by the model class.
- **Regularization** shrinks $\mathcal{H}$: approximation error up a touch, estimation error down a lot.



Ridge regression (L_2)

We saw that *big coefficients mean a complex model*. The first cure: penalize the **squared** size of the weights — smooth, closed-form, our workhorse.

Big coefficients = a complex model



Two **full** degree-20 fits, same data; weights on standardized powers z_j ($\|\boldsymbol{\theta}\|_2 = \approx 13{,}000$ vs ≈ 19) — **penalizing size** is what buys the smooth fit:

$$\begin{aligned}\hat{y}_{\text{unres}} &= 9.9 + 30z_1 - 98z_2 - 576z_3 + 599z_4 + 3839z_5 - 239z_6 \\ &\quad - 7790z_7 - 2149z_8 + 3180z_9 + 1234z_{10} + 4956z_{11} + 2631z_{12} - 698z_{13} \\ &\quad + 359z_{14} - 3878z_{15} - 2446z_{16} - 1779z_{17} - 2340z_{18} + 2762z_{19} + 2462z_{20}\end{aligned}$$

$$\begin{aligned}\hat{y}_{\text{reg}} &= 9.9 + 14z_1 + 2.9z_2 + 10.3z_3 + 2.8z_4 + 6.2z_5 + 2.0z_6 + 3.5z_7 \\ &\quad + 1.3z_8 + 1.9z_9 + 0.8z_{10} + 1.0z_{11} + 0.4z_{12} + 0.6z_{13} + 0.2z_{14} \\ &\quad + 0.5z_{15} + 0.1z_{16} + 0.6z_{17} + 0.1z_{18} + 0.8z_{19} + 0.2z_{20}\end{aligned}$$

Ridge regression (L_2): the idea

- Without a penalty, a high-capacity model on finite data lets coefficients grow **huge** — chasing noise, not signal (we just saw $\|\boldsymbol{\theta}\| \approx 13{,}000$).
- The L_2 penalty $\lambda\|\boldsymbol{\theta}\|_2^2$ **resists** that: big coefficients pay a quadratic cost; small ones pay almost nothing.
- Net effect: less variance (we stop chasing noise), a sliver of bias (we shrink the signal a touch too). Usually a win.

Shrink coefficients toward 0 by penalizing their squared L_2 norm:

$$\hat{\boldsymbol{\theta}}_{\text{ridge}} = \arg\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2 = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 + \lambda \sum_{j=1}^{p} \theta_j^2$$

Ridge regression (L_2): closed form

Add $\lambda \mathbf{I}$ before inverting — and a **closed form** survives:

$$\hat{\boldsymbol{\theta}}_{\text{ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$$

- We add positive entries along the diagonal "ridge" of $\mathbf{X}^\top \mathbf{X}$ (hence the name — shown shortly).
- This also **fixes a singular $\mathbf{X}^\top \mathbf{X}$**: $(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})$ is invertible for any $\lambda > 0$, so ridge works even with more features than samples, or perfectly collinear features.

Bayesian view: plain OLS is the **MLE** (no prior). Adding the penalty = adding a **prior**, which turns it into a **MAP** estimate. Ridge's prior is **Gaussian** $\boldsymbol{\theta} \sim \mathcal{N}(0, \tau^2 \mathbf{I})$ — "weights small, centred at 0"; larger λ = tighter prior. (Lasso uses a different prior — compared later.)

Ridge from the optimizer's view: weight decay

Take one gradient-descent step (rate η) on the regularized risk, term by term:

$$\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|_2^2 \qquad \text{regularized objective}$$

$$\nabla\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \nabla\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + 2\lambda\boldsymbol{\theta} \qquad \text{gradient (penalty adds } 2\lambda\boldsymbol{\theta}\text{)}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\,\nabla\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) \qquad \text{the GD update}$$

$$= \boldsymbol{\theta} - \eta\big(\nabla\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + 2\lambda\boldsymbol{\theta}\big) \qquad \text{substitute the gradient}$$

$$= \boxed{\underbrace{(1-2\eta\lambda)\,\boldsymbol{\theta}}_{\text{shrink}} \;-\; \underbrace{\eta\,\nabla\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}_{\text{usual GD step}}} \qquad \text{regroup the } \boldsymbol{\theta} \text{ terms}$$

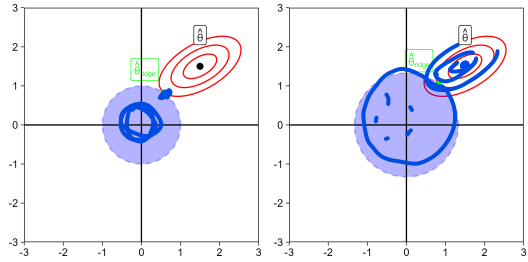
Shrink, then step: each weight is pulled toward 0 by the factor $(1-2\eta\lambda) < 1$ *before* the gradient update — exactly the deep-learning **weight decay**. ($\lambda = 0 \Rightarrow$ factor $1 \Rightarrow$ plain GD.)

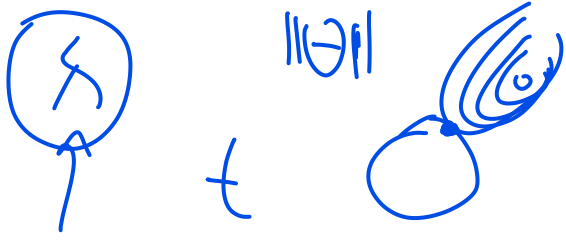
Ridge as constrained optimization

Equivalent constrained form (Lagrange duality):

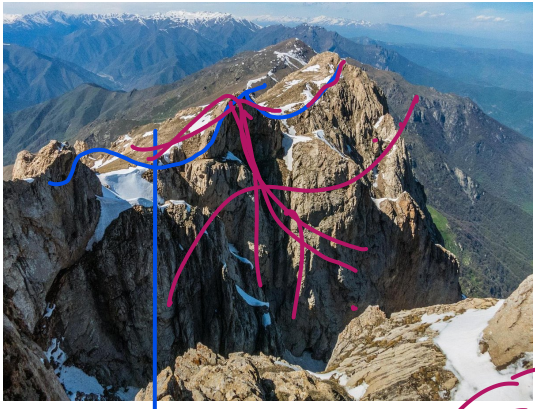
$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_2^2 \le t$$

- In the figure: the **ellipses** are contours of the least-squares loss (their centre is the OLS solution $\hat{\theta}$).
- The **circle** is the constraint region $\|\boldsymbol{\theta}\|_2 \le \sqrt{t}$ — the L_2 ball of radius $\sqrt{t}$.
- The ridge solution is where the smallest ellipse first **touches** the circle (on its boundary).
- Bijection: large $\lambda \Leftrightarrow$ small t (a tighter ball).





What is a "ridge"? (Mount Khustup)



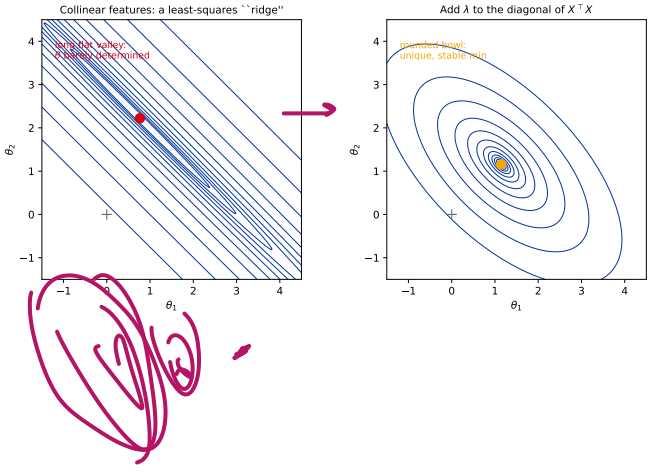
Mount Khustup (3206 m), Syunik, Armenia.

A mountain **ridge** is a long *crest* — walk along it and your height barely changes.

The least-squares loss for **correlated features** has the same shape: a long, near-flat direction where many different $\boldsymbol{\theta}$ give almost the same error.

OLS wanders along that crest (**unstable**). Ridge regression adds a penalty that pins down **one** point on it.

Ridge tames the collinear valley

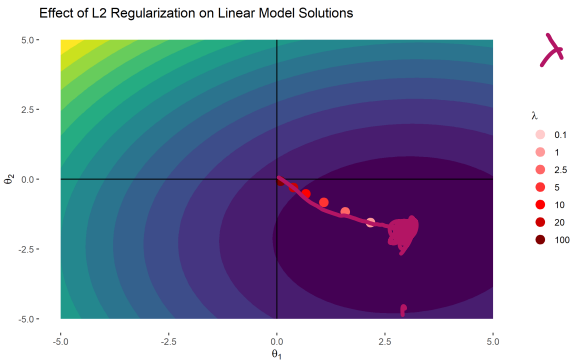


With **correlated features** the least-squares loss is a long, flat **valley** (a "ridge"): many very different $\boldsymbol{\theta}$ fit almost equally well, so OLS is **unstable** (here $\mathrm{cond}(\mathbf{X}^\top\mathbf{X}) \approx 800$).

Adding λ to the diagonal **rounds** the valley into a bowl with a single, stable minimum.

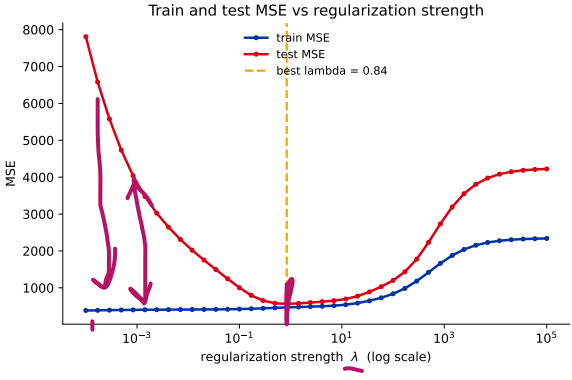
The name comes from Hoerl's *ridge analysis* of response surfaces (1962): seek the optimum on spheres about the origin — exactly the $\|\boldsymbol{\theta}\|_2 \leq t$ constraint.

Ridge solution path



- Toy DGP: $y = 3x_1 - 2x_2 + \varepsilon$. True optimum $\boldsymbol{\theta}^* = (3, -2)$.
- As $\lambda \uparrow$, $\hat{\boldsymbol{\theta}}_{\text{ridge}}$ is pulled toward the origin
- Contours = unregularized objective; red points = ridge solutions at increasing λ
- Path is **smooth and non-sparse**

The U-curve: test error vs. λ



Now that we know what λ does, sweep it. Same diagnostic as last week's **validation curve** (error vs. degree d) — but for λ.

- **Left** (small λ): high variance — the overfit panel.
- **Right** (large λ): high bias — the underfit panel.
- **Bottom of the U** ($\lambda \approx 0.84$): the sweet spot.

The U-shape *is* the bias-variance tradeoff made visible.

Predict first: scale your features before regularizing

Questions:

- You fit `Ridge(alpha=1)` on house data. `bathrooms` $\in [1, 4]$, `square_meters` $\in [50, 500]$. Both genuinely predict price. Which feature does Ridge **penalize harder**?
- Same setup, but you `StandardScaler()` first. What changes?

Answers:

- *Unscaled:* `bathrooms` is hit harder. To matter for y its coefficient must be $\sim 100\times$ larger — and the L_2 penalty punishes large coefficients quadratically. A genuinely useful feature gets shrunk on a *technicality of units*.
- *Scaled:* both features have unit variance, so the penalty is **fair** — Ridge chooses on signal strength, not on units.

Rule: always scale before Ridge/Lasso (`StandardScaler`). Fit the scaler on **train only**, then `.transform()` the test set — inside the CV loop.

Using Ridge in sklearn

```
from sklearn.linear_model import Ridge
Ridge(alpha=, fit_intercept=, solver=, max_iter=, tol=, random_state=)
```

arg	what it does	default	when to change
`alpha`	penalty strength λ (bigger = more shrinkage)	`1.0`	**tune by CV** — the knob that matters
`fit_intercept`	fits an intercept and does *not* penalize it	`True`	leave `True` (don't shrink the baseline)
`solver`	numerical method	`'auto'`	`'sparse_cg'`/`'lsqr'` for big sparse $\mathbf{X}$
`max_iter`, `tol`	budget/stopping tol for iterative solvers	`None`, `1e-3`	bump if it warns about convergence
`random_state`	seed for stochastic solvers (`sag`, `saga`)	`None`	set for reproducibility

Ridge shrinks every coefficient but zeros *none*. Sometimes you want a model that throws features away entirely — that is the next penalty.

Lasso regression (L_1)

Ridge shrinks every weight but **zeros none**. What if we want the model to *drop features entirely*? Switching the penalty to the L_1 norm does exactly that.

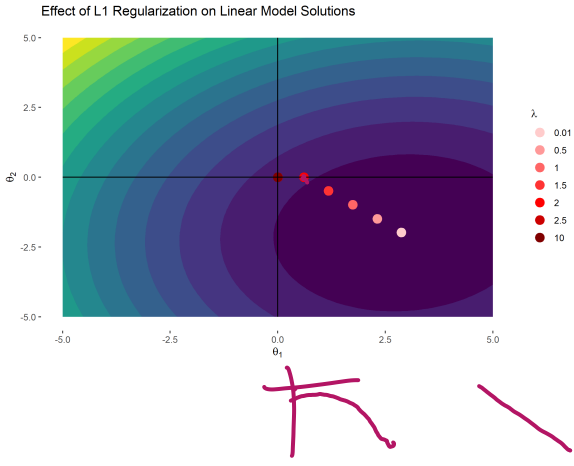
Lasso regression

Penalize the L_1 norm of the coefficients instead:

$$\hat{\boldsymbol{\theta}}_{\text{lasso}} = \arg\min_{\boldsymbol{\theta}} \ \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1 = \arg\min_{\boldsymbol{\theta}} \ \sum_{i=1}^{n}(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 + \lambda \sum_{j=1}^{p} |\theta_j|$$

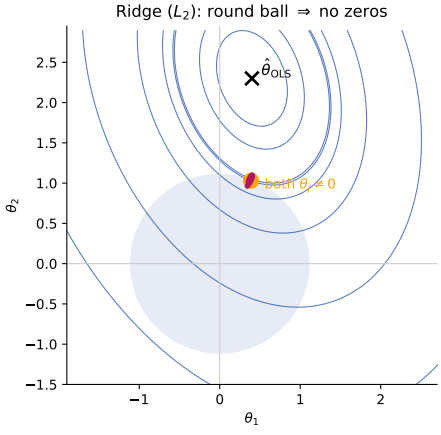
- "Least Absolute Shrinkage and Selection Operator"
- Still convex, but **not differentiable** at $\theta_j = 0 \Rightarrow$ no closed form
- Solve via coordinate descent, LARS, or proximal methods
- Key property: **sparsity** — many coefficients become *exactly* 0

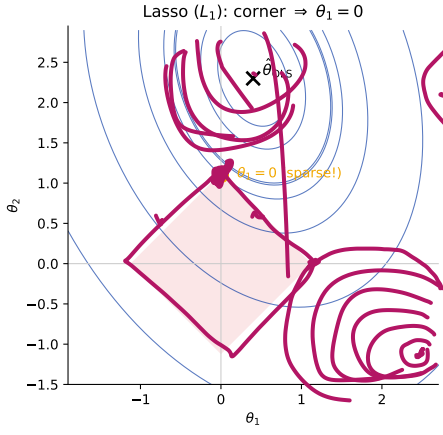
Lasso solution path



- Same toy DGP ($y = 3x_1 - 2x_2 + \varepsilon$)
- Lasso path takes a *different route* than ridge
- Coefficients can become **exactly zero** for large enough λ
- Side effect: built-in **feature selection**

Lasso vs ridge: geometric intuition



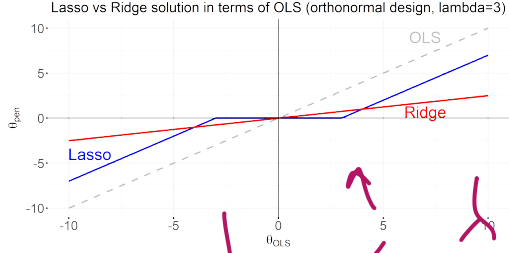


- Loss contours (centred at the OLS solution) grow until they **touch** the constraint region.
- **Ridge** (L_2 ball): the round boundary is touched *between* the axes — both $\theta_j \neq 0$.
- **Lasso** (L_1 diamond): contours hit a **corner** on an axis first — so $\theta_1 = 0$ (sparsity for free).

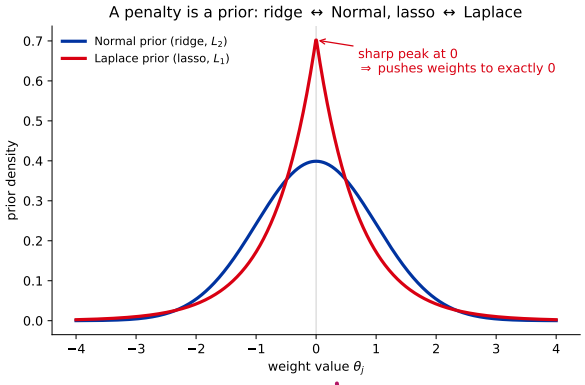
Soft thresholding (orthonormal $\mathbf{X}$)

In the special case $\mathbf{X}^\top \mathbf{X} = \mathbf{I}$, closed forms exist:

- **Lasso**: $\hat{\theta}_{\text{lasso}} = \text{sign}(\hat{\theta}_{\text{OLS}}) \cdot (|\hat{\theta}_{\text{OLS}}| - \lambda)_+$
  - "Soft thresholding": small coefs $\to 0$, large ones shrunk by λ
- **Ridge**: $\hat{\theta}_{\text{ridge}} = \hat{\theta}_{\text{OLS}}/(1+\lambda)$
  - Uniform downscaling, no sparsity



A probabilistic view: penalties are priors



Penalizing $\|\boldsymbol{\theta}\|$ = assuming a **prior** on the weights (MAP estimation):

- **Ridge** ↔ **Normal** prior: smooth, rounded at 0 — shrinks but rarely zeroes.
- **Lasso** ↔ **Laplace** prior: a **sharp peak** at 0 puts real probability mass on exactly-zero weights ⇒ **sparsity**.

Two lenses, one story: geometry (diamond vs ball) and probability (Laplace vs Normal) explain sparsity the *same* way.

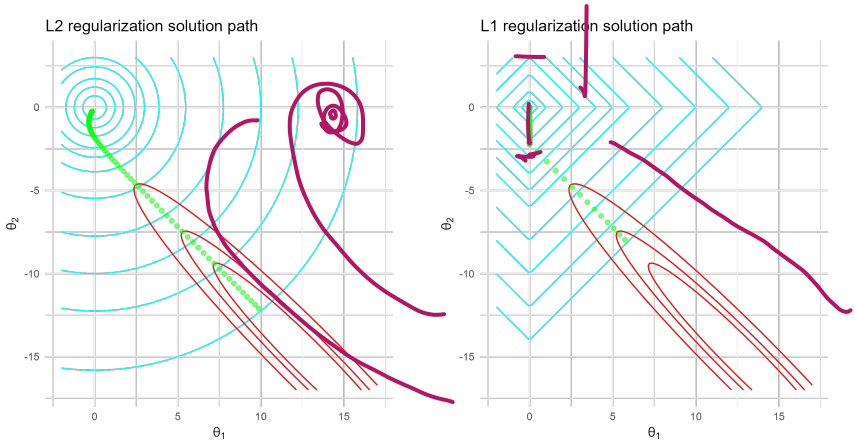
Using Lasso in sklearn

```
from sklearn.linear_model import Lasso
Lasso(alpha=, fit_intercept=, max_iter=, tol=, selection=, warm_start=)
```

arg	what it does	default	when to change
`alpha`	penalty strength λ for the L_1 term	`1.0`	**tune by CV**; larger $\rightarrow$ more zeros
`fit_intercept`	fits an unpenalized intercept	`True`	leave `True`
`max_iter`	coordinate-descent iterations	`1000`	bump on a "did not converge" warning (common!)
`tol`	stopping tolerance	`1e-4`	loosen/tighten alongside `max_iter`
`selection`	coordinate order: `'cyclic'`/`'random'`	`'cyclic'`	`'random'` $\rightarrow$ faster on wide data
`warm_start`	reuse previous solution	`False`	`True` when sweeping a path of αs

Same rule as Ridge: **scale first**. Lasso warns about convergence far more often — reach for `max_iter` before you panic.

Solution paths side-by-side



- **Ridge**: smooth, non-sparse path; all features stay in the model
- **Lasso**: piecewise-linear path; coefficients hit 0 one by one as λ grows

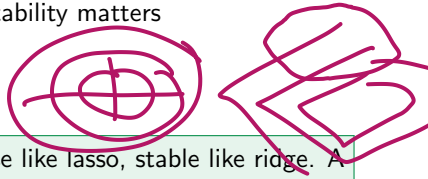
When to use which

Ridge (L_2):

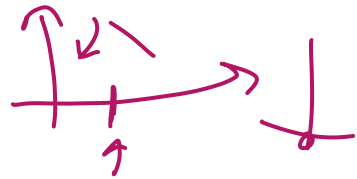
- Many small-to-moderate effects expected
- Multicollinearity present (helps numerically)
- Closed-form fit is desirable (cheap)
- You don't need exact zeros

Lasso (L_1):

- You suspect only a few features matter (sparse truth)
- You want automatic feature selection
- Interpretability matters

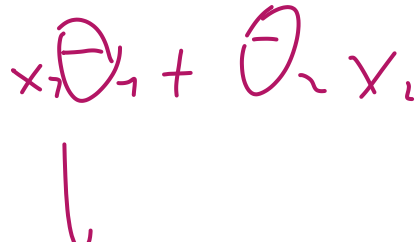


Elastic Net $= \lambda\big(\alpha\|\boldsymbol{\theta}\|_1 + (1-\alpha)\|\boldsymbol{\theta}\|_2^2\big)$: blends both — sparse like lasso, stable like ridge. A good default when in doubt.



Beyond L_1 and L_2

Penalties on linear weights are just one family. Every model has its own knobs — and some regularization needs *no penalty term at all*.



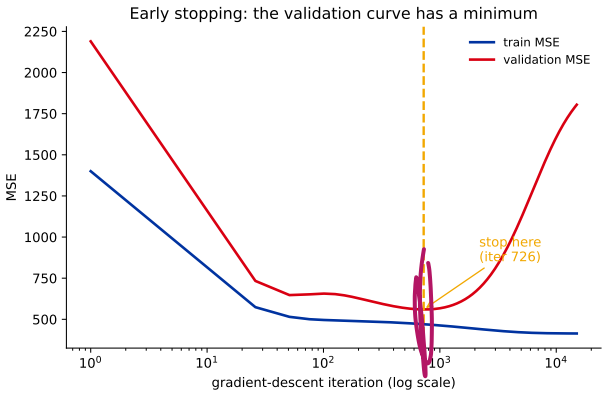
Different models, different regularizers

L_1/L_2 are the linear-model knobs. Every family has its own way to "stay simple":

Model family	**Typical regularization**
Linear / logistic	L_1, L_2, elastic net
Decision trees	`max_depth`, `min_samples_leaf`, cost-complexity pruning
Random forests / GBMs	tree depth, number of trees, learning rate, subsampling
Neural networks	weight decay (L_2), dropout, early stopping, data augmentation
k-NN	the number of neighbours k
SVM	the soft-margin penalty C

Same idea everywhere: **trade a little fit for a lot of stability**. The knob's name changes; the bias-variance logic does not.

Early stopping: regularization for free



Train with gradient descent and watch a **validation** set:

- Train error keeps falling; validation error falls, bottoms out, then **rises** (overfitting).
- **Stop at the validation minimum** ($\approx$ iter 730, val MSE $\approx$ 560 — near the noise floor).
- Run to convergence and it overfits (val MSE $\to$ 1800).

Fewer GD steps keeps the weights small — like L_2, but with **no penalty term** (implicit regularization).

Wrap-up

1. Big coefficients = complex model. Regularization penalizes size: $\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda J(\boldsymbol{\theta})$.
2. λ is a **continuous knob** on the bias-variance tradeoff — the U-curve has a sweet spot (pick it by CV).
3. **Ridge** (L_2): smooth shrinkage, closed-form, Gaussian prior, $\equiv$ weight decay. No sparsity.
4. **Lasso** (L_1): kinked, Laplace prior, sparse — feature selection for free.
5. Other knobs: tree depth, dropout, **early stopping**. And always **scale features first**.

We said "tune λ by CV" — **next** we make that practical: **hyperparameter tuning** (grid / random / Optuna), then a closer look at **which loss** we even minimize (MSE / MAE / Huber / quantile).