Machine Learning and Causal Inference.

Regularisation.

Eduardo Fé

05/12/2023

Introduction

Regression analysis studies how an outcome, Y_i correlates with a series of explanatory variables (covariates or regressors), $X_1, X_2, ..., X_p$.

$$\begin{aligned} Y_i &= \beta_0 + \beta_1 X_{1,i} + \beta_2 X_{2,i} + \ldots + \beta_p X_{p,i} + \varepsilon_i \\ &= \beta_0 + \sum_{j=1}^{p} \beta_j \cdot X_{j,i} + \varepsilon_i \end{aligned} \tag{1}$$

The Premise: the above equation describes the TRUE relationship between Y and $X_1, X_2, ..., X_p$ (up to an error, ε, whose effect is negligible, on average).

The $\{\beta_j\}_{j=0}^{p}$ are not known; the $\{Y_i, X_{1,i}, ..., X_{p,i}\}_{i=1}^{n}$ are observed for *many* people/units.

Ordinary Least Squares (**OLS**) makes an efficient use of $\{Y_i, X_{1,i}, ..., X_{p,i}\}_{i=1}^{n}$ to obtain good estimates of $\{\beta_j\}_{j=0}^{p}$.

Linear. . .

$$\text{Acceleration}_i = \beta_0 + \text{Milliseconds} \cdot \beta + \varepsilon_i$$

.

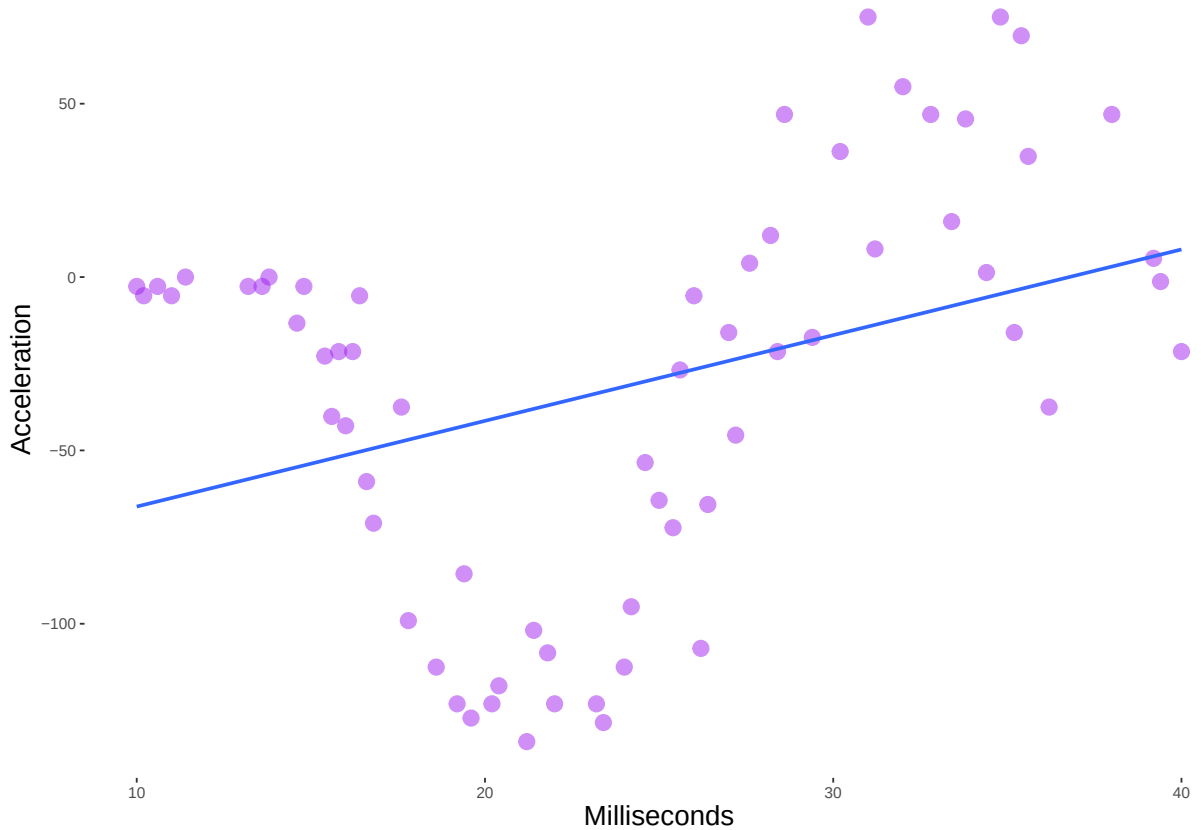


Figure 1: The motorcycle data. The horizontal axis measures time in milliseconds. The vertical axis measures acceleration in the head of a crash test dummy in tests of motorcycle crashes.

Linear. . . E pur si muove!

$$\text{Acceleration} = \beta_0 + \beta_1 \cdot \text{Milliseconds} + \beta_2 \cdot \text{Milliseconds}^2 + \varepsilon$$

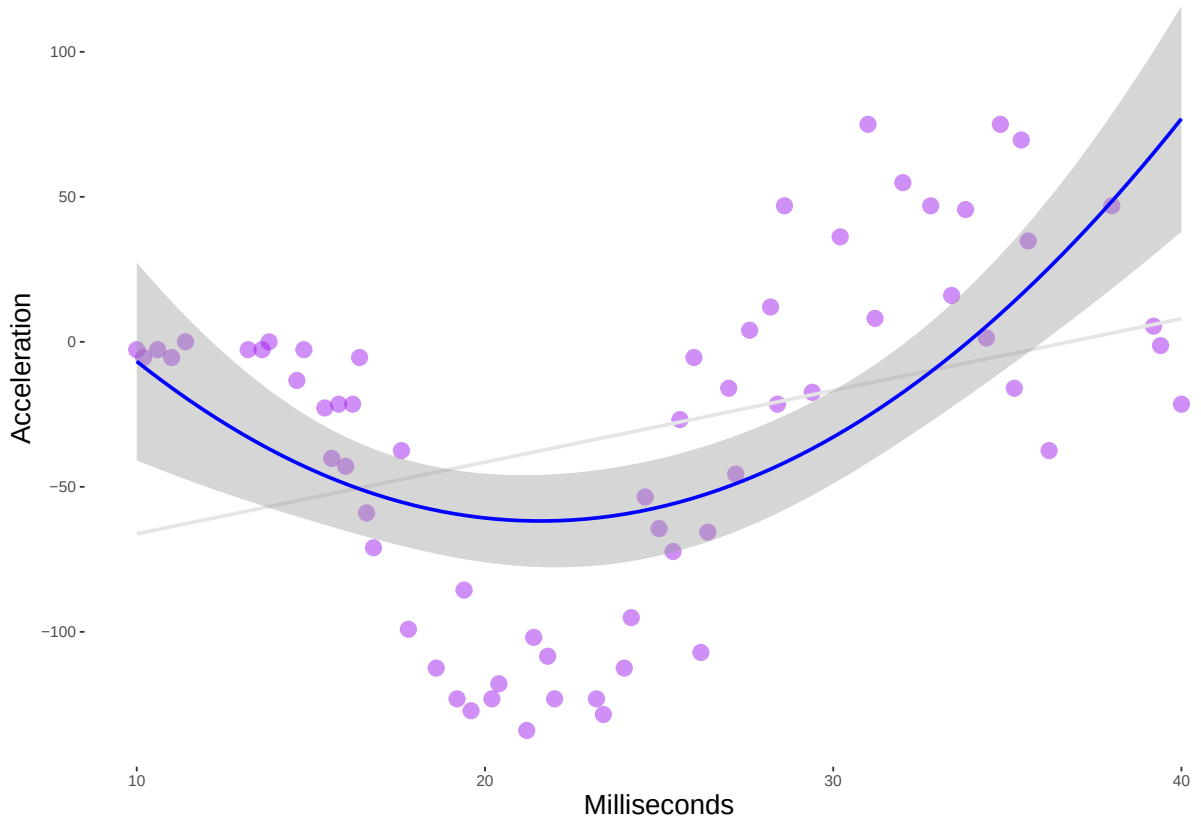


Figure 2: The motorcycle data. The horizontal axis measures time in milliseconds. The vertical axis measures acceleration in the head of a crash test dummy in tests of motorcycle crashes.

Linear. . . E pur si muove!

$$\text{Acceleration} = \beta_0 + \beta_1 \cdot \text{Milliseconds} + \beta_2 \cdot \text{Milliseconds}^2 + \beta_3 \cdot \text{Milliseconds}^3 + \varepsilon$$

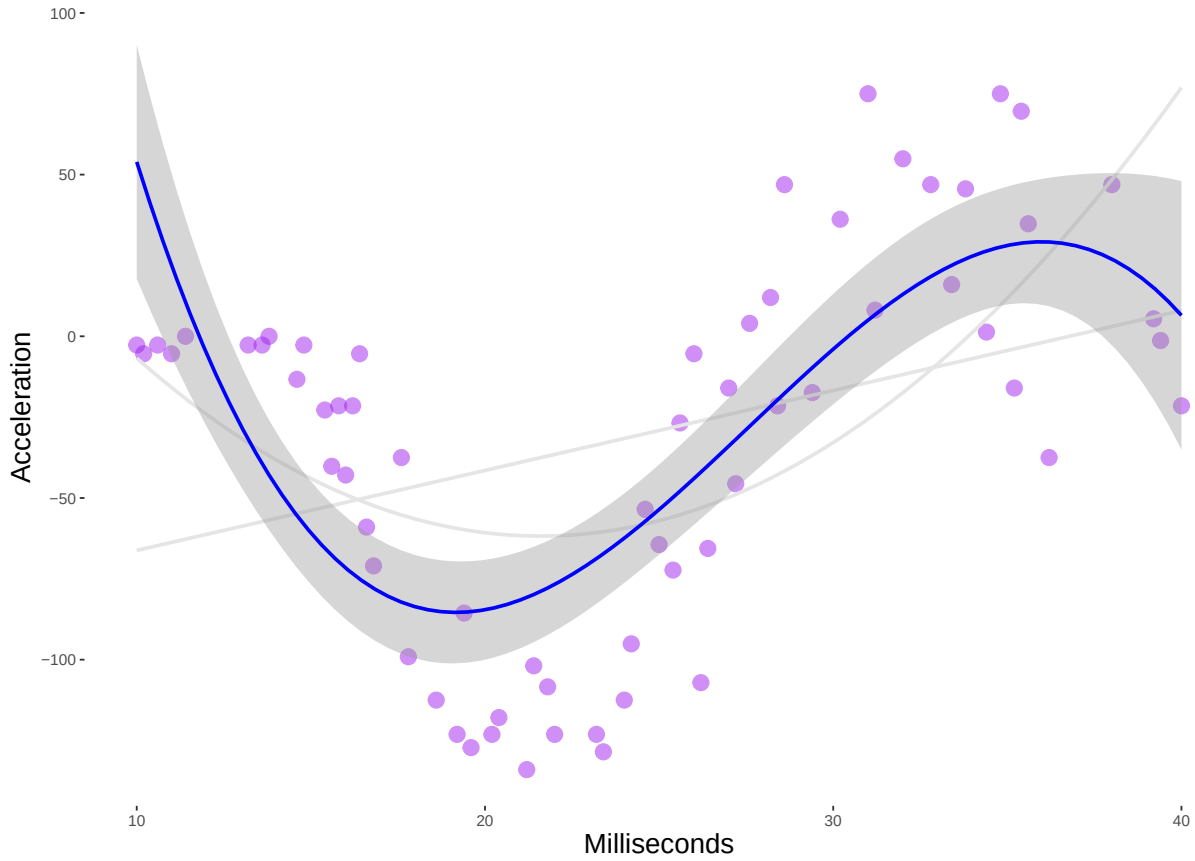


Figure 3: The motorcycle data. The horizontal axis measures time in milliseconds. The vertical axis measures acceleration in the head of a crash test dummy in tests of motorcycle crashes.

Linear. . . E pur si muove!

$$\text{Acceleration} = \beta_0 + \sum_{j=1}^{5} \beta_j \cdot \text{Milliseconds}^j + \varepsilon$$

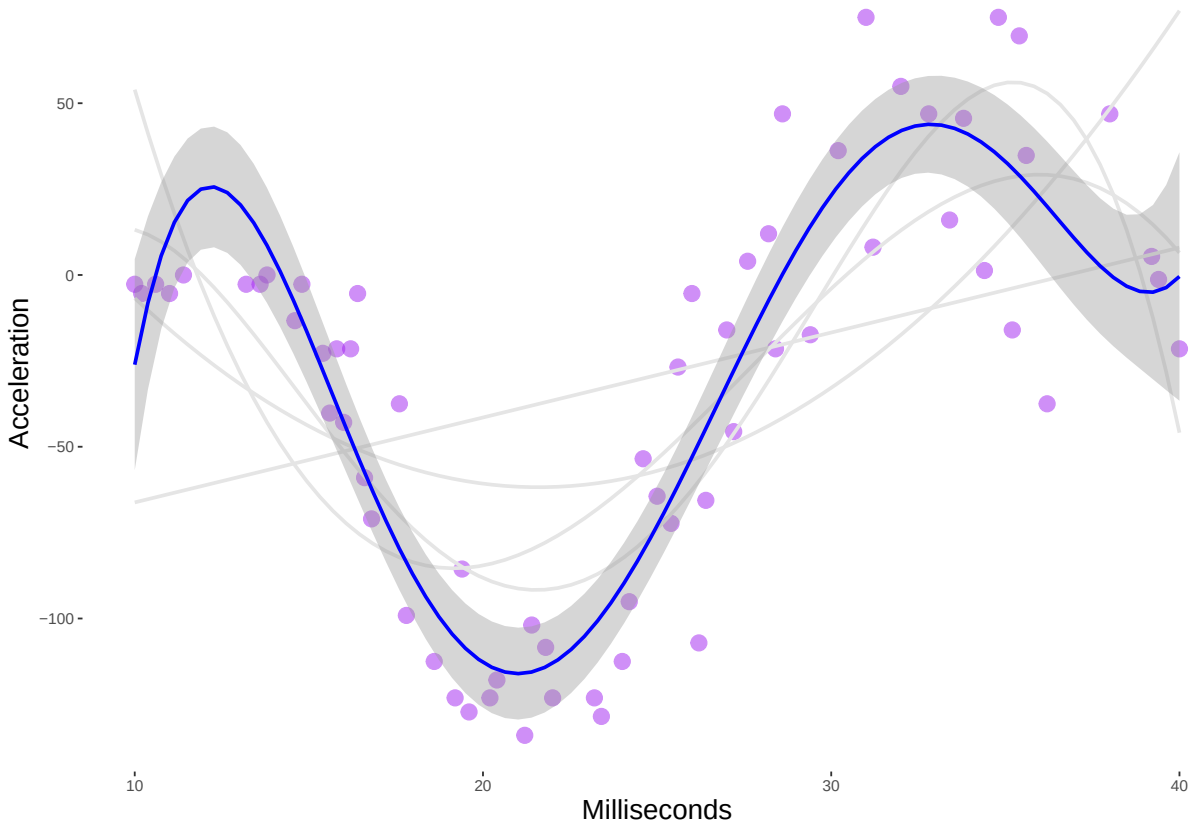


Figure 4: The motorcycle data. The horizontal axis measures time in milliseconds. The vertical axis measures acceleration in the head of a crash test dummy in tests of motorcycle crashes.

Modelling

Regressions can capture “jumps”

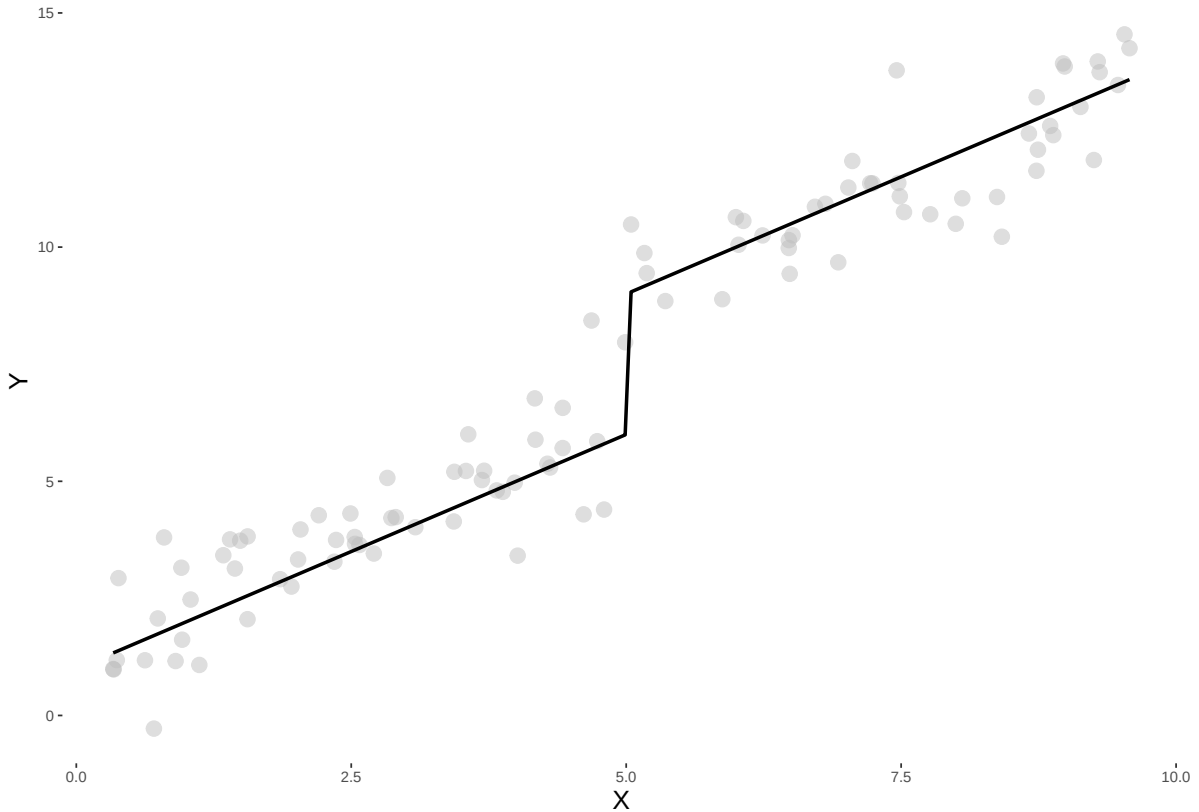


Figure 5: Simulated data of a regression model with a jump at x= 5. At that point, the average value of Y jumps by 3. The black line is the true model

Jumps...

Dummy variables

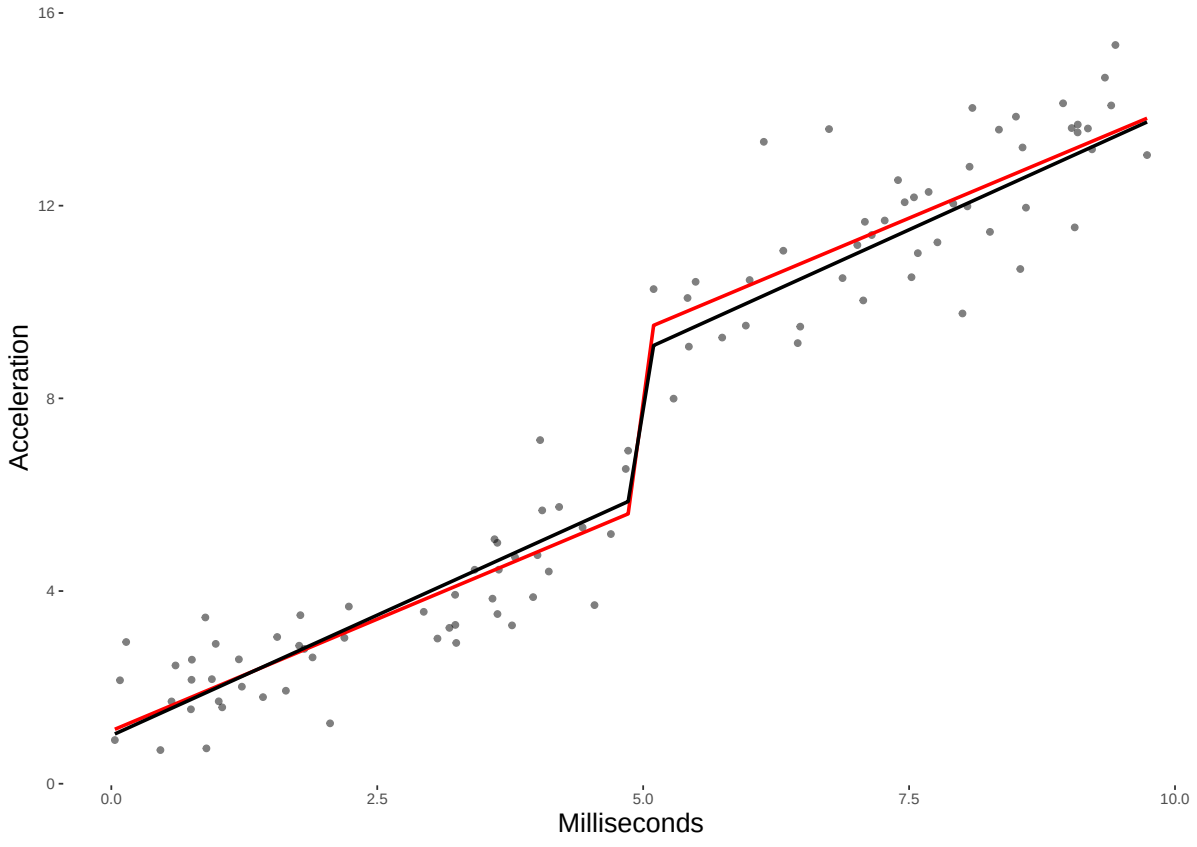
$$X_{2,i} = \begin{cases} 1 & \text{if } X_{1,i} > 5 \\ 0 & \text{Otherwise} \end{cases}$$

and so on.

Then,

$$Y_i = \beta_0 + \beta_i X_{i,1} + \beta_2 \cdot X_{2,i} + \varepsilon_i = \begin{cases} \beta_0 + \beta_1 X_{1,i} + \varepsilon_i & X_{1,i} < 5 \\ \beta_0 + \beta_1 X_{1,i} + \beta_2 + \varepsilon_i & X_{1,i} \geq 5 \end{cases}$$

Jumps. . .



. . . and Kinks

Regressions can also capture **Interactions**:

- If you are clever (CL) you earn GPB100 per month more, on average
- If you are hard working (HW) you earn GPB100 per month more, on average
- If you are clever and hardworking, you earn GBP500 per month more, on average.

$$
\begin{aligned}
Y_i &= \beta_0 + \beta_1 \cdot \mathrm{CL}_i + \beta_2 \cdot \mathrm{HW}_i + \beta_3 \cdot CL_i \times HV_i + \varepsilon_i \\
&= \begin{cases}
(\beta_0 + \beta_1 + \beta_2 + \beta_3) + \varepsilon_i & \text{if clever and hard-working} \\
\beta_0 + \beta_1 + \varepsilon_i & \text{if clever, not hard-working} \\
\beta_0 + \beta_2 + \varepsilon_i & \text{if not clever, hard-working} \\
\beta_0 + \varepsilon_i & \text{...}
\end{cases}
\end{aligned}
$$

Regularisation

With the OLS estimators $\hat{\beta}_j$ $(j = 1, \ldots, p)$, you can predict Y for an arbitrary $X_j = x_j$ $(j = 1, \ldots, p)$,

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j \cdot x_j \tag{2}$$

Why bother, then, with anything else?

Machine Learning seeks to produce good predictions/classifications:

- The model that fits the "training data" best is not guaranteed to be the model that produces the best predictions (out of sample).
- Specifically, in large p situations (close to n) OLS can "Overfit" the data.
- We can improve predictions by allowing some bias in our estimates of β_j.

Modelling

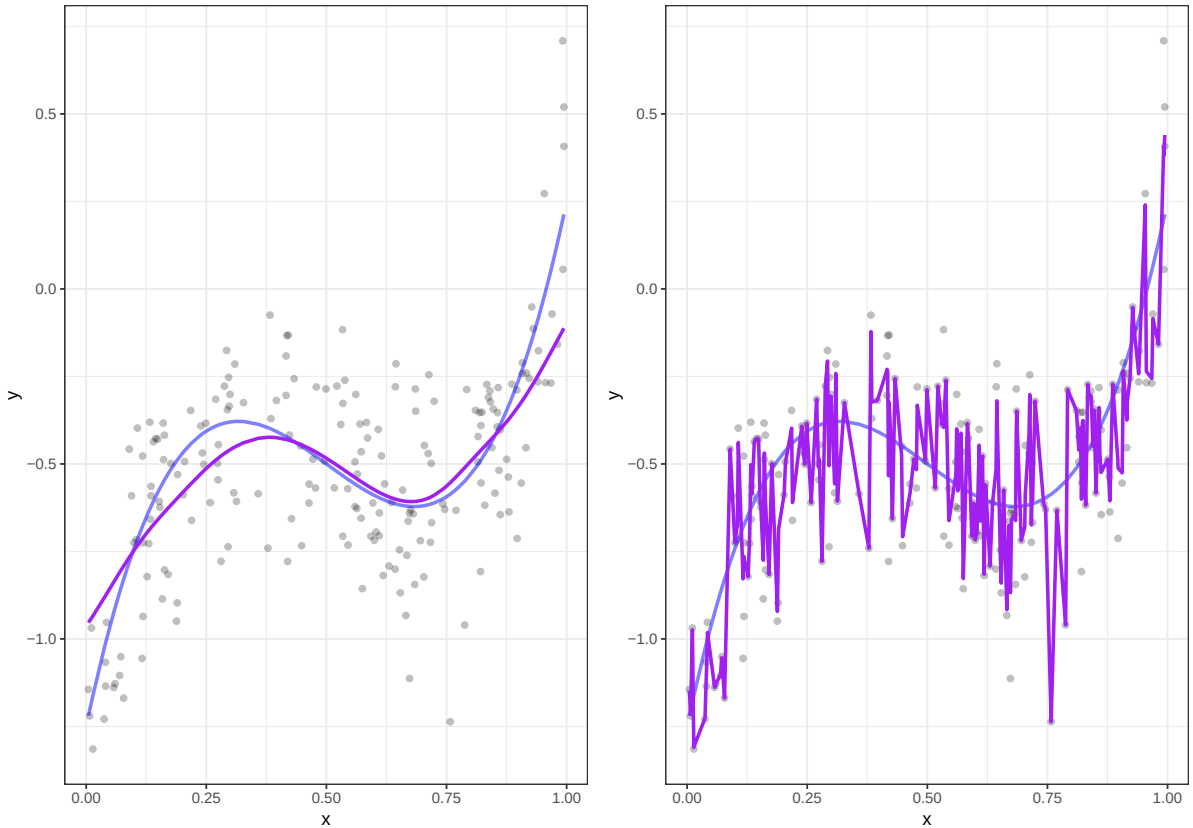


Figure 6: Overfitting in the motorcycle data. The model used a polynomial of order 24 in 'milliseconds'. The horizontal axis measures time in milliseconds. The vertical axis measures acceleration in the head of a crash test dummy in tests of motorcycle crashes.

Train vs Test

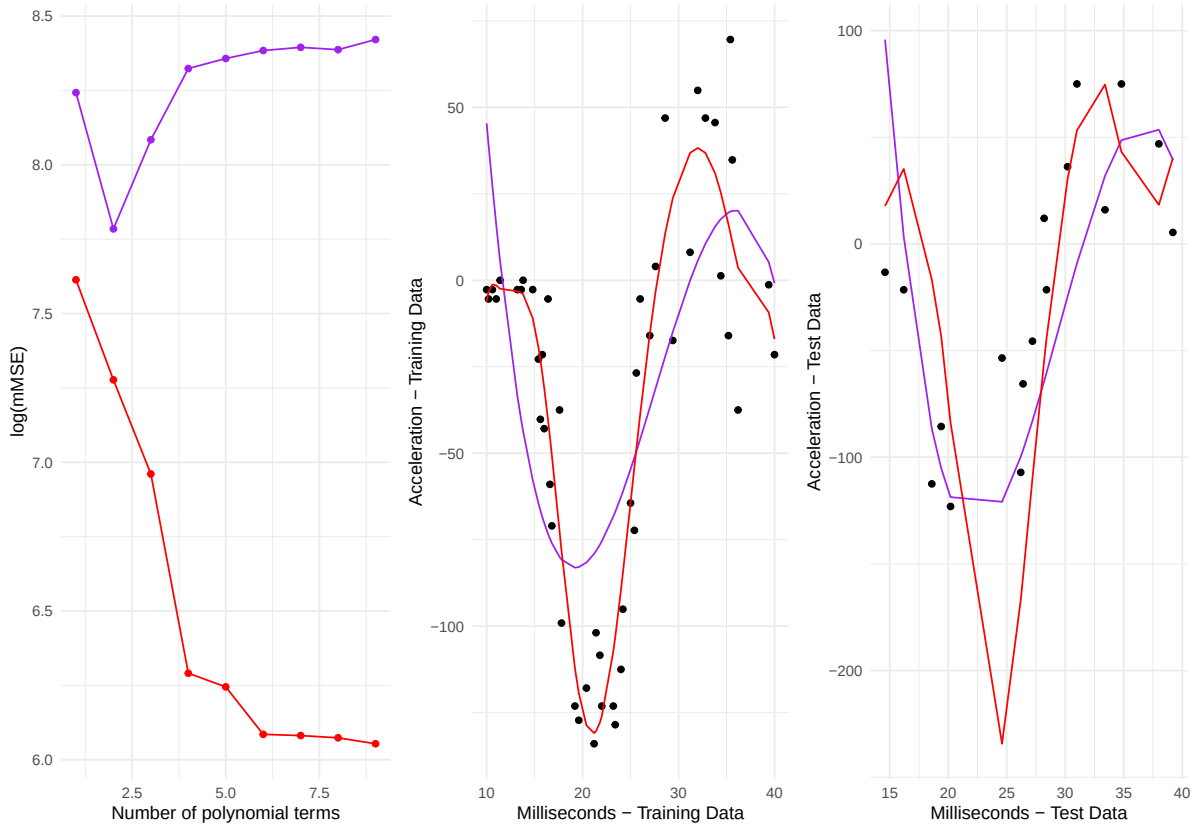


Figure 7: We split the motorcycle data into a 'training' and 'test' datasets (n=48 v 17). We fitted linear regressions with polynomials ranging from order 2 to 10 in each subsample. The left figure displays the logarithm of the Mean Squared Error in both the training (red line) and test (purple line) datasets for each regression model. Meanwhile, the middle and right figures depict the predictions obtained from the best models recommended by the training (red) and test (purple) datasets. In the middle figure, predictions generated by both models using the training data are shown, while the right figure displays predictions generated using the test data.

Bias v Variance

We face a trade-off:

- Complexity can reduce bias, at the cost of increasing the variance in the predictions
- Simplicity can reduce the variance in the predictions, at the cost of increasing the bias

There will typically be an optimal point balancing both components.

In other words: **We can improve the quality of predictions by allowing a certian amount of bias the estimates of β_j.**

Shrinkage methods.

Recall that OLS produced estimates of β_j so as to minimise

$$\text{argmin}_{\beta_j} RSS = \text{argmin}_{\beta_j} \sum_{i=1}^{n} (Y_i - \beta_0 - \sum_{j=1}^{p} \beta_j \cdot X_{j,i})^2$$

If $p > n$ there are ∞ solutions to this problem. If p is not much smaller than n we face over-fitting.

Shrinkage estimators also minimise RSS but they introduce a penalty as model complexity increases.

$$\text{argmin}_{\beta_j} \sum_{i=1}^{n} (Y_i - \beta_0 - \sum_{j=1}^{p} \beta_j \cdot X_{j,i})^2 + \lambda \cdot \text{penalty}(\beta_1, ..., \beta_p)$$

Here $\lambda \geq 0$ is chosen by the researcher (potentially via a data-driven method). The larger λ, the larger complex models are penalised.

Bridge Regulariser

The Bridge Regulariser sets the penalty function equal to a *l-norm*,

$$\text{argmin}_{\beta_j} \sum_{i=1}^{n}(Y_i - \beta_0 - \sum_{j=1}^{p} \beta_j \cdot X_{j,i})^2 + \lambda \cdot \sum_{j=1}^{p} |\beta_j|^l$$

for an arbitrary $l > 0$.

- If $l = 1$, the estimator is the LASSO (Least Absolute Shrinkage and Selection Operator)
- If $l = 2$, the estimator is Ridge Regression

The LASSO has the advantage of setting some coefficients to 0, thus effectively engaging in variable selection automatically.

LASSO vs Ridge Regression

We simulated data from

$$Y_i = \sum_{j=1}^{p} \beta_j \cdot X_{j,i} + U_i$$

where $U \sim N(0, 1)$,

X_j are multivariate normal with variance 1 and correlation 0.4. The true value of the coefficients were

$$(2.000, 1.789, 1.578, 1.367, 1.156, 0.944, 0.733, 0.522, 0.311, 0.100)$$

The sample size was 1000

We consider 100 different values of λ between 0.01 and 100

LASSO vs Ridge Regression

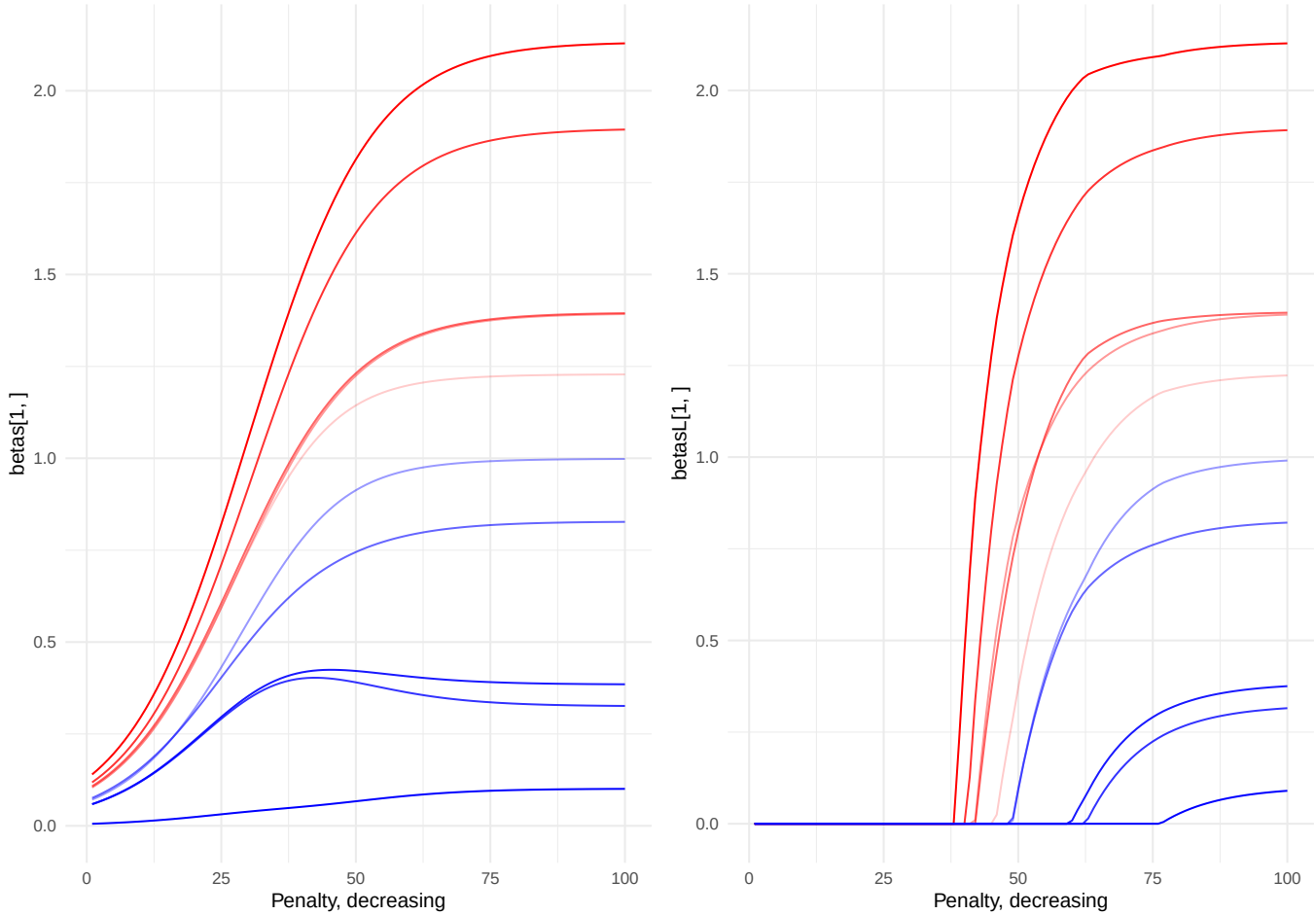


Figure 8: Value of the estimated coefficients, Ridge Regression (left) and Lasso (right). The horizontal axis represents the value of the tuning parameter (which decreases from left to right).

LASSO vs Ridge Regression vs OLS

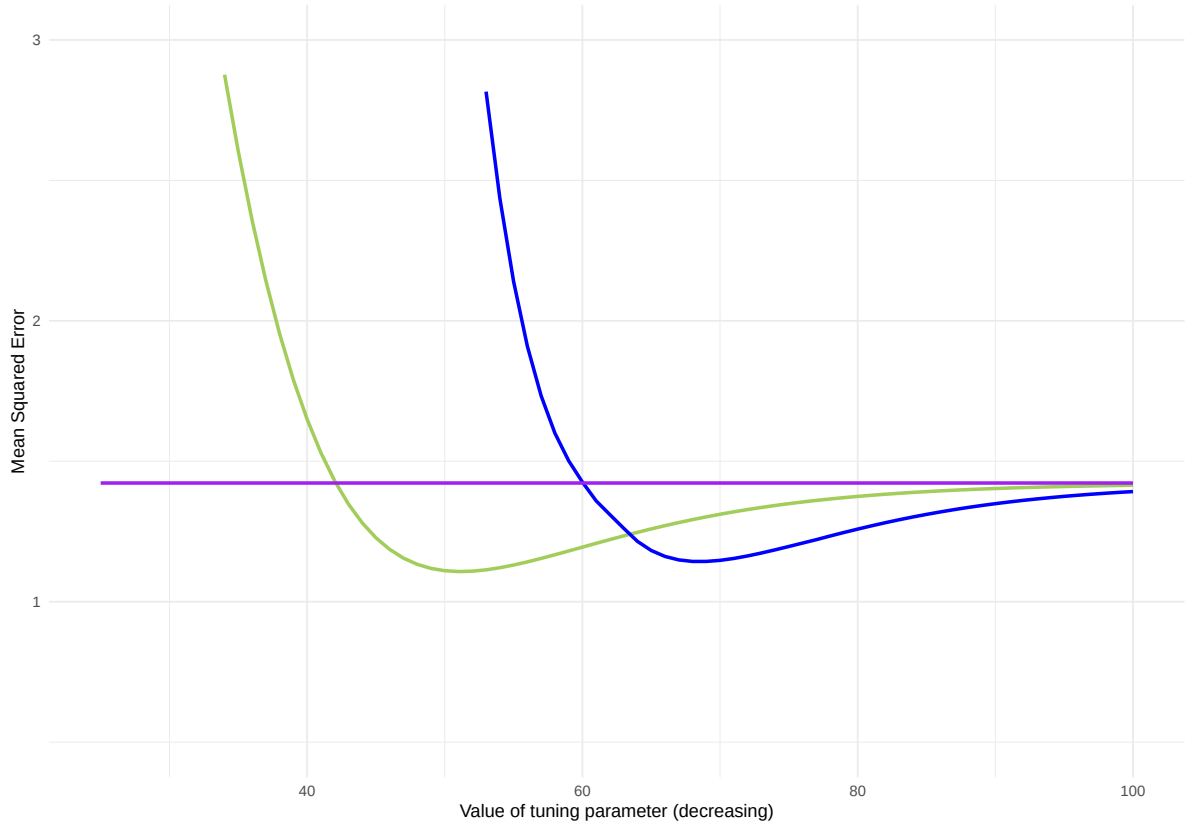


Figure 9: Mean squared error of lasso (blue) vs ridge regression (green), vs OLS (purple), test data. As the penalty (lambda) increases, the test mean squared error (MSE) of the lasso eventually equals the sample variance of Y. This is because the lasso shrinks the coefficients of the model towards zero, which can help to reduce overfitting, but end up removing all explanatory variables. Ridge regression also shrinks the coefficients of the model, but it does so to a lesser extent than the lasso. As a result, the MSE of ridge regression does not increase as much as the MSE of the lasso as the penalty increases

How to select the tuning parameter

Leave-one-out Cross-validation.

For $i = 1, ..., n$, and for a fixed λ

1. Exclude observation i from the training data set.
2. Estimate the model via shrinkage, using all remaining $n - 1$ observations.
3. Using X_i, use the estimated shrinkage model obtain a prediction for Y_i, namely $\hat{Y}_i$
4. Compute the $MSE_i = (Y_i - \hat{Y}_i)^2$

The Leave-one-out cross validation estimate for the **test** MSE is the average

$$CV_n(\lambda) = n^{-1} \sum_{i=1}^{n} MSE_i$$

Select the λ resulting in the smallest $CV_n(\lambda)$

How to select the tuning parameter

K-fold Cross-validation Leave-one-out cross validation can be computational costly if n is large.

Instead, **randomly** split the data into K groups (folds) of the same size. Let $\mathcal{F}_k$ be the set of observations in fold $k = 1, ..., K$. For $k = 1, ..., K$, and for a fixed λ

1. Estimate the model via shrinkage, using all but fold k.
2. Obtain predicted values of Y for all $i \in \mathcal{F}_k$
3. Compute the test $MSE_k = \sum_{i \in \mathcal{F}_k} (Y_i - \hat{Y}_i)^2$

The k-fold cross validation estimate for the **test** MSE is the average

$$CV_k(\lambda) = k^{-1} \sum_{k=1}^{K} MSE_k$$

Select the λ resulting in the smallest $CV_k(\lambda)$

How to select the tuning parameter

There is a variance-bias trade-off when selecting the number of folds. More folds reduce the bias, but increase the variability in the test MSE. In practice, 5 to 10 folds do well and for large samples, the difference in test MSE is small.