

Regularization

Introduction

Where we are:

- [Generalization](#): a model that fits its training data too closely – it **overfits** – predicts new data poorly.
- [Lecture 10](#): **least squares**, solved through the normal equations $X^T X \beta = X^T \mathbf{y}$.
- [Lecture 11A](#): **logistic regression**, fit by maximum likelihood.

Both fits choose β to minimize a loss, and both can overfit.

Overview

Today:

- the situation where regression needs help – **multicollinearity** – and how the **condition number** diagnoses it,
- ridge and LASSO: what each penalty does to the coefficients and when to use which,
- the two rules that come with any penalty: **standardize first**, and choose the hyperparameter c by **cross-validation**,
- and finally the same penalty applied to the logistic model from [Lecture 11A](#).

Overfitting

We have referenced the concept of overfitting previously in our [generalization](#) and [decision tree](#) lectures.

We know that a model that overfits does not generalize well. In other words, the model does not perform well on data it was not trained on.

We observe overfitting when the training accuracy of the model is high, while the accuracy on the validation set (or held out test set) stagnates.

In this lecture we will discuss **regularization**. A technique to help prevent overfitting. We will see how to apply regularization to regression problems.

However, regularization is a general technique to ensure models learn broad patterns and is applicable in neural networks.

What is regularization?

We know regularization is a way to prevent overfitting, but how does this actually work?

The idea behind regularization is to penalize a model's loss function during training. The added penalty will discourage the model from becoming too complex (i.e., overfitting).

In regression, our training process was to compute coefficients β by minimizing

$$\min_{\beta} \|X\beta - \mathbf{y}\|_2^2,$$

where, as in the [linear regression](#) lecture, $X \in \mathbb{R}^{n \times p}$ is the design matrix with one row per observation (n observations, p features) and $\mathbf{y} \in \mathbb{R}^n$ holds the dependent variable.

What is regularization?

Regularization adds a function $R(\beta)$ to the minimization problem. A regularized minimization problem is then: compute β such that

$$\min_{\beta} \|X\beta - \mathbf{y}\|_2^2 + cR(\beta).$$

The regularization coefficient c is a hyperparameter that controls the importance of the regularization term $R(\beta)$.

We will consider two common forms of $R(\beta)$:

- $R(\beta) = \|\beta\|_2^2$, called **ridge regression**, and
- $R(\beta) = \|\beta\|_1$, called **LASSO** regression.

Multicollinearity

In statistics, multicollinearity (also collinearity) is a phenomenon in which two or more predictor variables in a multiple regression model are highly correlated, meaning that one can be linearly predicted from the others with a substantial degree of accuracy.

We will see that multicollinearity can be problematic in our regression models.

To address these issues, we will consider

- What are the potential sources for multicollinearity?
- What does multicollinearity tell us about our data?

Understanding these questions will inform us how regularization can be used to mitigate the issue of multicollinearity.

Sources of Multicollinearity

We will be working with our design matrix $X \in \mathbb{R}^{n \times p}$, where each row corresponds to an observation and the p columns are the p features.

We will see that multicollinearity arises when the columns of X are linearly dependent (or nearly linearly dependent).

As a consequence, the matrix $X^T X$ is no longer invertible. However, as we saw in our [Linear Regression](#) lecture, the least squares solution still exists. It is just not unique.

Sources of Multicollinearity

What are the situations where this can happen?

One clear case is when $n < p$, which means X has more columns than rows. That is, there are **more features than there are observations** in the data.

However, we can still observe multicollinearity when $n > p$.

This can happen when the columns of X happen to be linearly dependent because of the nature of the data itself. In particular, this happens when one column is a linear function of the other columns. This means that one independent variable is a linear function of one or more of the others.

Unfortunately, in practice we will run into trouble even if variables are *almost* linearly dependent.

The Longley Dataset

To illustrate the multicollinearity problem, we'll load a standard dataset.

The [Longley](#) dataset contains various US macroeconomic variables from 1947–1962.

We have the following features:

- GNP - GNP (Gross National Product)
- GNPDEFL - GNP deflator, a way to account for inflation over different years
- UNEMP - Number of unemployed
- ARMED - Size of armed forces
- POP - Population
- YEAR - Year (1947 - 1962)

We want to predict:

- TOTEMP - Total Employment

The Longley Dataset

► Code

```
X.head()
      GNPDEFL      GNP  UNEMP  ARMED      POP  const
YEAR
1947.0      83.0  234289.0  2356.0  1590.0  107608.0    1.0
1948.0      88.5  259426.0  2325.0  1456.0  108632.0    1.0
1949.0      88.2  258054.0  3682.0  1616.0  109773.0    1.0
1950.0      89.5  284599.0  3351.0  1650.0  110929.0    1.0
1951.0      96.2  328975.0  2099.0  3099.0  112075.0    1.0
```

```
y.head()
YEAR
1947.0    60323.0
1948.0    61122.0
1949.0    60171.0
1950.0    61187.0
1951.0    63221.0
Name: TOTEMP, dtype: float64
```

Fitting Longley by least squares

Read the notes under the table: statsmodels warns that the condition number is large.
What does this mean?

► Code

```
=====
                        OLS Regression Results
=====
Dep. Variable:          TOTEMP      R-squared:                0.987
Model:                  OLS        Adj. R-squared:             0.981
Method:                 Least Squares    F-statistic:           156.4
Date:                  Sat, 17 Oct 2026    Prob (F-statistic):     3.70e-09
Time:                  20:04:10          Log-Likelihood:         -117.83
No. Observations:      16              AIC:                   247.7
Df Residuals:          10              BIC:                   252.3
Df Model:               5
Covariance Type:       nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
GNPDEFL	-48.4628	132.248	-0.366	0.722	-343.129	246.204
GNP	0.0720	0.032	2.269	0.047	0.001	0.143
UNEMP	-0.4039	0.439	-0.921	0.379	-1.381	0.573
ARMED	-0.5605	0.284	-1.975	0.077	-1.193	0.072
POP	-0.4035	0.330	-1.222	0.250	-1.139	0.332
const	9.246e+04	3.52e+04	2.629	0.025	1.41e+04	1.71e+05

Condition Number

The notion of conditioning pertains to the perturbation behavior of a mathematical problem. A well-conditioned problem is one where small changes to the inputs produce small changes to the output. An ill-conditioned problem is one where small changes in the input can produce very large changes in the output.

The condition number of a matrix provides an indication of how accurately you can compute with it.

A large condition number tells us that our problem is ill conditioned, i.e., small changes to the input can produce very large changes in the output.

A small condition number tells us that our problem is well-conditioned.

Condition Number

The condition number is derived using matrix norms, which is beyond the scope of this course. However, we will use the following definition for the condition number of our matrix

$$\kappa(X) = \frac{\sigma_{\max}}{\sigma_{\min}},$$

where $\sigma_{\max}$, $\sigma_{\min}$ are the maximum and minimum singular values, respectively, of the design matrix X .

The SVD – next lecture – again provides us with important properties of a matrix.

Condition Number

Another important fact is that

$$\kappa(X^T X) = \frac{\sigma_{\max}^2}{\sigma_{\min}^2} = \kappa(X)^2.$$

This means that if we have a poorly conditioned data matrix X , then $X^T X$ is even more poorly conditioned.

This is why, in practice, you should not *form* $X^T X$ and invert it. The normal equations describe the solution; good solvers (`np.linalg.lstsq`, `sm.OLS`) reach the same $\hat{\beta}$ by working with X directly, at condition number $\kappa(X)$ rather than $\kappa(X)^2$.

Normal Equations: a callback

From the [linear regression](#) lecture: the least-squares solution satisfies the normal equations $X^T X \beta = X^T \mathbf{y}$, which always have at least one solution.

- The solution is **unique** exactly when $X^T X$ is invertible, i.e. when the columns of X are linearly independent.
- If the columns are **linearly dependent**, $X^T X$ has a zero singular value, $\kappa(X^T X) = \infty$, and $(X^T X)^{-1}$ does not exist. There are then infinitely many β with the *same* fitted values $X\beta$ and the same residual norm: the **fit** is determined, the **coefficients** are not.
- This is the simplest kind of multicollinearity. It is a problem of interpretation, not of prediction.

Near Linear Dependence

Near linear dependence causes problems as well. This can happen, for example, due to measurement errors. Or when two or more columns are **strongly correlated**.

In such a situation, we have some column of our design matrix that is **close to** being a linear combination of the other columns.

When these situations occur we will have problems with linear regression.

Near Linear Dependence

As a result of near linear dependence, the smallest singular value of the design matrix X will be close to zero. This means that $\kappa(X)$ will be very large.

The condition number tells us that a small change in the input to our problem can result in large changes to the output.

This means that for a design matrix X with near linearly dependent columns, the values we compute for β in our linear regression can vary significantly.

Put in the language of [Generalization](#): an unstable $\hat{\beta}$ is **variance**. Refit on a slightly different sample and the coefficients swing, so each individual fit has chased noise. The cure for variance there was to constrain the model; here the constraint is a penalty on β , which *regularizes* the solution.

Longley: reading the condition number

Recall that the condition number of our data is around 10^8 .

A large condition number is evidence of a problem.

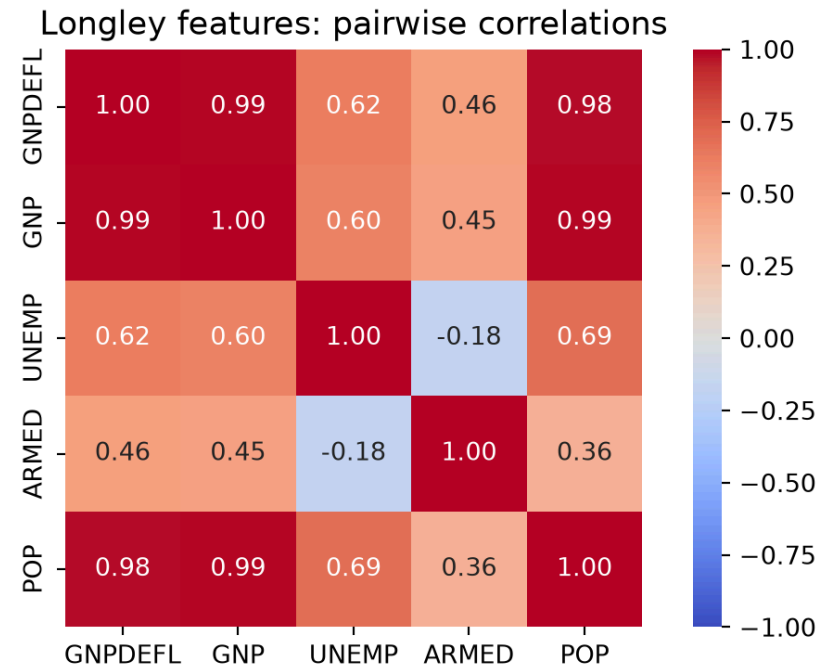
As a general rule of thumb:

- If the condition number is less than 100, there is no serious problem with multicollinearity.
- Condition numbers between 100 and 1000 imply moderate to strong multicollinearity.
- Condition numbers bigger than 1000 indicate severe multicollinearity.

Longley: which columns are nearly dependent?

The correlation matrix of the features shows where the near-dependence comes from.

► Code



We can see **very** strong linear relationships between **GNP Deflator**, **GNP**, and **Population** (correlations above 0.97): three columns carrying essentially one piece of information.

Addressing Multicollinearity

Here are two strategies we can employ to address multicollinearity:

1. Ridge Regression
2. Model Selection via LASSO

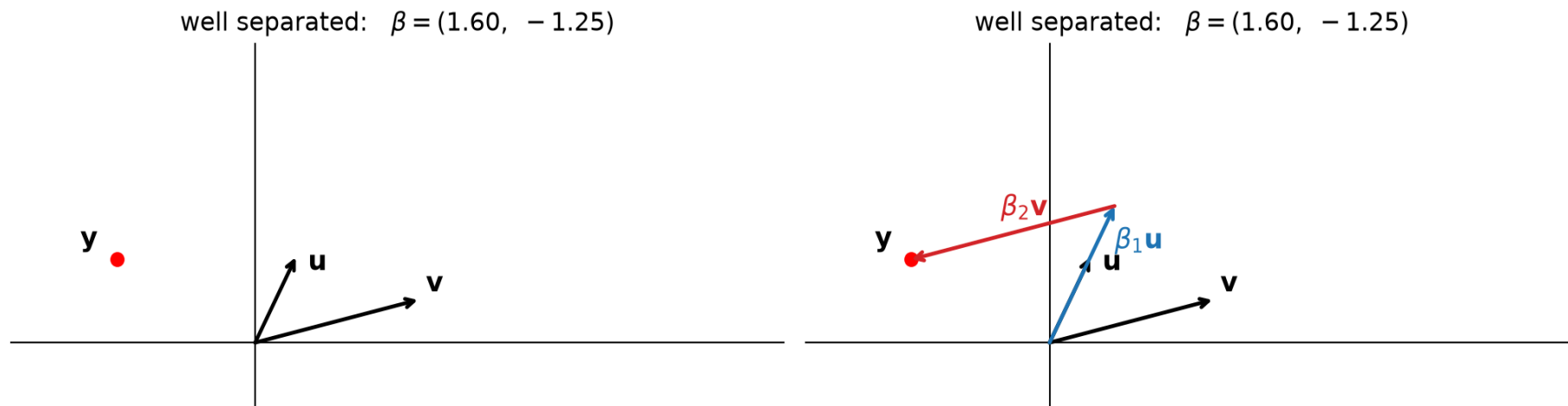
PCA also addresses multicollinearity by transforming the correlated features into uncorrelated features. However in this approach you lose the original features, which is less explainable.

Why the coefficients blow up

The first thing to note is that when columns of X are nearly dependent, the components of $\hat{\beta}$ tend to be **large in magnitude**.

Consider a regression in which we are predicting the point $\mathbf{y}$ as a linear combination $\beta_1 \mathbf{u} + \beta_2 \mathbf{v}$ of two X columns $\mathbf{u}$ and $\mathbf{v}$. In two dimensions this is exact, and the coefficients are the solution of a 2×2 system, computed below with `np.linalg.solve`.

► Code



Why the coefficients blow up

- **Top row:** $\mathbf{u}$ and $\mathbf{v}$ point in clearly different directions, and the same $\mathbf{y}$ is reached with coefficients of modest size, $\beta = (1.60, -1.25)$.
- **Bottom row:** $\mathbf{u}$ and $\mathbf{v}$ are nearly parallel. The part of $\mathbf{y}$ that points *away* from their shared direction can only be built as the difference of two long, nearly cancelling vectors: $\beta = (5.60, -3.65)$, more than three times as large for the same target.
- Push $\mathbf{u}$ and $\mathbf{v}$ closer still and the coefficients grow without bound – and flip wildly if $\mathbf{y}$ moves a little.

This geometric argument illustrates why the regression coefficients will be very large under multicollinearity. As a result, $\|\beta\|_2$ will be very large – which is exactly what ridge regression penalizes.

Ridge Regression

Ridge regression adjusts the least squares regression by shrinking the estimated coefficients towards zero.

The purpose is to fix the magnitude inflation of $\|\beta\|_2$.

To do this, Ridge regression assumes that the model has no intercept term – both the response and the predictors have been centered so that $\beta_0 = 0$.

Ridge regression then consists of adding a penalty term to the regression:

$$\hat{\beta} = \arg \min_{\beta} \|X\beta - \mathbf{y}\|_2^2 + c\|\beta\|_2^2.$$

Ridge Regression

For any given c this has a closed-form solution in which $\hat{\beta} = (X^T X + cI)^{-1} X^T \mathbf{y}$.

The solution to the Ridge regression problem always exists and is unique, even when the data contains multicollinearity: adding c to every eigenvalue of $X^T X$ lifts the smallest one off zero.

Here, $c \geq 0$ is a tradeoff parameter and controls the strength of the penalty term:

- When $c = 0$, we get the least squares estimator: $\hat{\beta} = (X^T X)^{-1} X^T \mathbf{y}$
- When $c \rightarrow \infty$, we get $\hat{\beta} \rightarrow 0$.
- Increasing the value of c forces the norm of $\hat{\beta}$ to decrease, yielding smaller coefficient estimates in magnitude.

For a finite, positive value of c , we are balancing two tasks: fitting a linear model and shrinking the coefficients.

The coefficient c is a **hyperparameter** that controls the model complexity. We typically set c by holding out data, i.e., **cross-validation**.

Ridge in one dimension

With a single centered (mean-subtracted) column $\mathbf{x}$, write the sum of squares as $s = \mathbf{x}^T \mathbf{x}$.

Then $\hat{\beta}_{LS} = \mathbf{x}^T \mathbf{y} / s$ is the least-squares slope/solution.

Expanding the squared error around it, $\|\mathbf{y} - \mathbf{x}\beta\|_2^2 = s(\beta - \hat{\beta}_{LS})^2 + \text{const}$, so ridge minimizes

$$s(\beta - \hat{\beta}_{LS})^2 + c\beta^2 \implies 2s(\beta - \hat{\beta}_{LS}) + 2c\beta = 0 \implies \hat{\beta}_{\text{ridge}} = \frac{s}{s + c} \hat{\beta}_{LS} = \frac{\hat{\beta}_{LS}}{1 + c/s}.$$

- Ridge **multiplies** the least-squares estimate by a factor in $(0, 1)$: it shrinks, but never reaches zero for finite c .
- For a standardized column $s = \mathbf{x}^T \mathbf{x} = n$, so the shrinkage factor is $1/(1 + c/n)$: a penalty of $c = n$ halves every coefficient. (Keep this in mind when reading the c axis of the plots below – c is on the scale of n .)
- In many dimensions the same thing happens along each singular direction of X , with factor $\sigma_j^2/(\sigma_j^2 + c)$: the directions with tiny σ_j – the nearly dependent ones – are shrunk the most. That is the next lecture's story.

Scaling

Note that the penalty term $\|\beta\|_2^2$ would be unfair to the different predictors if they are not on the same scale: a useful feature measured in tiny units needs a huge coefficient and is punished for it, while a feature in huge units is nearly free.

Therefore, if we know that the variables are not measured in the same units, we first standardize the columns of X (zero mean, unit variance) and center y , and then perform ridge regression.

By centering y and X we do not need (or include) an intercept in the model – and the intercept should never be penalized.

Fit the scaler on the training split only. The means and standard deviations are part of the model; computing them on data you will later test on leaks test information into training.

Another name for ridge regression is **Tikhonov regularization**. You may see this terminology used in textbooks on optimization.

Standardizing is needed for this specific method. This is in contrast to the [linear regression](#) lecture where we allowed the coefficients to correct for the scaling differences between different units of measure.

Ames housing: a data set for the sweeps

Longley has only 16 rows – fine for showing a condition number, hopeless for cross-validation (8 training points for 5 features). For the penalty sweeps we switch to the Ames housing data from the [linear regression](#) lecture: 1460 houses, target **SalePrice**, and eight numeric features chosen to include three **genuinely collinear pairs**.

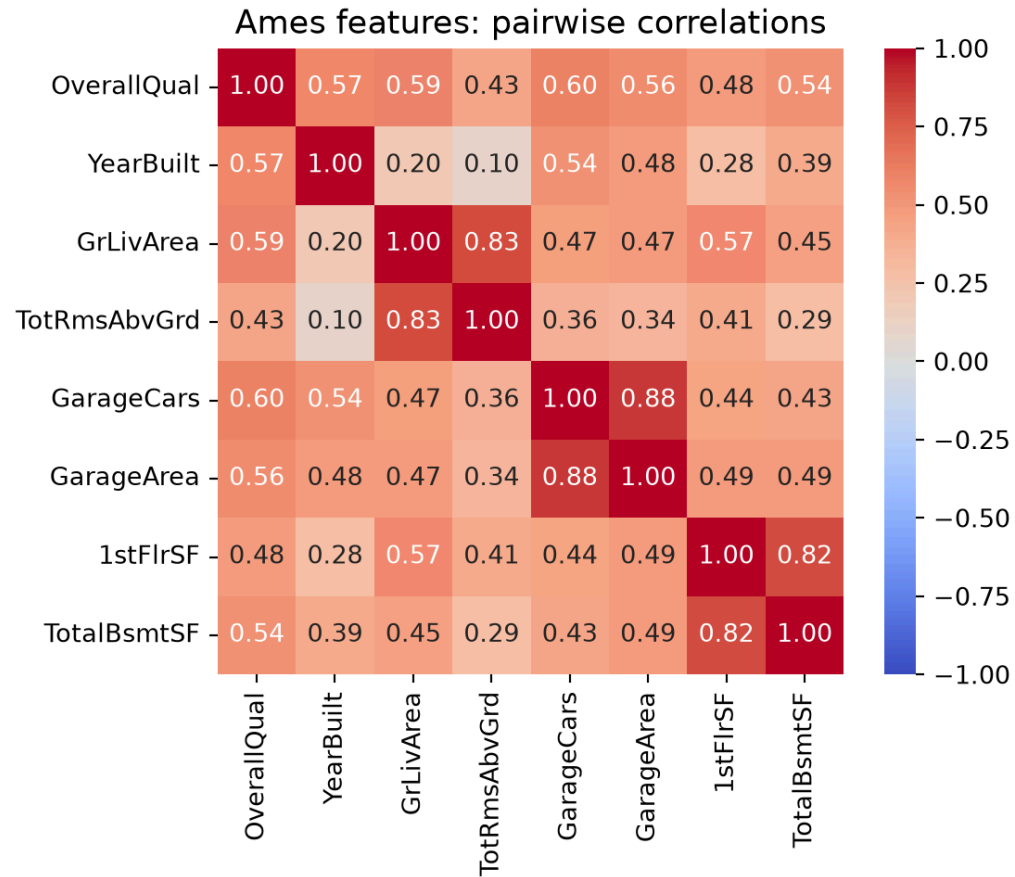
► Code

(1460, 8)

	OverallQual	YearBuilt	GrLivArea	TotRmsAbvGrd	GarageCars	GarageArea	1stFlr
mean	6.0	1971.0	1515.0	7.0	2.0	473.0	1163.
std	1.0	30.0	525.0	2.0	1.0	214.0	387.0
min	1.0	1872.0	334.0	2.0	0.0	0.0	334.0
max	10.0	2010.0	5642.0	14.0	4.0	1418.0	4692.

Ames: the collinear pairs

► Code

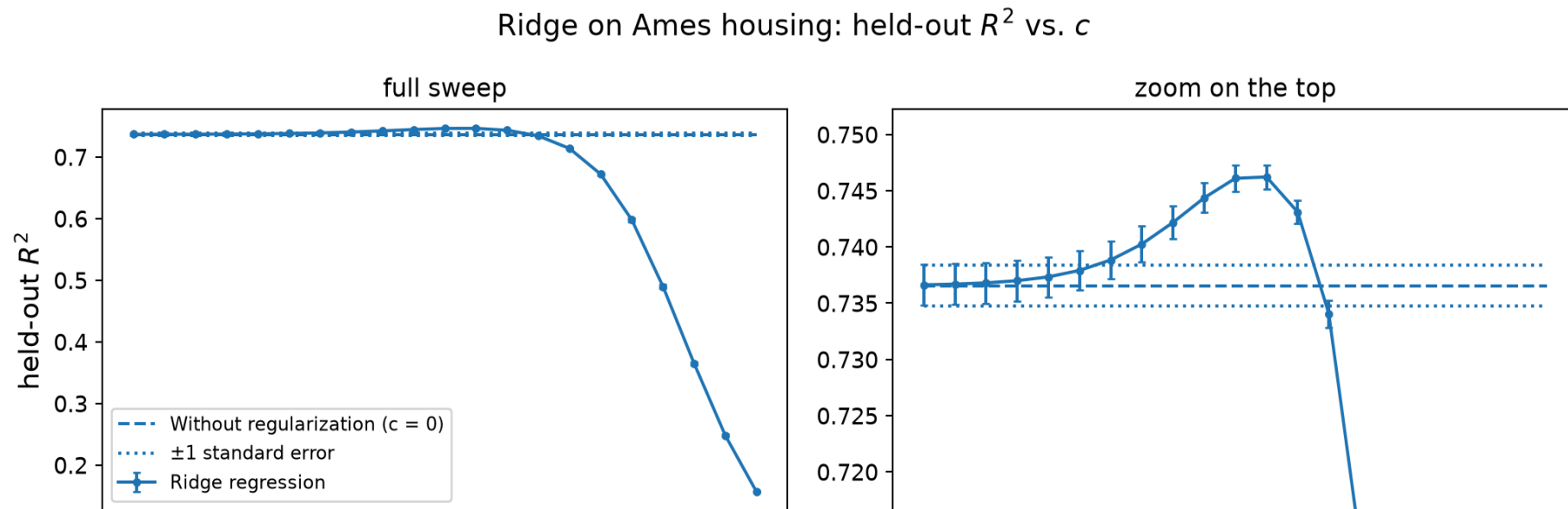


Ridge on Ames: the held-out sweep

For each c we repeat 100 random train/test splits. To make the variance of $\hat{\beta}$ – the thing the penalty controls – visible, we train on a deliberately small 20% split (292 houses) and test on the other 80%. The scaler is fit on the training split each time. We use `sklearn`'s `Ridge`, whose `alpha` is exactly our c .

► Code

► Code



Reading the ridge sweep

- For small c nothing changes: the penalty is negligible next to the squared error.
- Around $c \approx 10$ – 100 (recall $n = 292$, so this is shrinkage of roughly 5–25%) the held-out R^2 rises **above** the unpenalized fit, by several standard errors. Shrinking the coefficients made the predictions *better*: bias bought, variance sold.
- Past $c \approx 300$ the shrinkage is too strong, the model underfits and R^2 collapses toward zero.
- The sweet spot is found exactly this way – on **held-out** data. Training error is always lowest at $c = 0$; the penalty can only make the training fit worse.

Ridge Regression: summary

To sum up the idea behind Ridge regression:

1. There may be many β values that are consistent with the equations.
2. Over-fit β values tend to have large magnitudes.
3. We add the regularization term $c\|\beta\|_2^2$ to the least squares to avoid those solutions.
4. We tune c to an appropriate value via cross-validation.

Model Selection

Of course, one might attack the problem of multicollinearity as follows:

- Multicollinearity occurs when variables (features) are close to linearly dependent.
- These variables do not contribute anything *meaningful* to the quality of the model
- As a result why not simply remove variables from the model that are nearly linearly dependent?

We create a new model when we remove these variables from our regression.

This strategy is called **model selection**.

Model selection by hand

Removing columns is easy to *do*. With statsmodels' formula interface a model is a one-line, R-style formula – `response ~ feature + feature` – and the intercept is included for you:

► Code

	coef	std err	t	P> t	[0.025	0.975]
Intercept	5.695e+04	7449.449	7.644	0.000	4.09e+04	7.3e+04
GNPDEFL	-85.1065	123.736	-0.688	0.504	-352.423	182.210
GNP	0.0439	0.013	3.269	0.006	0.015	0.073

Keeping only GNP and the GNP deflator does not help: the two are correlated at 0.99, and the data cannot tell their effects apart – the deflator's interval runs from -352 to $+182$. We would have to try another subset, and another. Formula syntax (categorical variables, transformations, dropping the intercept) is covered in [Appendix: Regression Formulas with statsmodels](#).

Model Selection

One of the advantages of model selection is **interpretability**: by eliminating variables, we get a clearer picture of the relationship between truly useful features and dependent variables.

However, there is a big challenge inherent in model selection. In general, the possibilities to consider are exponential in the number of features.

That is, if we have p features to consider, then there are $2^p - 1$ possible models that incorporate one or more of those features. This space is usually too big to search directly – and we just saw that picking a subset by eye is not reliable either.

Can we use Ridge regression for this problem?

Ridge regression does not set any coefficients exactly to zero unless $c \rightarrow \infty$, in which case they're all zero.

This means Ridge regression cannot perform variable selection. Even though it performs well in terms of prediction accuracy, it does not offer a clear interpretation.

The LASSO

LASSO differs from Ridge regression **only in terms of the norm** used by the penalty term.

$$\hat{\beta} = \arg \min_{\beta} \|X\beta - \mathbf{y}\|_2^2 + c\|\beta\|_1.$$

However, this small change in the norm makes a **big difference** in practice.

The nature of the ℓ_1 penalty will cause some coefficients to be shrunk to zero exactly.

This means that LASSO can perform model selection by telling us which variables to keep and which to set aside.

As c increases, more coefficients are set to zero, i.e., fewer variables are selected.

In terms of prediction error, LASSO performs comparably to Ridge regression but it has a **big advantage with respect to interpretation**.

Why ℓ_1 gives exact zeros: the soft threshold

Repeat the one-dimensional calculation with the ℓ_1 penalty:

- minimize $s(\beta - \hat{\beta}_{LS})^2 + c|\beta|$.

The derivative of $|\beta|$ is ± 1 away from zero, so for $\beta > 0$ we need

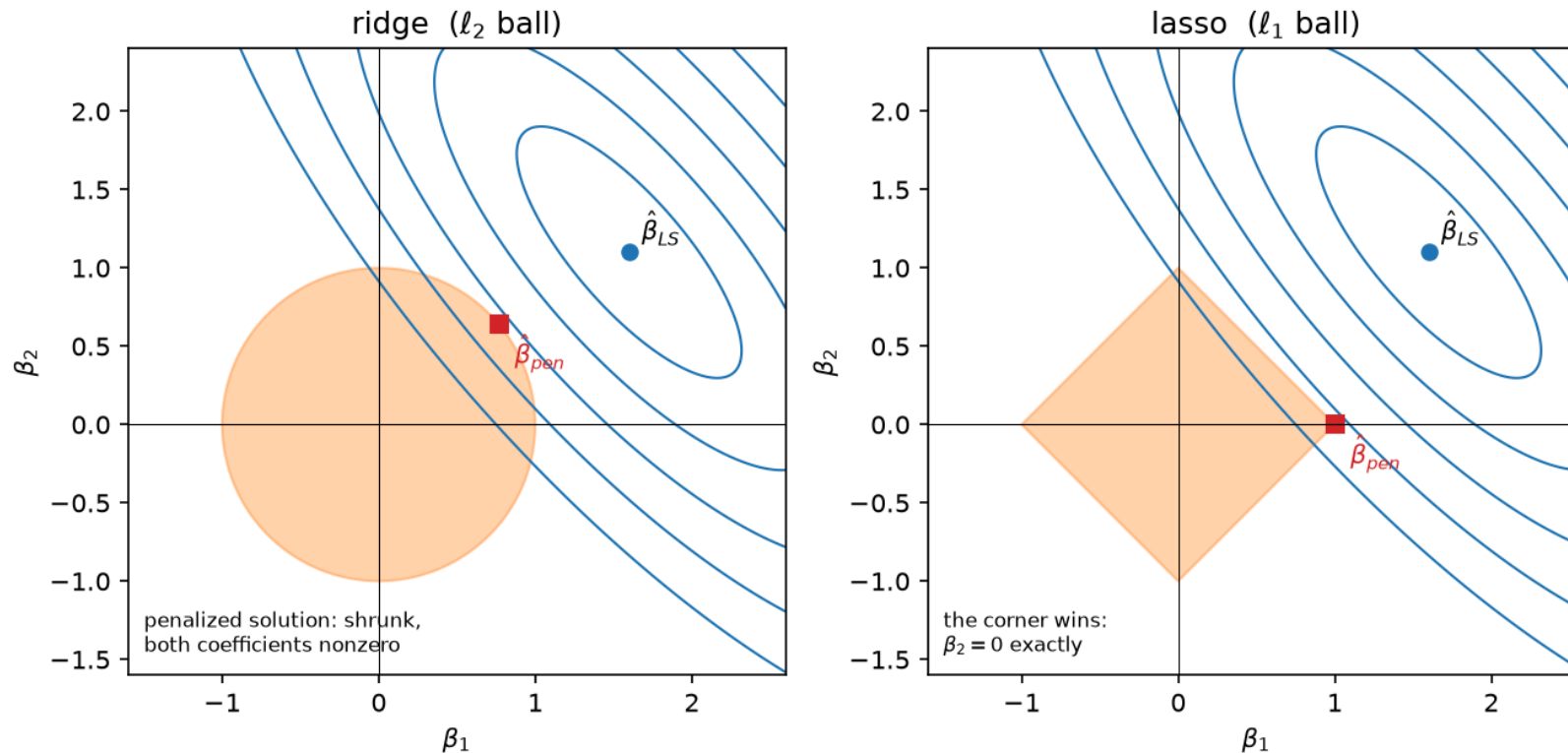
$$2s(\beta - \hat{\beta}_{LS}) + c = 0, \text{ i.e. } \beta = \hat{\beta}_{LS} - c/(2s)$$

which is only valid if the result is still positive. Symmetrically for $\beta < 0$. Otherwise the minimum is at the kink, $\beta = 0$:

$$\hat{\beta}_{\text{lasso}} = \text{sign}(\hat{\beta}_{LS}) \max\left(|\hat{\beta}_{LS}| - \frac{c}{2s}, 0\right).$$

- Ridge **multiplied** by a factor: a small coefficient stays small but nonzero.
- LASSO **subtracts** a fixed amount $c/(2s)$ and stops at zero: every coefficient whose least-squares value is smaller than $c/(2s)$ is set *exactly* to zero. That is why the ℓ_1 penalty selects variables.

The same fact, geometrically



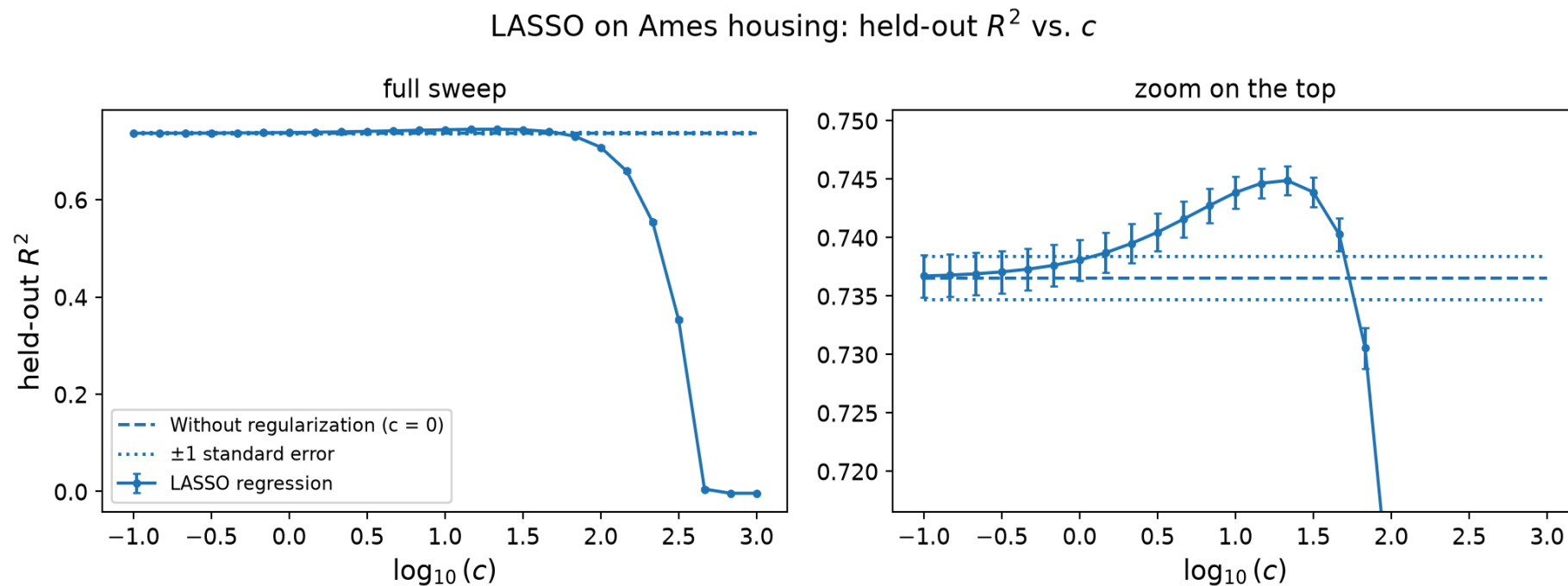
Minimizing the penalized loss is equivalent to minimizing the squared error subject to $R(\beta) \leq \text{some budget}$: the ellipses are contours of the squared error, the shaded region is the penalty ball. The ℓ_2 ball is round, so the contours touch it at a generic point with both coefficients nonzero. The ℓ_1 ball is a diamond with **corners on the axes**, and the contours usually hit a corner first – a corner means a coefficient is exactly zero.

LASSO on Ames: the held-out sweep

Same protocol as for ridge. `sklearn`'s `Lasso` minimizes $\frac{1}{2n} \|X\beta - \mathbf{y}\|_2^2 + \alpha \|\beta\|_1$, so its `alpha` is $c/(2n)$; we convert so that the c axis means the same thing in both plots.

► Code

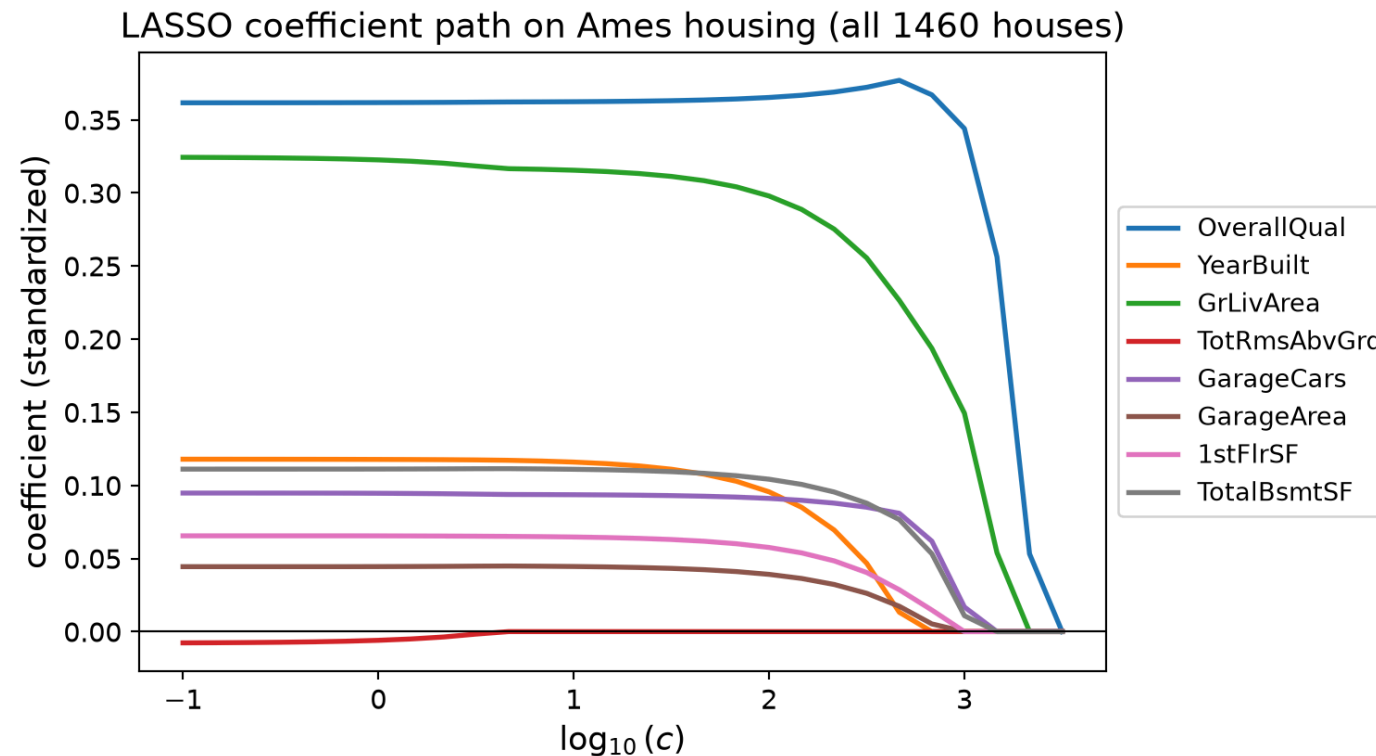
► Code



LASSO on Ames: the coefficient path

The interesting output is not the R^2 curve but **which coefficients survive**. For the path we fit the LASSO once on all 1460 (standardized) houses for each c , so the zeros are exact rather than averaged away. Each line is one feature's coefficient as c grows.

► Code



Interpreting the coefficient path

- With almost no penalty `TotRmsAbvGrd` has a **negative** coefficient: “more rooms, lower price, holding living area fixed”. That sign is an artefact of splitting one effect between two near-duplicate columns – the geometry from earlier, in data.
- It is the **first coefficient LASSO zeroes** (by $c \approx 5$). Of the pair `GrLivArea` / `TotRmsAbvGrd`, LASSO keeps the square footage.
- The remaining coefficients shrink slowly until c is in the hundreds, then `YearBuilt` goes, followed by `GarageArea` and `1stFlrSF` – the weaker member of each remaining pair. `GarageCars` and `TotalBsmtSF` survive them.
- By $c \approx 1000$ two features carry the model, `OverallQual` and `GrLivArea`, with a little `GarageCars` and `TotalBsmtSF` left.

LASSO did model selection for us: from each collinear pair it kept one column, and it dropped them in the order of how little they added. Ridge would have shrunk every one of these coefficients and set none to zero.

Regularizing the logistic model

The penalty is not special to least squares. Add it to the negative log-likelihood of [Lecture 11A](#) and you have **regularized logistic regression**:

$$\min_{\beta} -\ell(\beta) + c\|\beta\|_2^2 \quad \text{or} \quad -\ell(\beta) + c\|\beta\|_1.$$

`sklearn`'s `LogisticRegression` does this **by default**: its parameter `C` is the *inverse* penalty, $C = 1/c$, and the default is `C=1.0`, i.e. an ℓ_2 penalty with $c = 1$. Setting `C=np.inf` (older versions: `penalty=None`) gives the plain maximum-likelihood fit of Lecture 11A.

Regularizing the logistic model: admissions

The admissions data of Lecture 11A, three standardized features (the penalty needs it), fit with no penalty, with the default, and with a strong penalty:

► Code

	no penalty: C=inf	default: C=1 (c=1)	C=0.01 (c=100)
gre	0.265	0.263	0.148
gpa	0.295	0.292	0.148
rank	-0.528	-0.521	-0.233

- With $n = 400$ and three well-conditioned, standardized features, the default penalty barely moves the coefficients – the data dominate $c = 1$.
- Turn the penalty up ($c = 100$) and every coefficient is shrunk toward zero, roughly halved; the model becomes less confident everywhere.
- The practical point: **unless you ask for `penalty=None`, sklearn is regularizing**, and **C** is $1/c$ – *smaller C means more penalty*. In today's activity you will turn this knob the other way: an ℓ_1 penalty on the logistic model, to cut the attributes down to a short list.

Recap: what you can now do

1. **Diagnose multicollinearity:** explain why near-dependent columns make the coefficients large and unstable, and read the condition number $\kappa(X)$ (above 1000 is severe).
2. **Penalize the coefficients:** ridge shrinks by $1/(1 + c/s)$ and never reaches zero; LASSO soft-thresholds to exact zeros, so it selects variables.
3. **Follow the two rules:** standardize before penalizing (scaler fit on the training split), and choose c on held-out data. In `sklearn's LogisticRegression`, `C` is $1/c$ and the penalty is on by default.

For this lecture's class activity you will diagnose multicollinearity (condition number, bootstrap instability of the OLS coefficients), trace ridge and LASSO paths, choose α by cross-validation, see what happens without standardizing, and build a short list with an ℓ_1 -penalized logistic regression.