RIDGE, LASSO, AND TREES

Kerry Back

Tue Sep 8, 2026

Empirical Asset Pricing via Machine Learning

The Review of Financial Studies 33 (2020), 2223–2273. Submitted September 2018, accepted September 2019, published February 2020.

Shihao Gu

Booth School of Business, University of Chicago

Bryan Kelly

Yale University; Head of Machine Learning, AQR

Dacheng Xiu

Booth School of Business, University of Chicago

Trees and neural networks win, and the gains come from interactions a linear model cannot see. A value-weighted long-short decile spread on their neural network forecasts earns an out-of-sample Sharpe ratio of 1.35. The paper and a short video are in this session's files.

Today

1. Least squares, and what it does with a short sample
2. Ridge and lasso — the same regression with a penalty on the coefficients
3. Regression trees, and gradient boosting them
4. Choosing the knob without looking at the test set

> Four apps in this deck. Every slider refits the model in the browser and redraws the test-set error.

Root Mean Squared Error

A standard deviation measures spread around a mean. RMSE measures spread around a prediction, and it is the same arithmetic.

STANDARD DEVIATION OF A SAMPLE

$$s_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

ROOT MEAN SQUARED ERROR

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

> Each point's own prediction $\hat{y}_i$ takes the place of the one number $\bar{x}$. Both come out in the units of the variable — here, logs of P/S.

R-Squared

The squared error you removed, over the squared error you started with.

$$R^2 = \frac{\frac{1}{n}\sum_i (y_i - \bar{y})^2 - \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2}{\frac{1}{n}\sum_i (y_i - \bar{y})^2}$$

> The denominator is what you would suffer predicting $\bar{y}$ for everyone. The term subtracted from it is RMSE squared. The $1/n$ cancels — it is written in so that both pieces are the averages on the last slide.

One Small Example

Everything today is fitted to the same 40 firms and scored on the same 120, drawn from `session4.parquet`.

Left-hand side	log of the price-to-sales ratio
Features	the eight ratios from Thursday — margins, turnover, R&D, liquidity, growth, leverage
Firms	market cap above $2b, 40 fitted and 120 held out

> Forty is small on purpose. Everything that goes wrong with a thousand features and ten thousand rows goes wrong here too, and here you can watch it happen.

LEAST SQUARES

What OLS Is Doing

Pick the coefficients that make the sum of squared errors on the training data as small as possible. Nothing else is asked of them.

0.68
TRAIN R-SQUARED

0.44
TEST R-SQUARED

8
FEATURES, 40 FIRMS

> The fit was chosen to make the first number large. The second is what it is worth on firms it has not seen, and the gap between them is the whole subject.

Net margin and EBITDA margin have a correlation of 0.82 across these firms. Least squares is asked which of them explains P/S, and the data cannot say.

	OLS COEFFICIENT
net margin	+0.63
EBITDA margin	−0.27

> A large positive on one and a negative on its near-twin. The pair predicts almost nothing between them, and the sign on the second is an accident of these 40 firms.

Features That Move Together

RIDGE

The Ridge Criterion

$$\min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_j \beta_j x_{ij} \right)^2 + \lambda \sum_j \beta_j^2$$

The second term

A price on size. Large coefficients now have to earn their keep.

Standardize first

One penalty for all the coefficients only works if the features share a scale.

Not the intercept

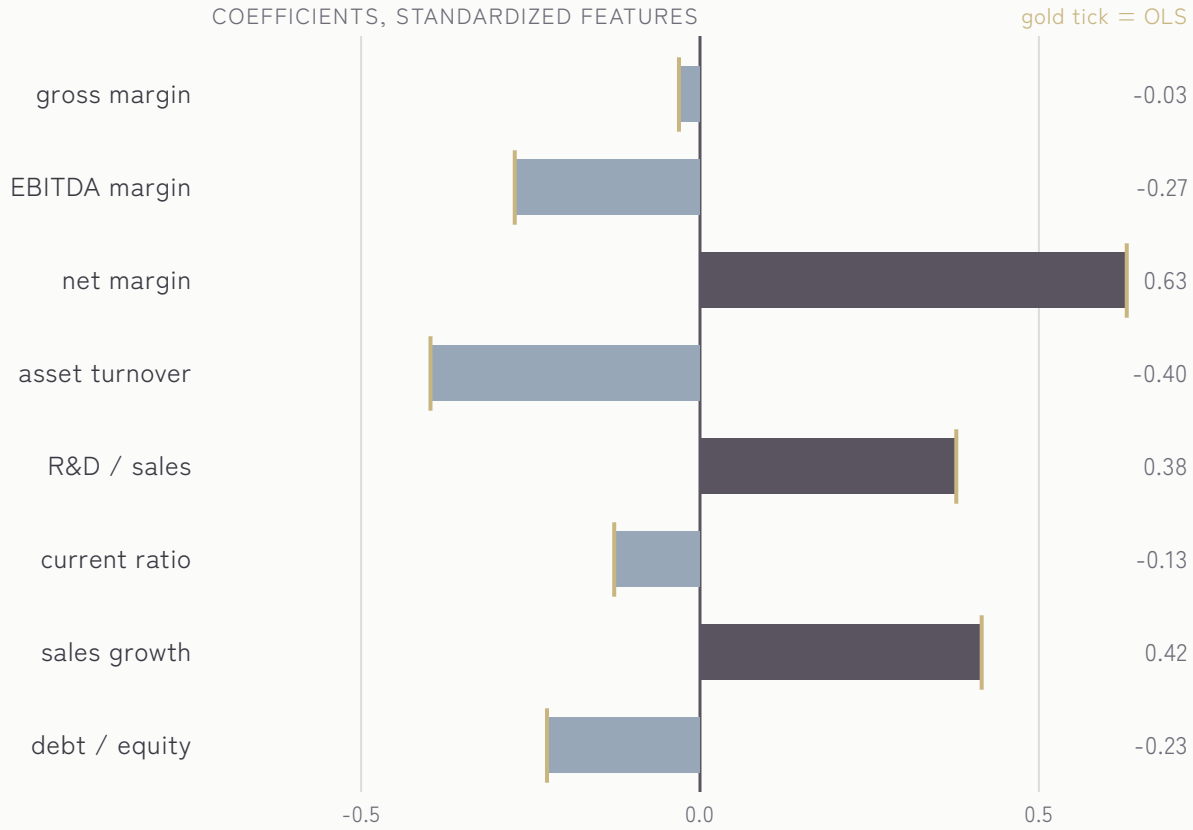
β_0 is left out of the sum. Shrinking it would just move the whole fit.

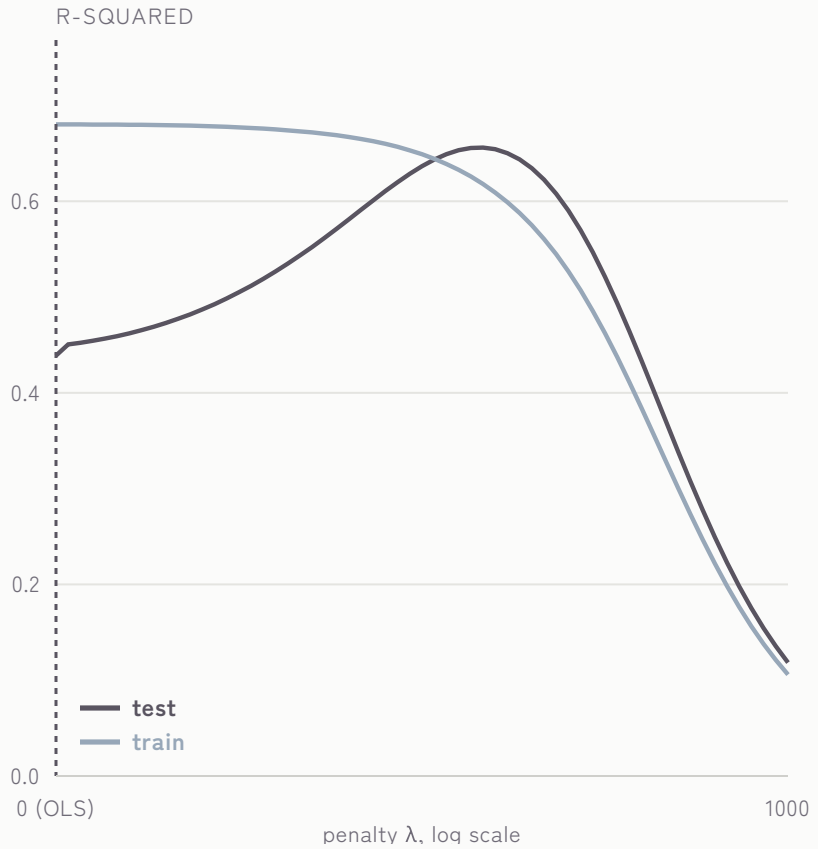
λ is a hyperparameter — a number you set before fitting, rather than one the fitting picks for you. The β's are parameters and come out of the data; λ decides how hard the data has to argue for them.

Ridge

RIDGE PENALTY $\lambda = 0$

0.63	0.68	0.44
LARGEST |B|	TRAIN R^2	TEST R^2





Everything shrinks

Toward zero, together. At $\lambda = 0$ the fit is exactly OLS; the gold ticks are where it started.

The twins converge

$+0.63$ and -0.27 become $+0.15$ and $+0.13$. Ridge splits the credit rather than picking a winner.

Nothing reaches zero

Ridge shrinks every coefficient and drops none. All eight features stay in the model.

> Test R-squared climbs from 0.44 to 0.66 and then falls away again. The training R-squared drops the whole way.

What the Slider Showed

LASSO

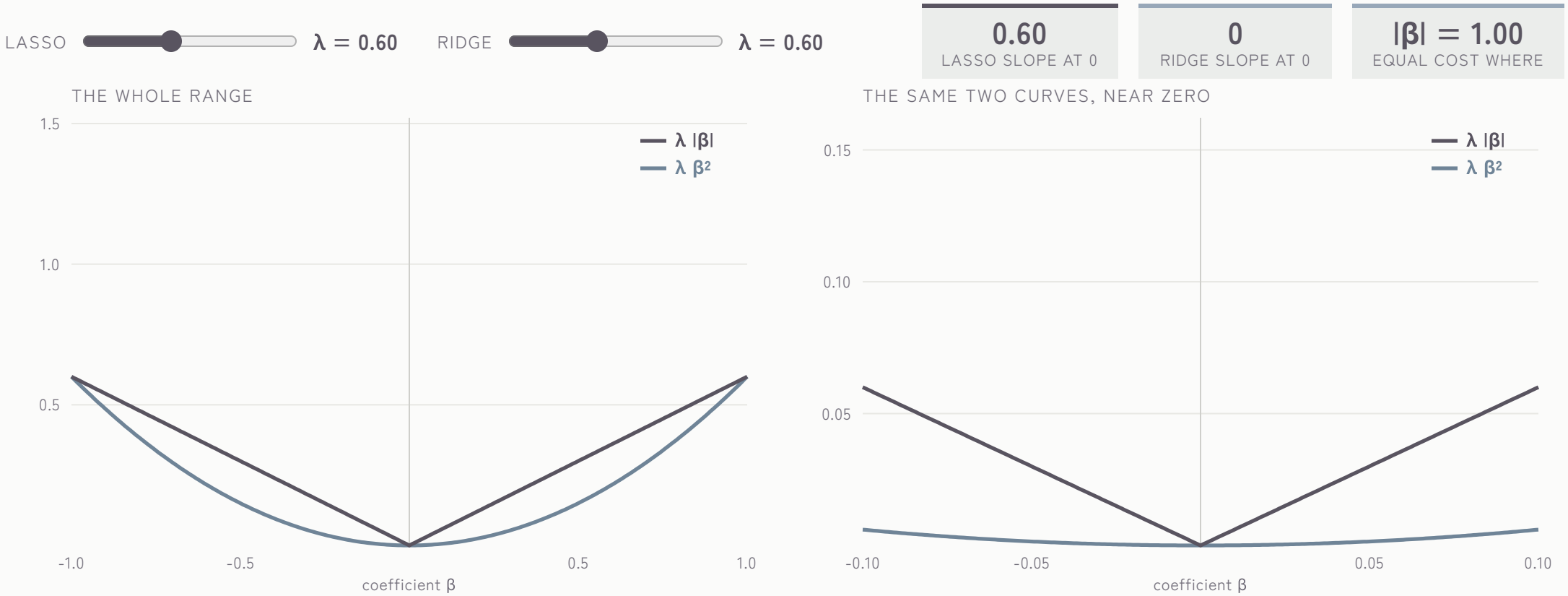
The Lasso Criterion

$$\min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_j \beta_j x_{ij} \right)^2 + \lambda \sum_j |\beta_j|$$

One character different, and the behavior is not the same. Squaring makes the penalty on a small coefficient negligible; the absolute value does not.

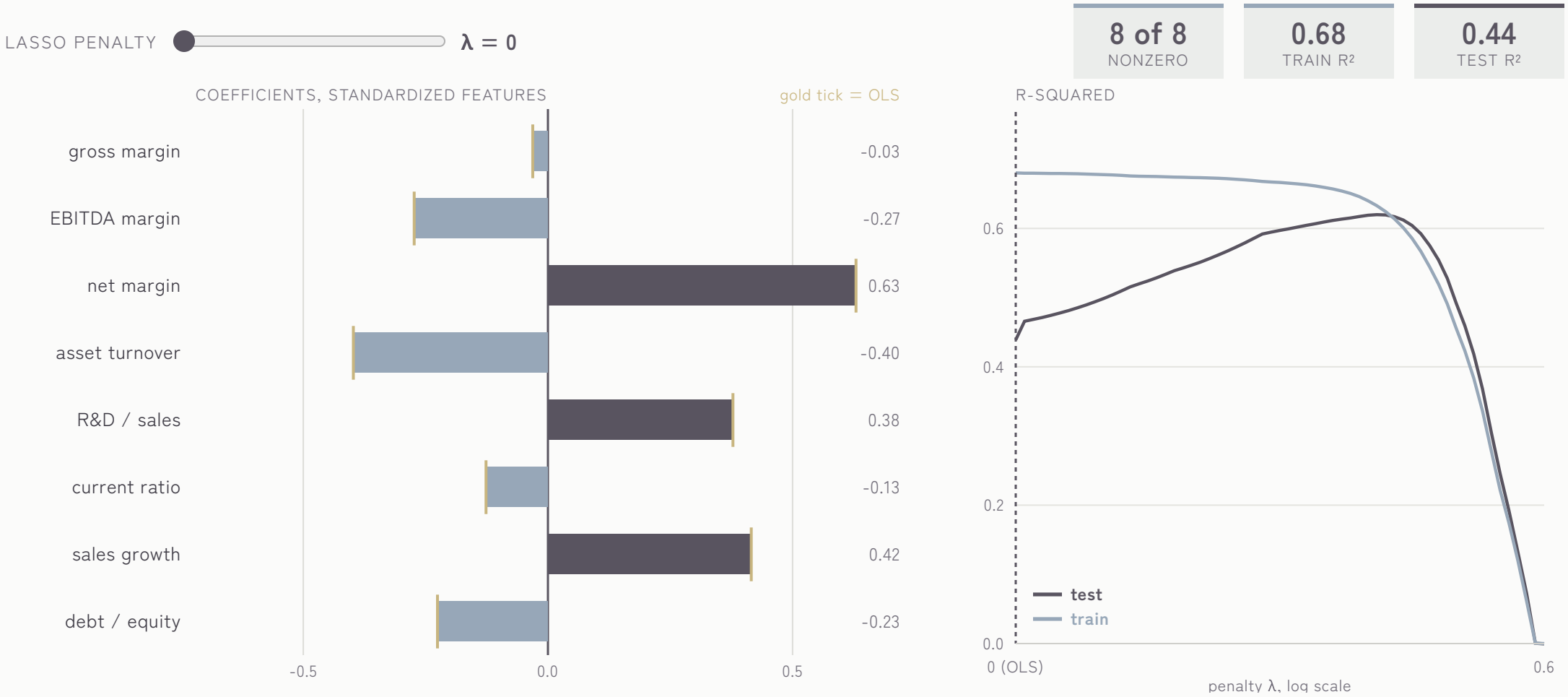
> A coefficient that is not paying for itself is pushed all the way to zero and stays there. Lasso fits and selects in one step.

The Shape of the Penalty

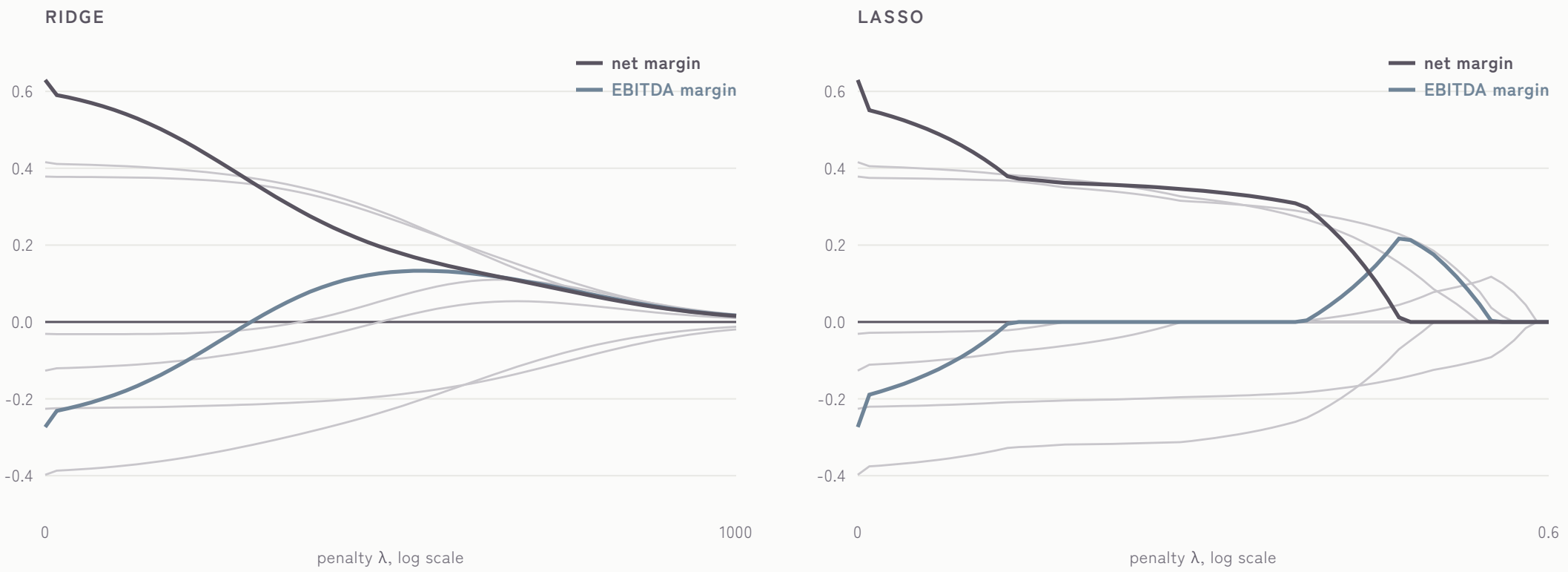


Near zero the square has nothing left to charge; the absolute value still charges full price.

Lasso


LASSO PENALTY
λ = 0
8 of 8
NONZERO
0.68
TRAIN R²
0.44
TEST R²
COEFFICIENTS, STANDARDIZED FEATURES
gold tick = OLS
gross margin
EBITDA margin
net margin
asset turnover
R&D / sales
current ratio
sales growth
debt / equity
-0.03
-0.27
0.63
-0.40
0.38
-0.13
0.42
-0.23
-0.5
0.0
0.5
R-SQUARED
0.6
0.4
0.2
0.0
test
train
0 (OLS)
0.6
penalty λ, log scale

Ridge Shrinks, Lasso Drops



Every coefficient, plotted against the penalty. The two correlated margins are picked out; the other six are gray.

REGRESSION TREES

How a Tree Fits

1. Search every feature and every cut point for the one split that reduces the sum of squared errors the most.
2. Cut the sample in two on that rule.
3. Repeat inside each piece, and inside the pieces of those pieces.
4. Predict the average of the training firms that land in each final piece.

> No linear form is assumed, so a tree handles a kink or a threshold that a regression has to be told about. The next two slides use gross margin and net margin, because two is the most you can draw.

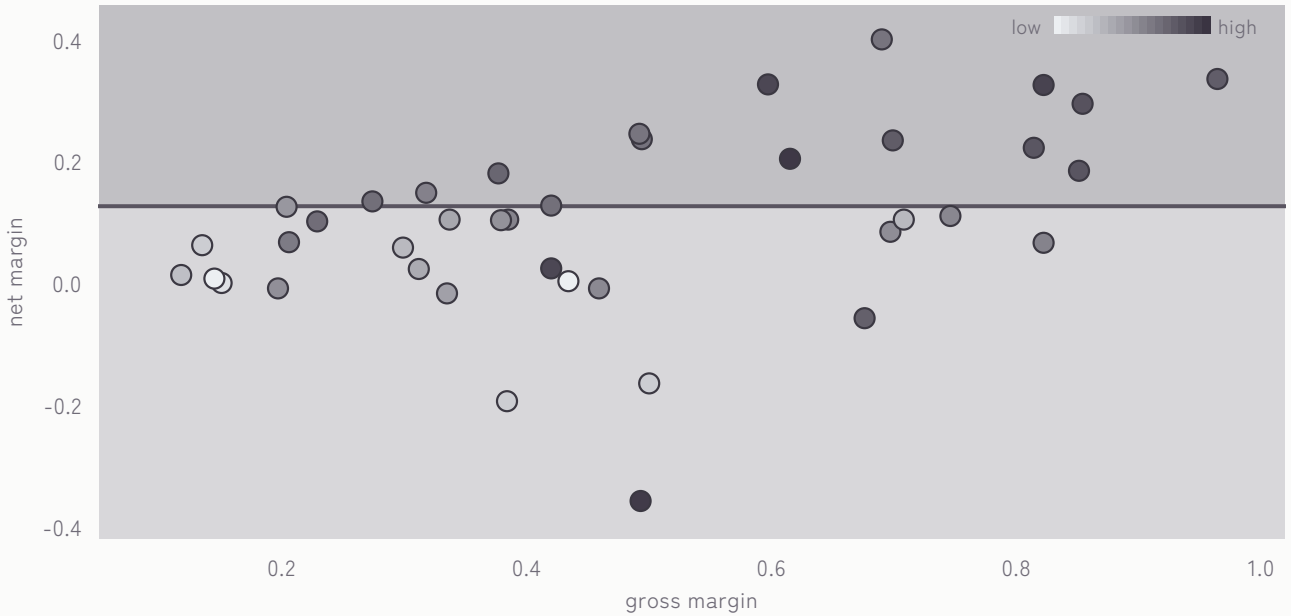
Splitting on Two Features

SPLITS MADE 1

2
PIECES

0.37
TRAIN R^2

THE 40 TRAINING FIRMS — COLOUR IS LOG P/S



THE CUTS, IN THE ORDER THEY WERE MADE

1. net margin ≤ 0.128

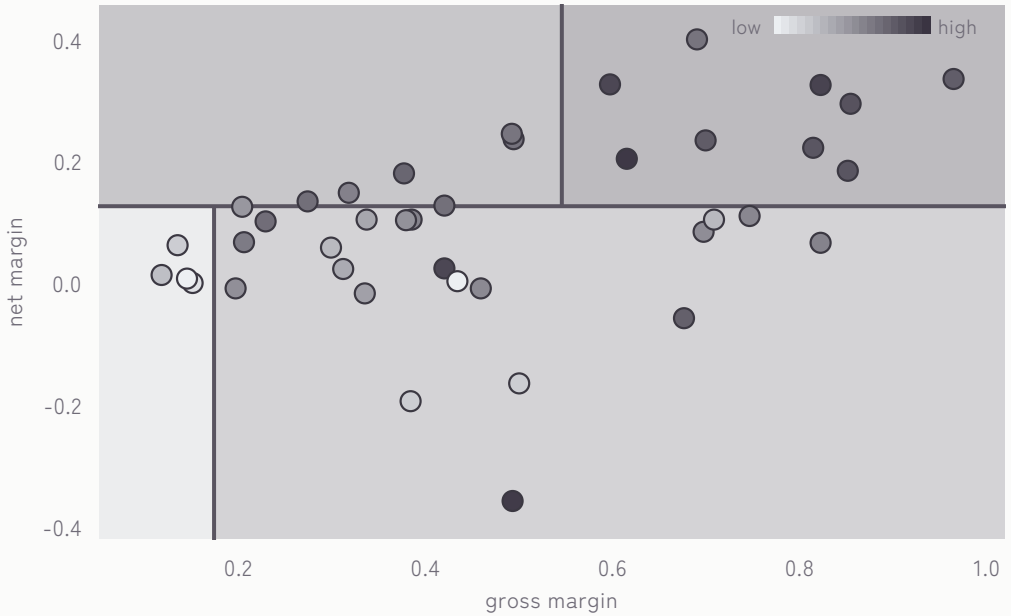
Each cut is chosen over both variables at once. The winner is whichever single line, in whichever variable, buys the most.

One Tree

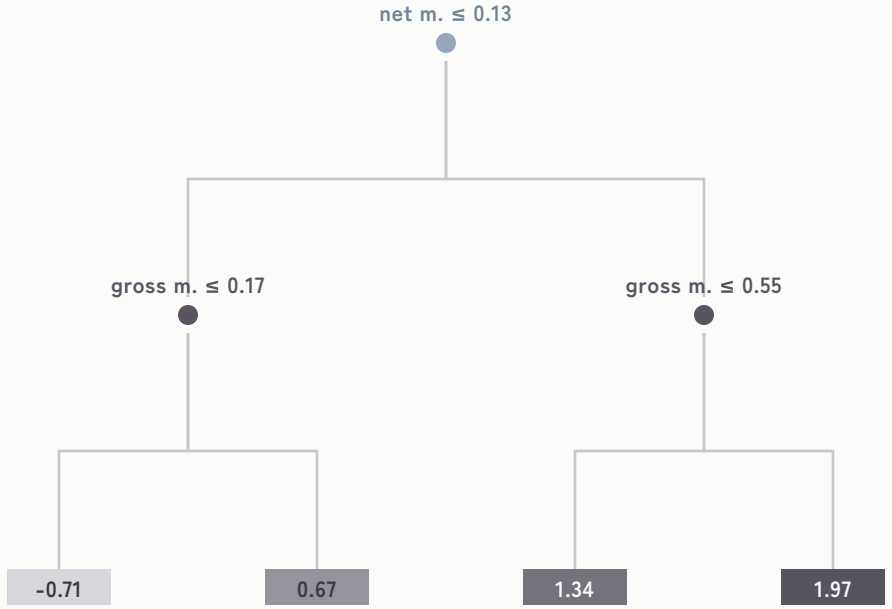
DEPTH 2 MIN PER LEAF 1

4	0.57	0.29
LEAVES	TRAIN R^2	TEST R^2

THE SAMPLE, CUT INTO BOXES



THE SAME THING, AS A TREE



The boxes and the tree are the same object. A path down the tree is a box; a leaf's colour is the average of the firms that landed in it.

Max depth

How many cuts deep the tree may go. Depth d allows up to 2^d leaves; depth 8 here gives 28 boxes for 40 firms.

Minimum per leaf

The smallest number of training points a leaf may hold. A leaf of one is a memorized observation.

At depth 2 the tree scores 0.57 on the training firms and 0.29 on the test firms. At depth 8 it scores 0.96 and -0.14 — worse than a flat line at the average.

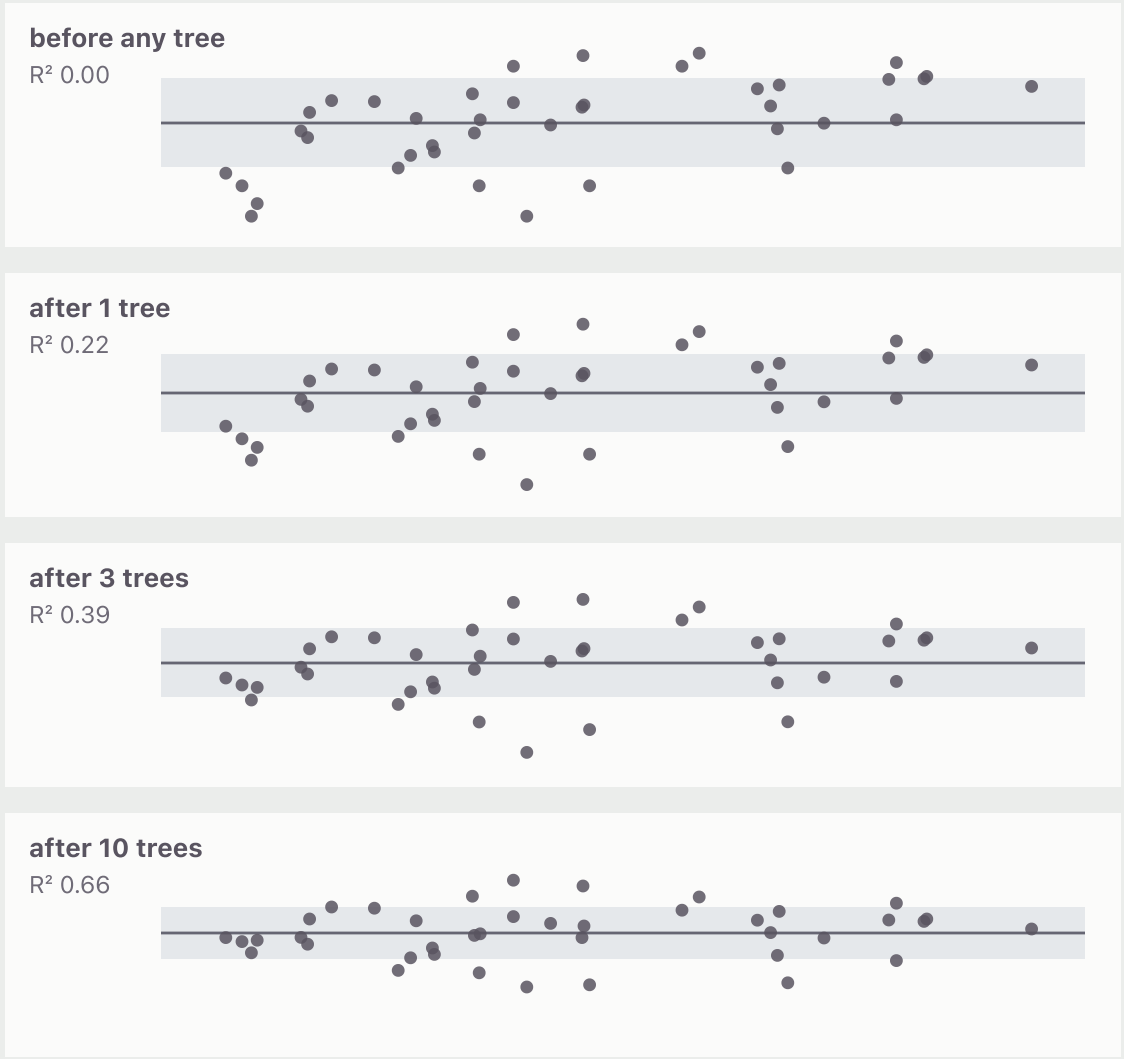
Two Ways to Stop It

GRADIENT BOOSTING

The Idea

1. Start by predicting the training mean for everyone.
2. Compute the residuals — what the current prediction is missing.
3. Fit a small tree to the residuals.
4. Add a fraction of that tree, the learning rate, to the prediction.
5. Go back to step 2.

> No single tree is any good. The sum of a few hundred deliberately weak ones is the model that wins most tabular prediction contests. Back to one feature here, so the sum can be drawn as a curve.



RESIDUALS AGAINST GROSS MARGIN

Fitting the Residuals

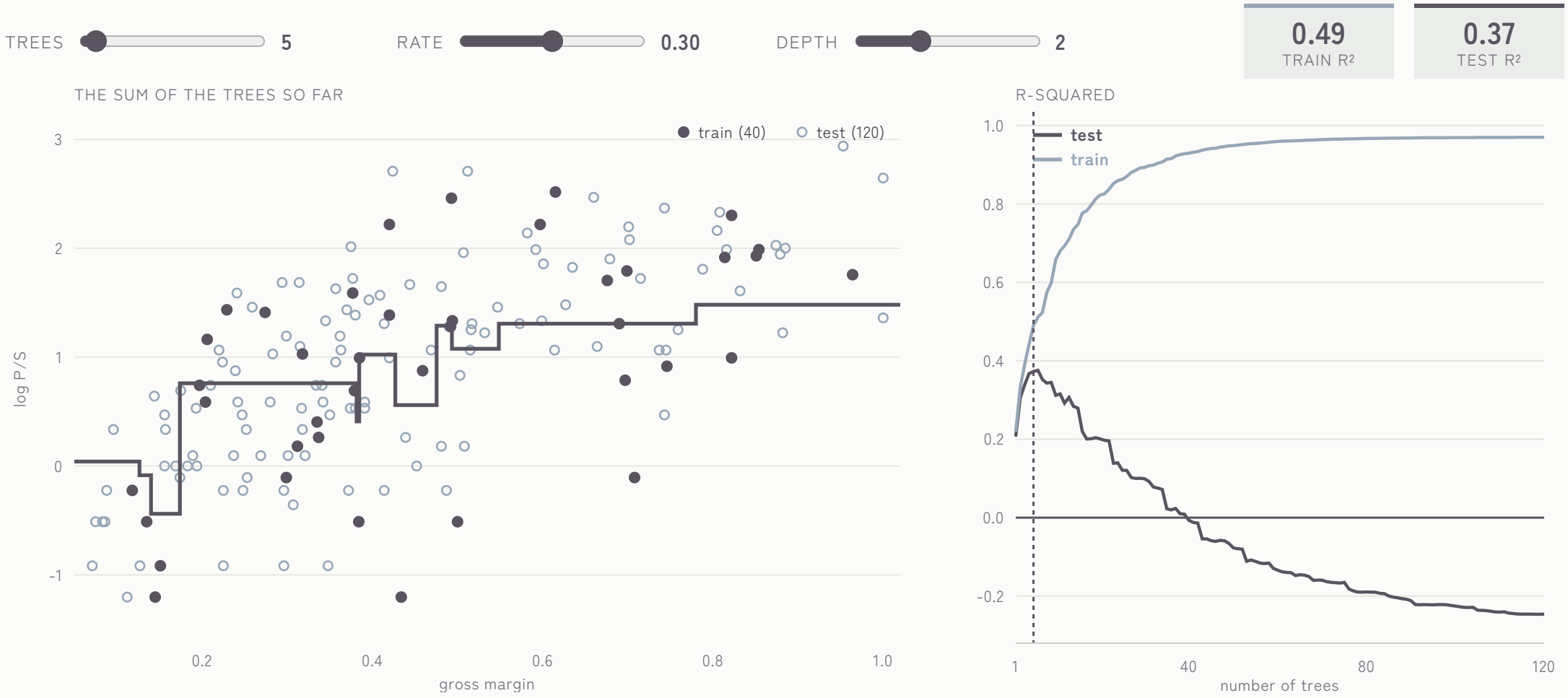
WHAT EACH TREE IS GIVEN

Every tree after the first is fitted to what the ones before it got wrong.

What rises is the R-squared: nothing before any tree, 0.66 after ten.

No one of these trees is a model. The sum of them is.

Gradient Boosting



Number of trees

The only knob that keeps improving the training fit forever. Test R-squared peaks early — here, after about five.

Learning rate

How much of each tree is added. Small rates need more trees and usually end up better.

Depth

How much each tree may do on its own. Two or three is standard; the boosting is supposed to do the work.

> Halve the learning rate and you roughly double the number of trees you need. Set the rate low, then stop adding trees when validation R-squared stops rising.

Three Knobs, Not Independent

Every Boosting Hyperparameter

HOW MUCH ENSEMBLE

- `n_estimators` — how many trees
- `learning_rate` — shrinkage per tree
- `subsample` — rows per tree
- `loss` — squared, absolute, huber
- `n_iter_no_change` — early stopping

HOW MUCH TREE

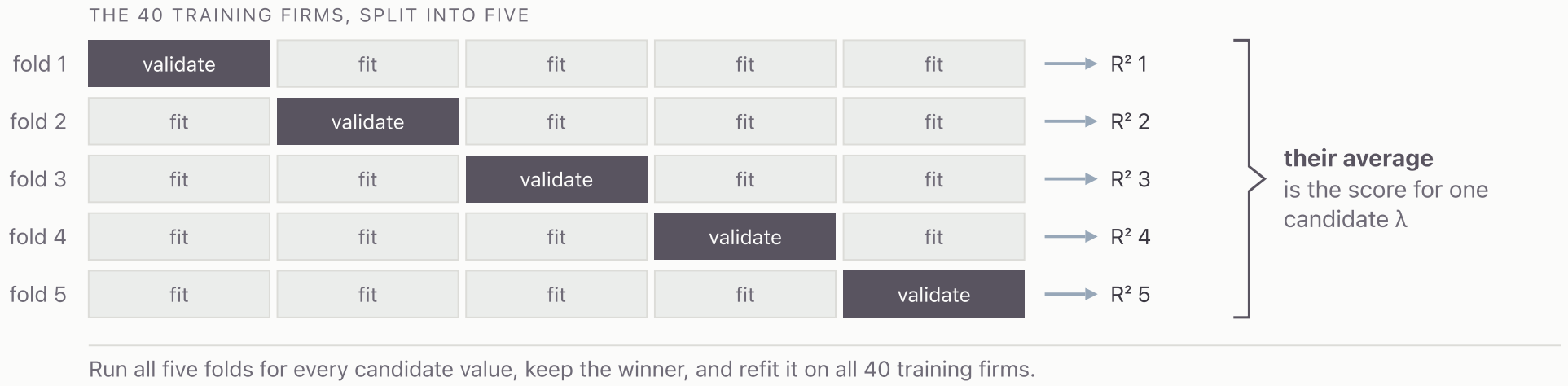
- `max_depth` — how deep
- `max_leaf_nodes` — or how many
- `min_samples_leaf` — smallest box
- `min_samples_split` — node minimum
- `max_features` — features per split

> All are set before fitting. Tune the first three and leave the rest alone.

CHOOSING THE KNOB

Cross-Validation

Split the training sample into 5 subsets. For each hyperparameter value, fit on 4, test on the 5th. Repeat across all 5 folds. Choose the hyperparameter with the best average performance on the 5 test folds.



The Same Shape Every Time

Every app in this deck drew the same picture. Training R-squared rises as the model is allowed more freedom; test R-squared rises, peaks, and falls away.

	KNOB	READ OFF THE TEST CURVE	TEST R-SQUARED
OLS	—	—	0.44
Ridge	$\lambda \approx 20$	shrunk, all 8 kept	0.66
Lasso	$\lambda \approx 0.08$	7 of 8 kept	0.62
Tree	depth 2	4 boxes	0.29
Boosting	5 trees, rate 0.3	depth 2	0.37

> Every one of those knobs was set by looking at the test curve, which is the one thing you are not allowed to do.

	FEATURES	CROSS-VALIDATION PICKED	TEST R-SQUARED
Ridge	8	$\lambda = 14$	0.65
Lasso	8	$\lambda = 0.014$	0.55
One tree	2	depth 2	0.29
One tree	8	depth 4	0.32
Boosting	1	25 trees, rate 0.1	0.35
Boosting	8	400 trees, rate 0.1	0.61
Random forest	8	300 trees, 3 features a split	0.69

> The small-feature rows are the versions the apps draw. Given all eight, boosting nearly catches ridge — and a random forest, which is only many deep trees averaged, wins.

Done Properly

Hands On

1. Use `session4.parquet` and do a train/test split.
2. Train `LinearRegression`.
3. Use `GridSearchCV` to find the optimal `n_estimators`, `learning_rate` and `max_depth` hyperparameters for `GradientBoostingRegressor`.
4. Report the R^2 on the test sample for both methods.

> Boosting does not need the features standardized — a tree only asks whether a value is above or below a cut point, and that answer does not change with the units. Ask AI which model it expects to win, and make it say why.

1. Refit `LinearRegression` on the entire sample.
2. Rerun `GridSearchCV` for `GradientBoostingRegressor` on the entire sample.
3. Save both refit models.

> We will use the models to build Stock Recommender apps on Thursday. Ask AI to save them with `joblib.dump` and to check that they load back.

Refit and Save Your Models