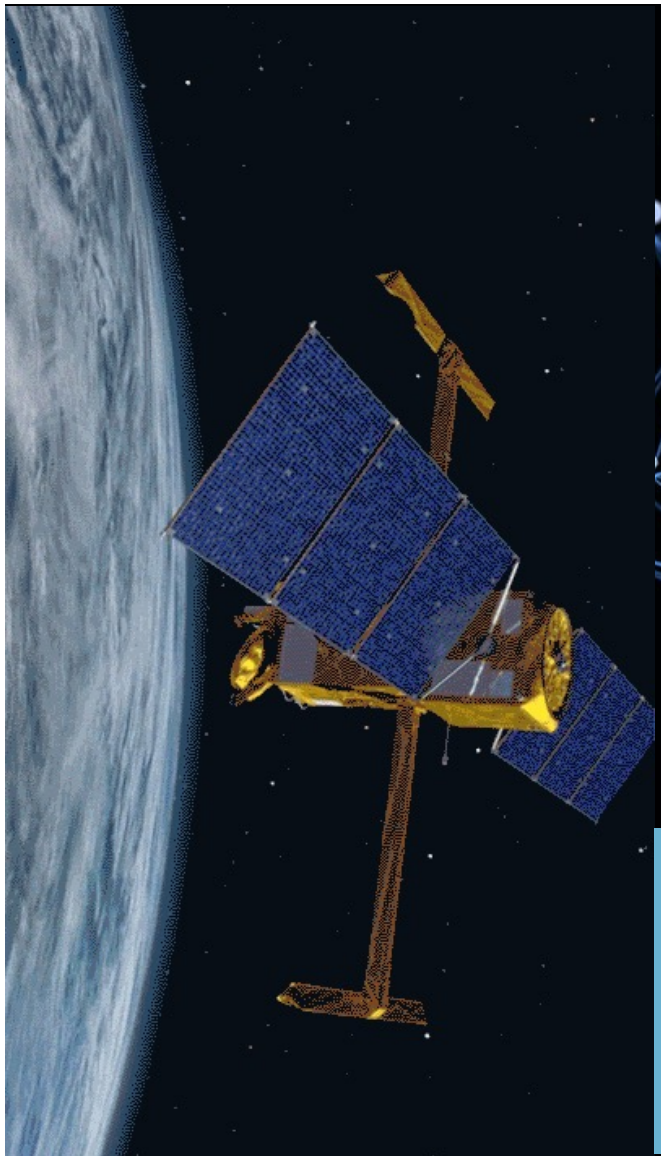


Lecture 6
Cross-Validation & Lasso and Ridge Regularization

Overview

- How to handle underfitting/overfitting?
  - Meaning of **bias** and **variance**
  - **Cross-validation**
  - **Lasso** & **Ridge Regularization**
  - Feature selection

Underfitting/Overfitting

= Bias & Variance

Bias, Variance, and Irreducible Noise

- When fitting perfect model to noise-free data, final error=cost=loss (e.g. MSE) = 0
- In reality, for *m* noisy data points and unknown model:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left(\text{Bias}^2(x^{(i)}) + \text{Variance}(x^{(i)}) + \sigma^2\right)$$

- Assume there is some true relation $f(x)$ between our features x and our target $y(x)$, but we have noisy measurements $y(x) = f(x) + \epsilon(x)$.

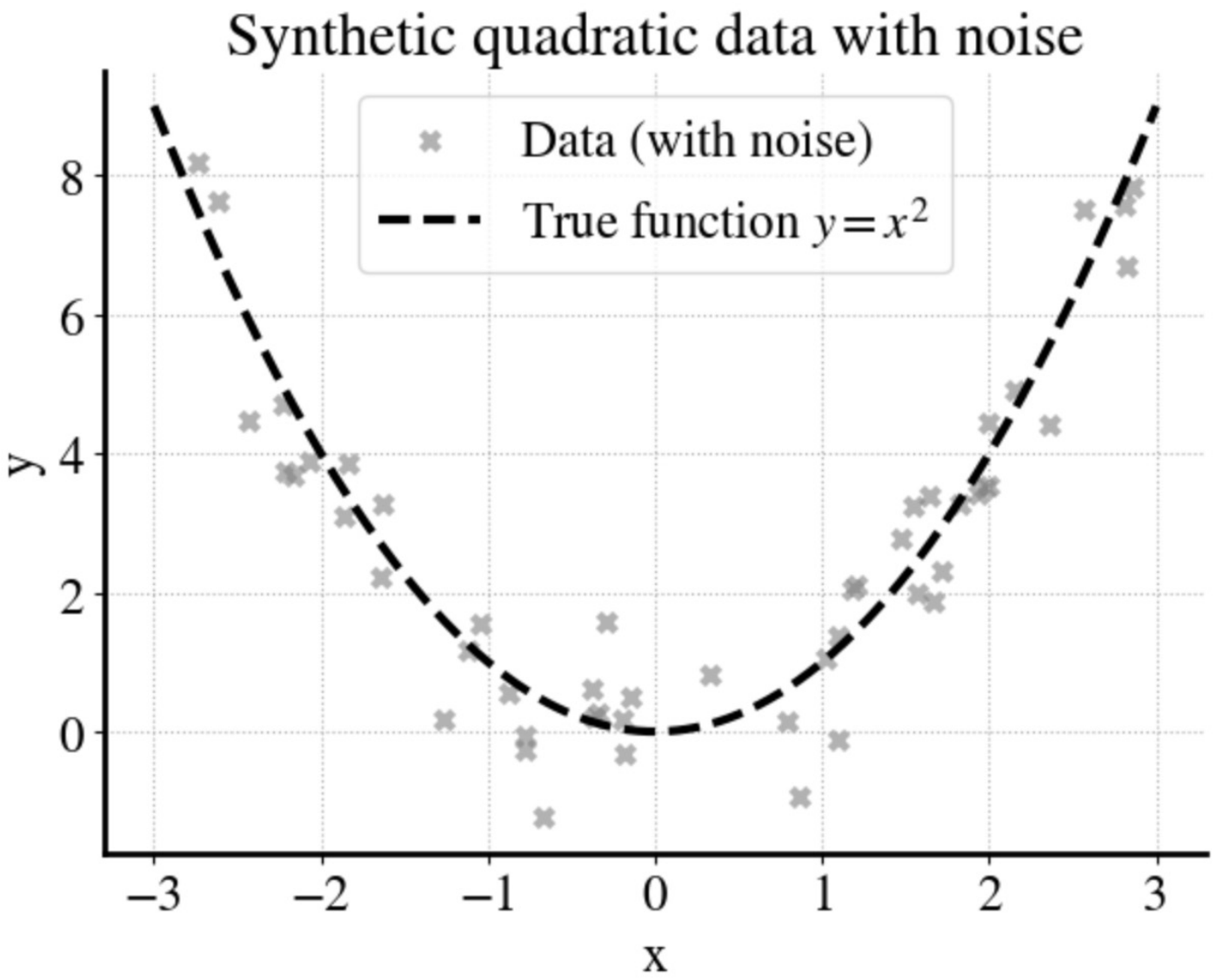
- σ^2: (variance of) **irreducible noise** ϵ in the data itself
- **Bias**^{2}: [underfitting] is difference between *average* model predictions and $f(x)$
- **Variance**: [overfitting] *spread* of predictions across training sets around average

Bias – Precise Definition

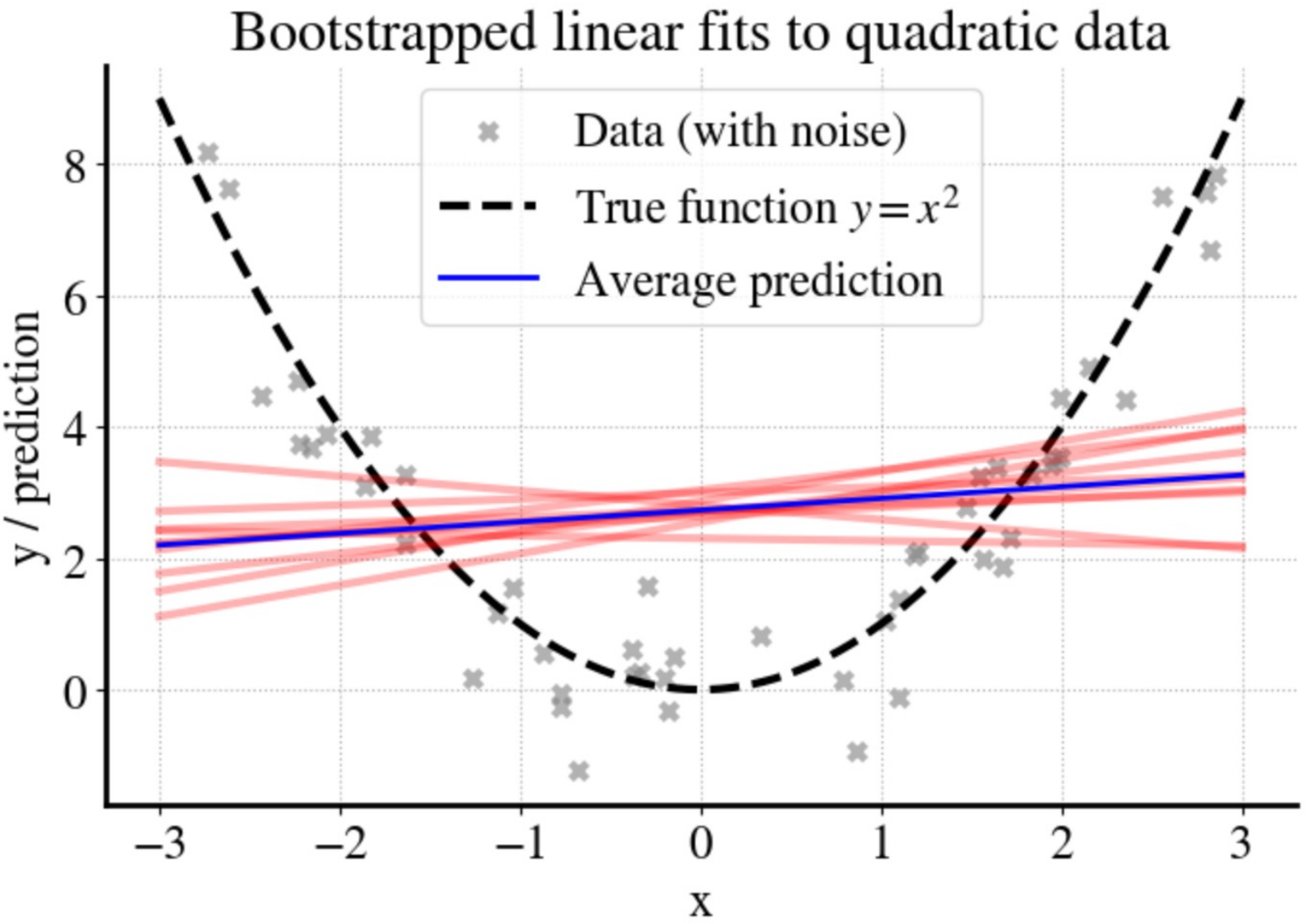
- Consider (a distribution of), say, $m = 1000$ measurements $y(x)$
- To mimic m *random* measurements out of an infinite/continuous distribution, we take B (e.g., 10) random selections of m points *with replacement* (bootstrapping)
- Some samples picked multiple times & some missing in each selection
- For each $b = 1, \dots, B$ we perform our numerical/logistical regression
- Afterwards, for a single sample point $x^{(i)}$, we have B different predictions $\hat{y}_b^{(i)}$
- Average prediction: $\bar{\hat{y}}^{(i)} = \frac{1}{B} \sum_{b=1}^{B} \hat{y}_b^{(i)}$
- Average **bias** (squared) across all m points: $\text{Bias}^2 = \frac{1}{m} \sum_{i=1}^{m} \left(\bar{\hat{y}}^{(i)} - f(x^{(i)}) \right)^2$
- (Squared) difference between average prediction and true value without noise.

Illustration

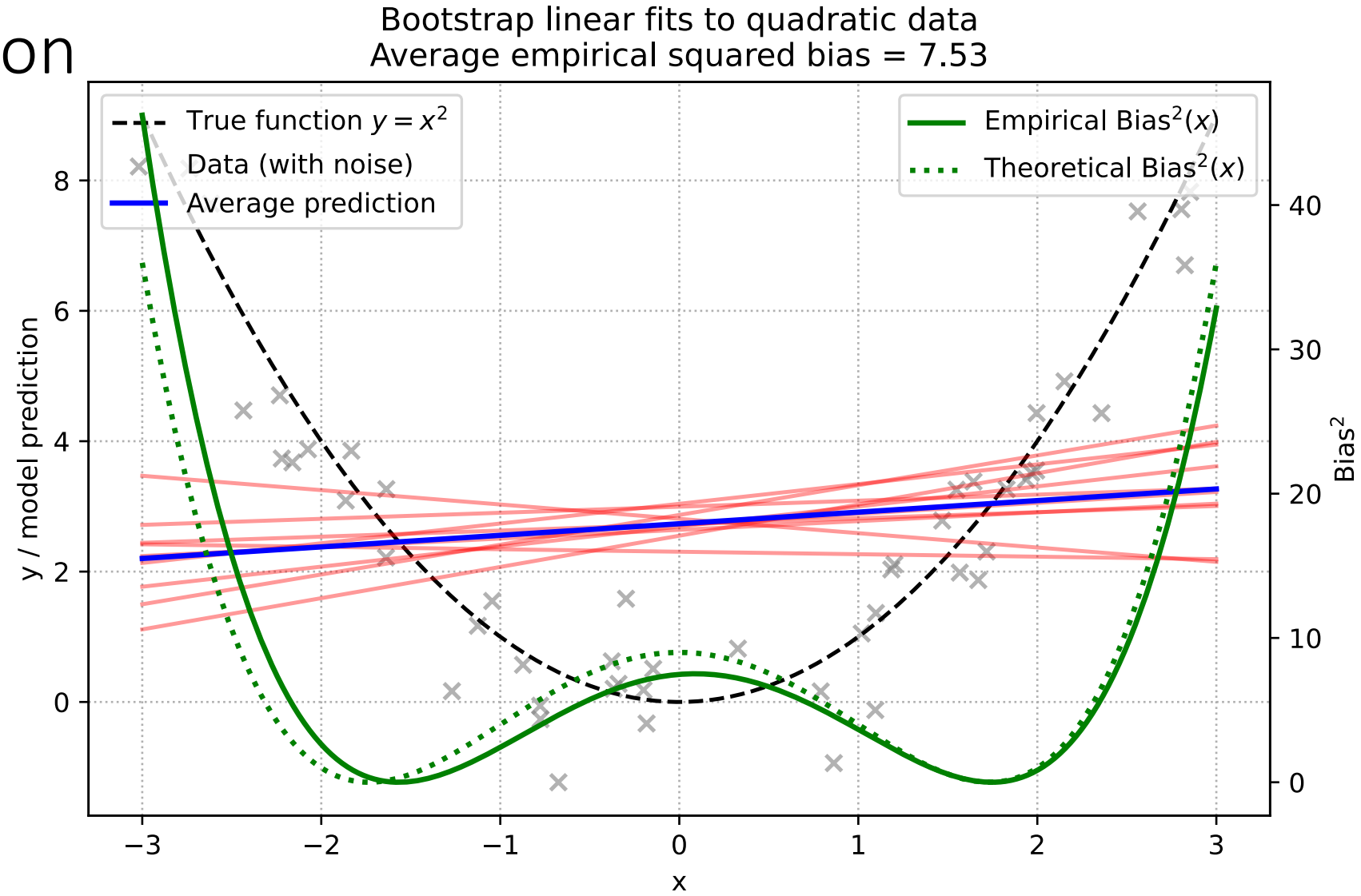
$m = 50$



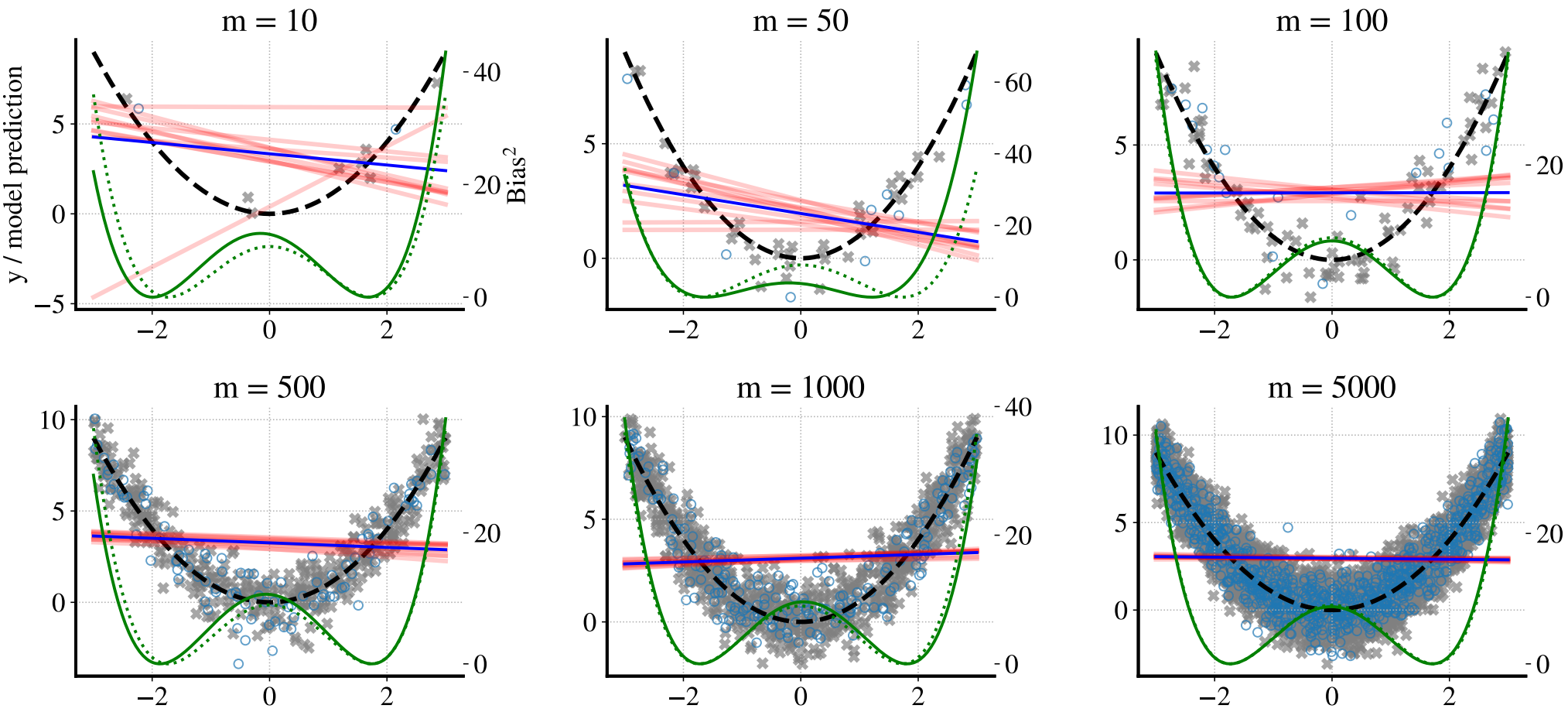
Illustration



Illustration

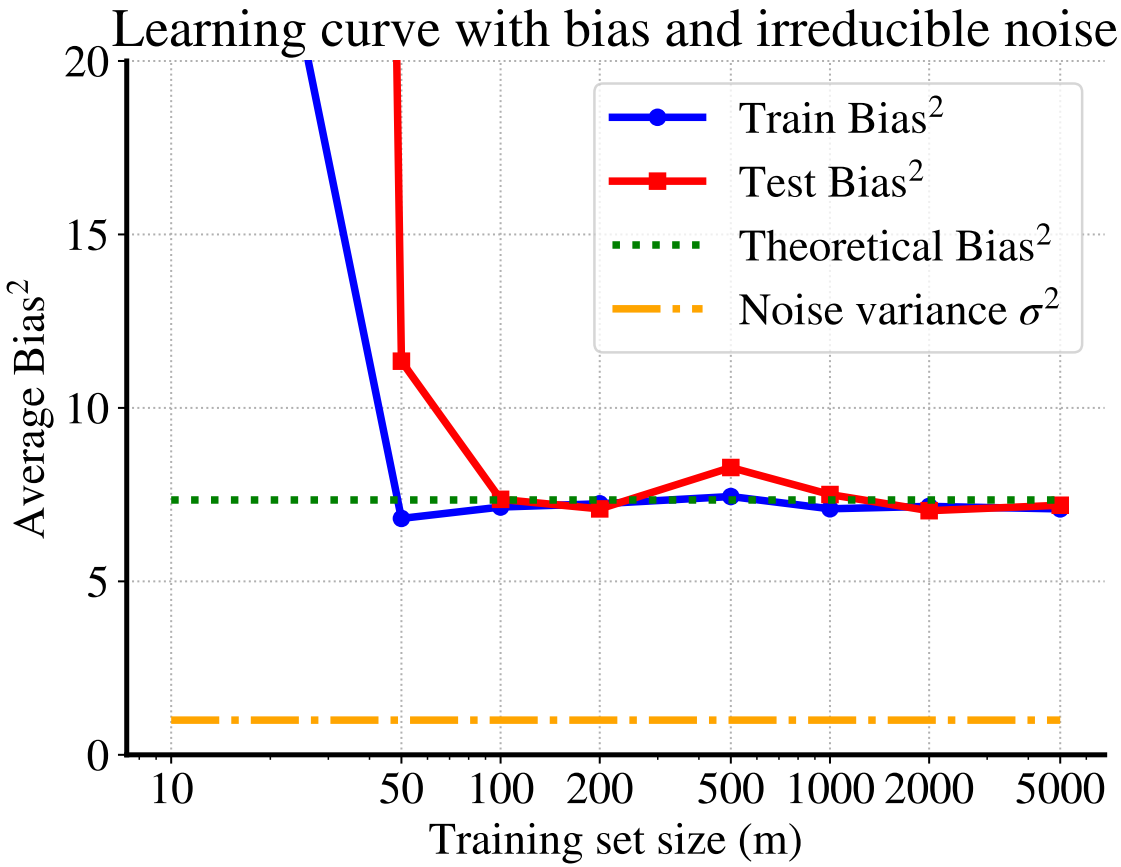


Bias^2 behavior with increasing training set size



MSE on training and test data converge when increasing *m*, but plateau at 'large' value: **underfitting!**

Learning Curve



Bias Summary

- Bias is due to model not being sophisticated enough
- Increasing amount of (training/validation) data does not help

- Solutions:
  - More complex model, such as higher-order polynomial, neural network, decision trees, etc.
  - Or! Additional features (e.g., not just pressure but also temperature, etc.).

Variance – Precise Definition

- Consider same definitions for *m* samples and *B* training data samples
- Variance is 'spread' of individual model predictions around the averaged model prediction across *B* sets of training data, with 'spread' defined just like a MSE:
- For one measurement point:

  (doesn't care about true value)

$$\mathrm{Var}(x^{(i)}) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{y}_b^{(i)} - \bar{\hat{y}}^{(i)}\right)^2$$

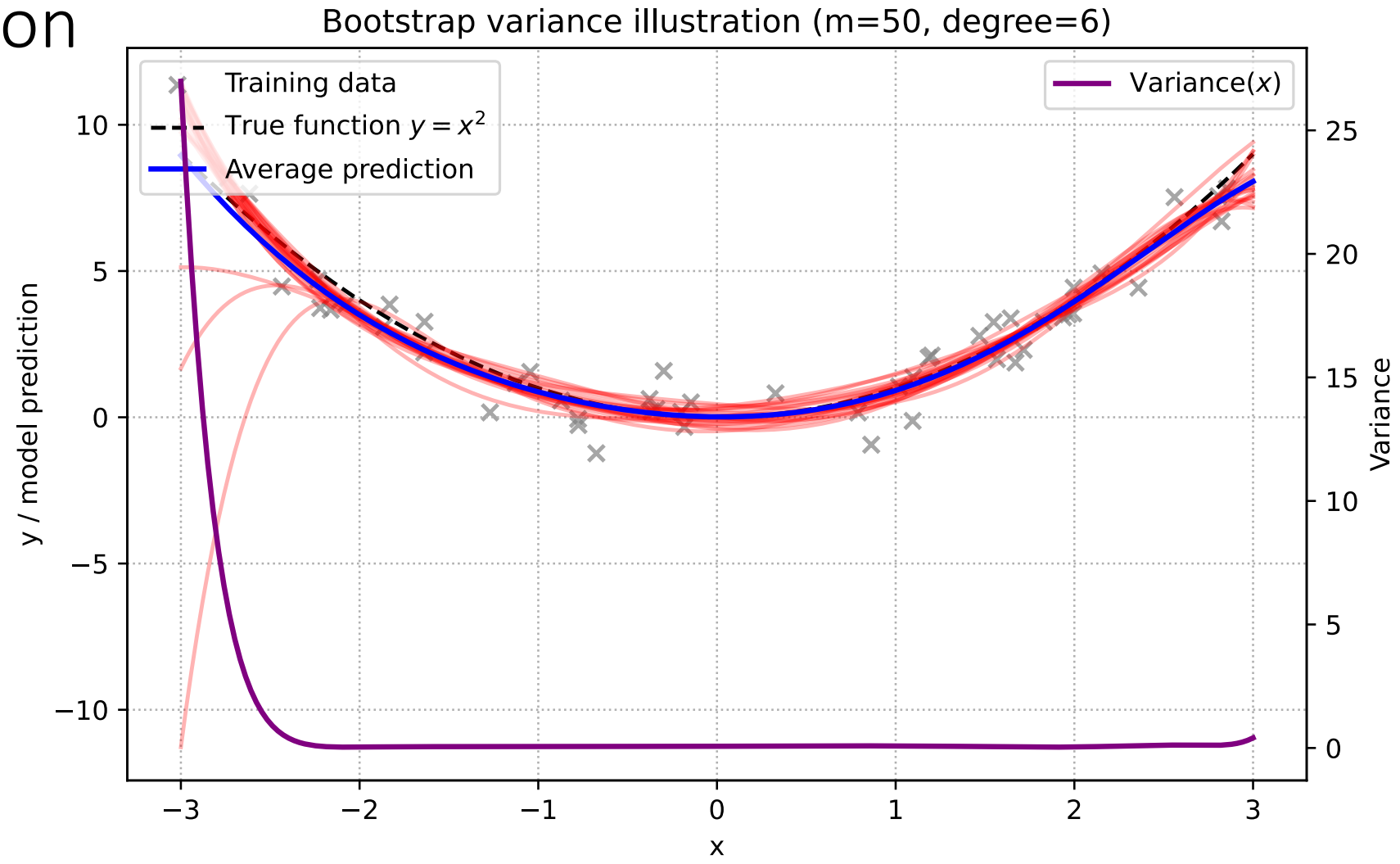
- Across all measurements:

$$\mathrm{Var} = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{B}\sum_{b=1}^{B}\left(\hat{y}_b^{(i)} - \bar{\hat{y}}^{(i)}\right)^2$$

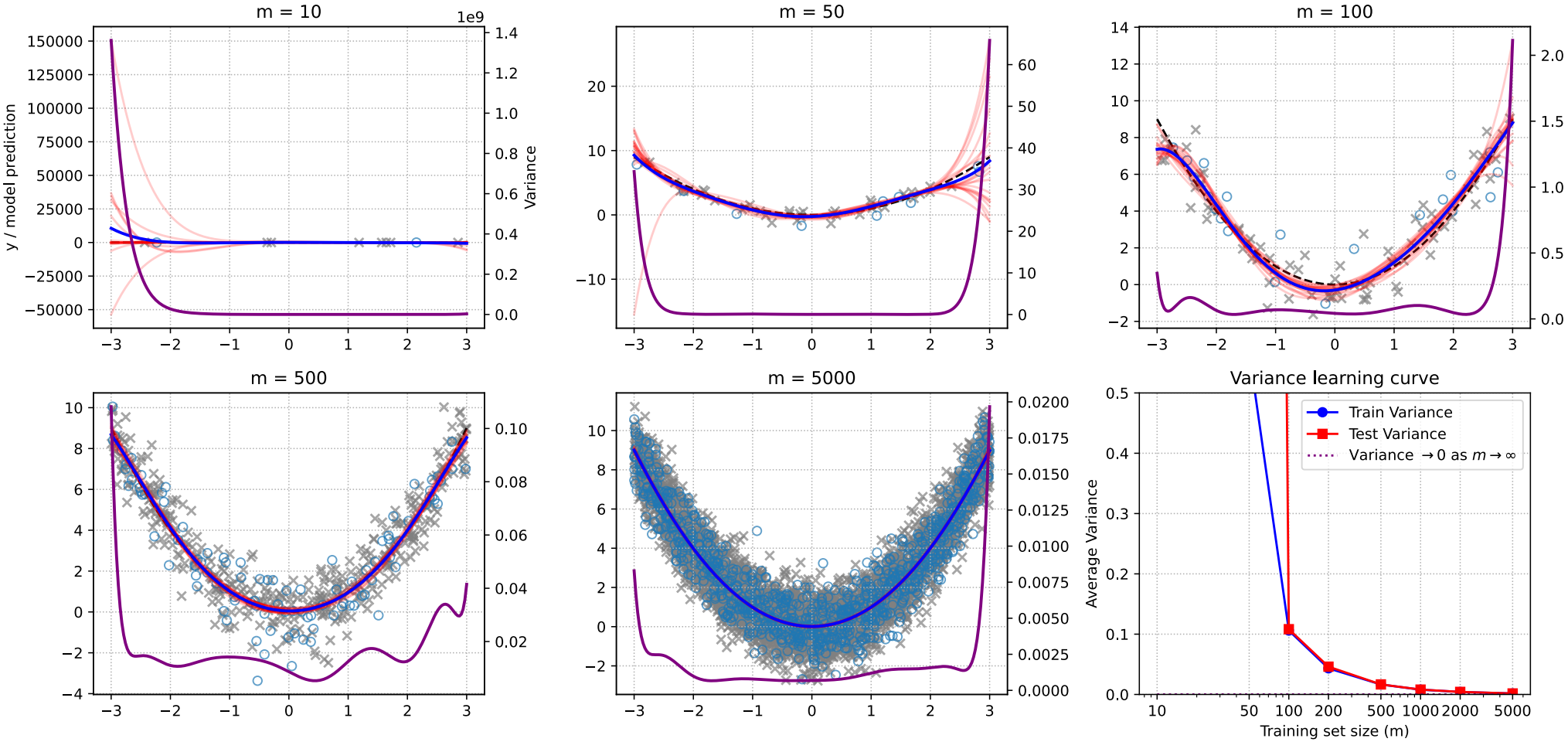
- Analogy: imagine throwing 20 darts at a dart board:
  - Bias is distance between average location of darts and bullseye ('intercept')
  - Variance is the spread in the darts around their average location

Illustration

6th order polynomial fits



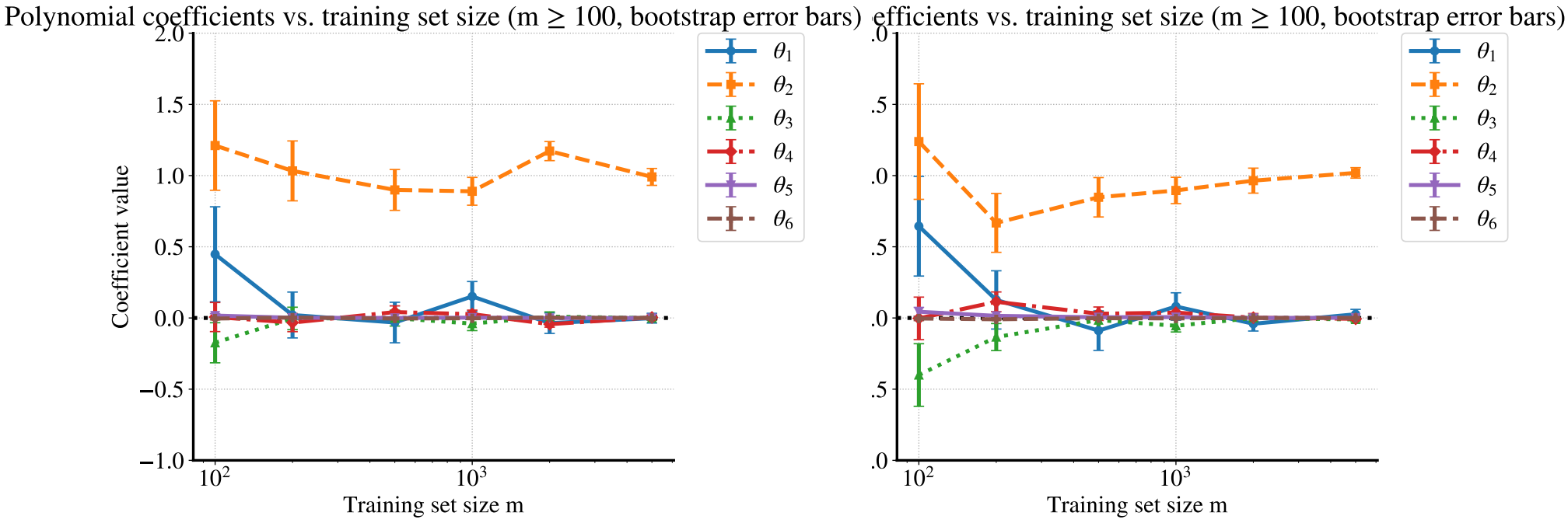
Variance behavior with increasing training set size (degree=6)



MSE on training and test data very different for small *m*, **overfitting**, but converge and reduce to zero for very large *m*.

Increasing *m* Does Feature Selection!

2 runs



Variance Summary

- For small/finite *m*, model may memorize training data but does not generalize to new unseen data: **overfitting**, especially when extrapolating.
- E.g., high-order polynomial 'wiggles' between selected training points
- With increasing *m*, 'wiggles' in training data and test data are similar (sampling ϵ irreducible noise) and variance-MSE for both converge
- For infinite *m*, variance approaches zero and higher-order terms reduce

- But what about for feasible finite *m*?

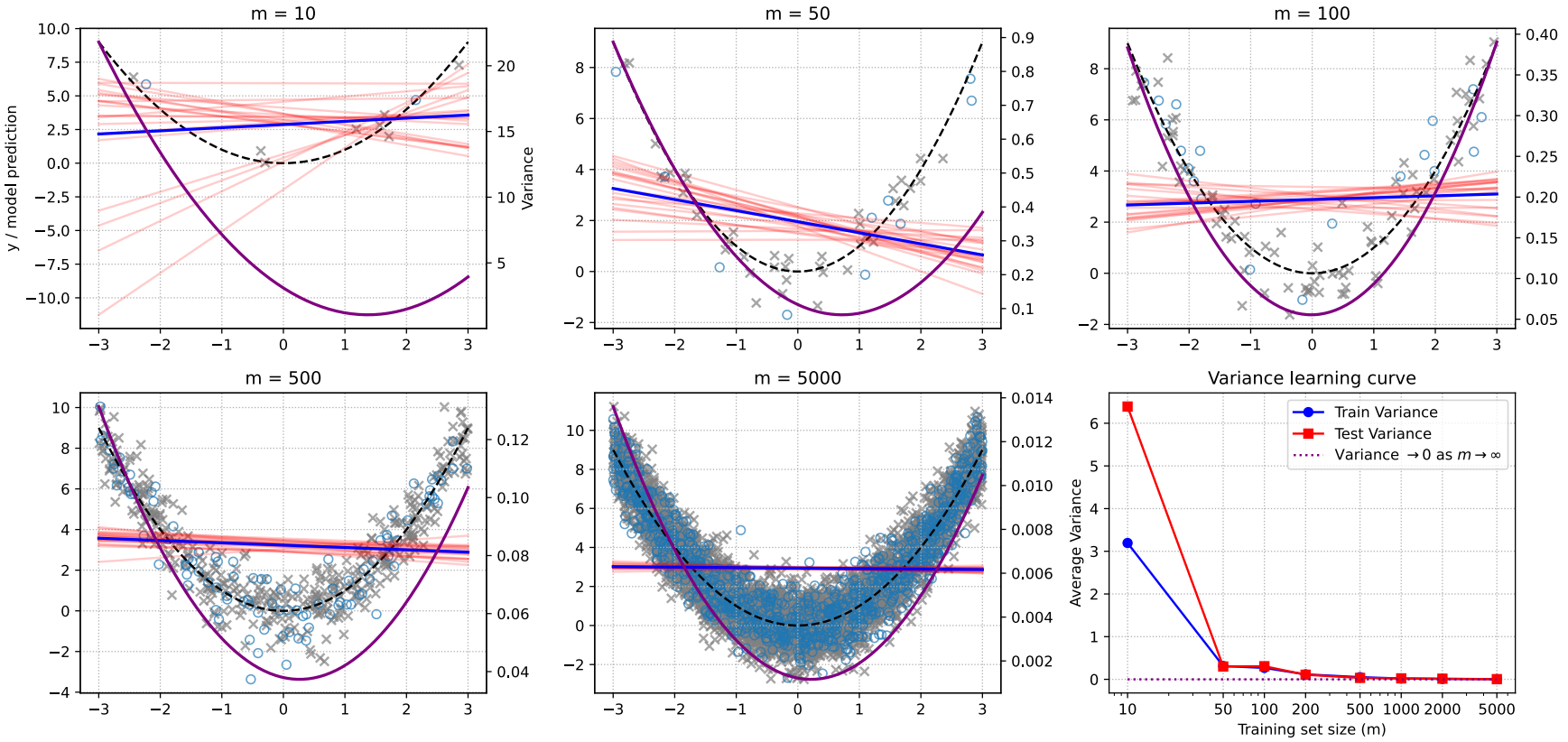
- **Answer**: Regularization (after a bit more patience)

Nuance

- Every model has *some* bias/variance/measurement errors

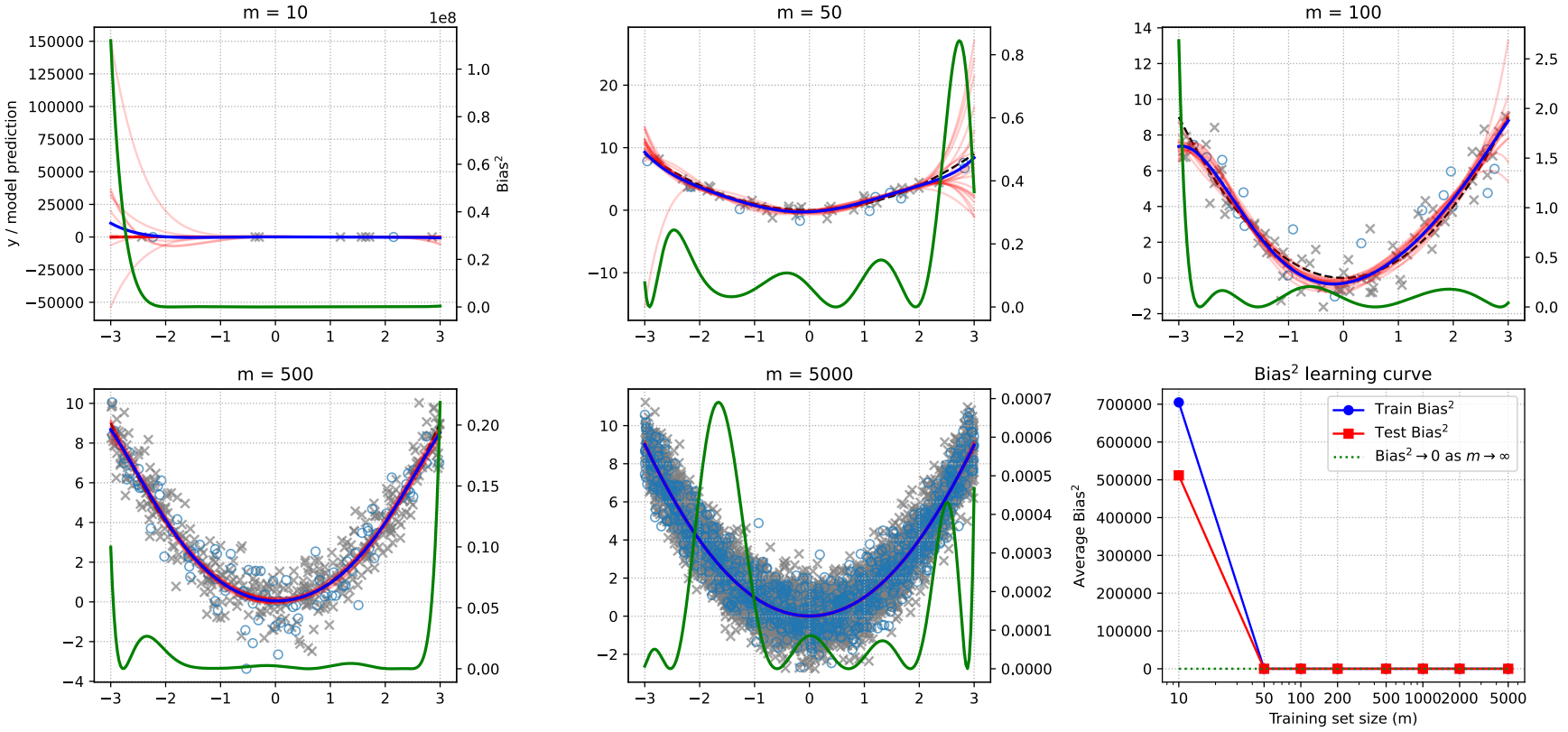
- High-bias, low-variance suggests underfitting
- Low-bias, high-variance suggests overfitting

High-Bias, Low-Variance Example



High-Variance, Low-Bias Example

Bias^2 behavior with increasing training set size (degree=6)



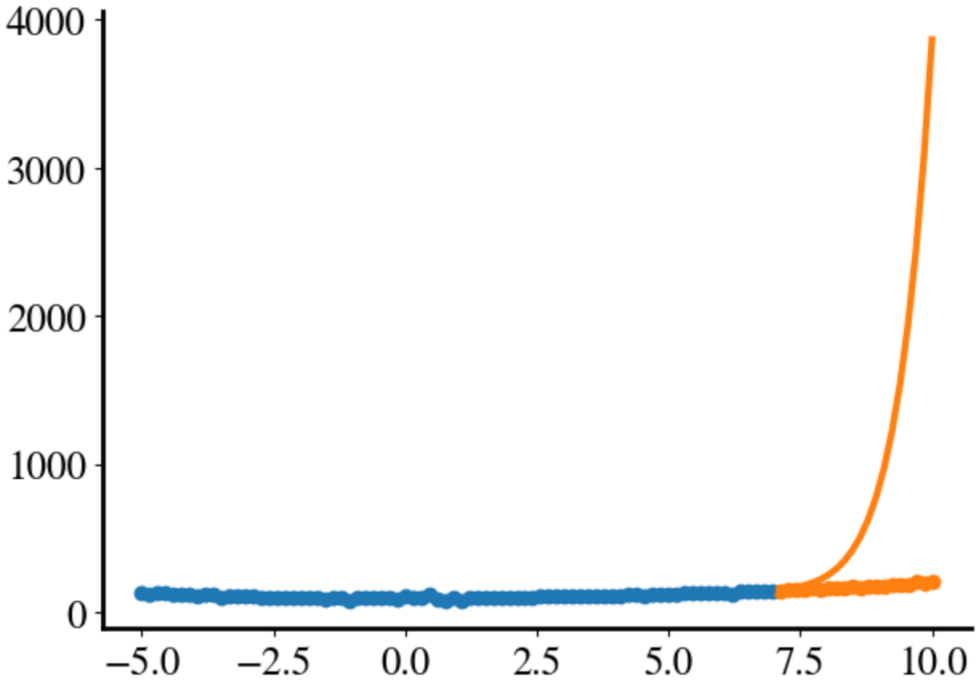
Cross-Validation

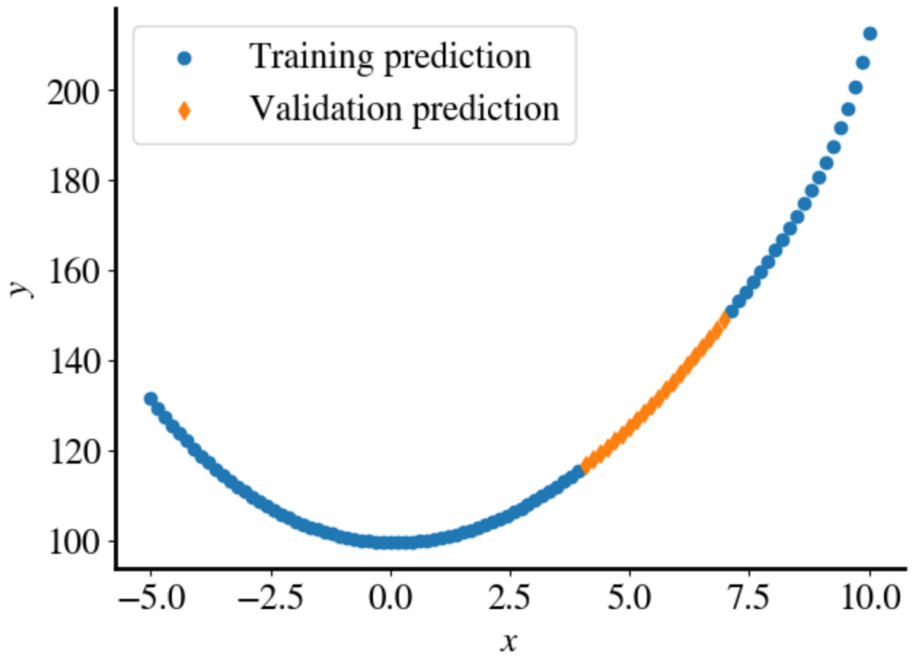
Cross-Validation

- Objective of ML algorithms is to find a *model* for a problem based on limited data (e.g., measurements) to make ***predictions*** for other conditions
- Model that fits training data but doesn't generalize to new data is useless
- In supervised learning, when we have (labeled) measurements we can 'fake' future measurements by splitting into training and hold-out validation data.
- Fit model to training data, predict for validation data, report MSE on the latter.
- One such split is generally not sufficient for small/finite *m*.
- **Solution**: *K*-fold **cross-validation**, i.e. do train:validation split *K* times to get better *statistics* on model generalizability.

Example: fit 15th–order polynomial to x^2 data

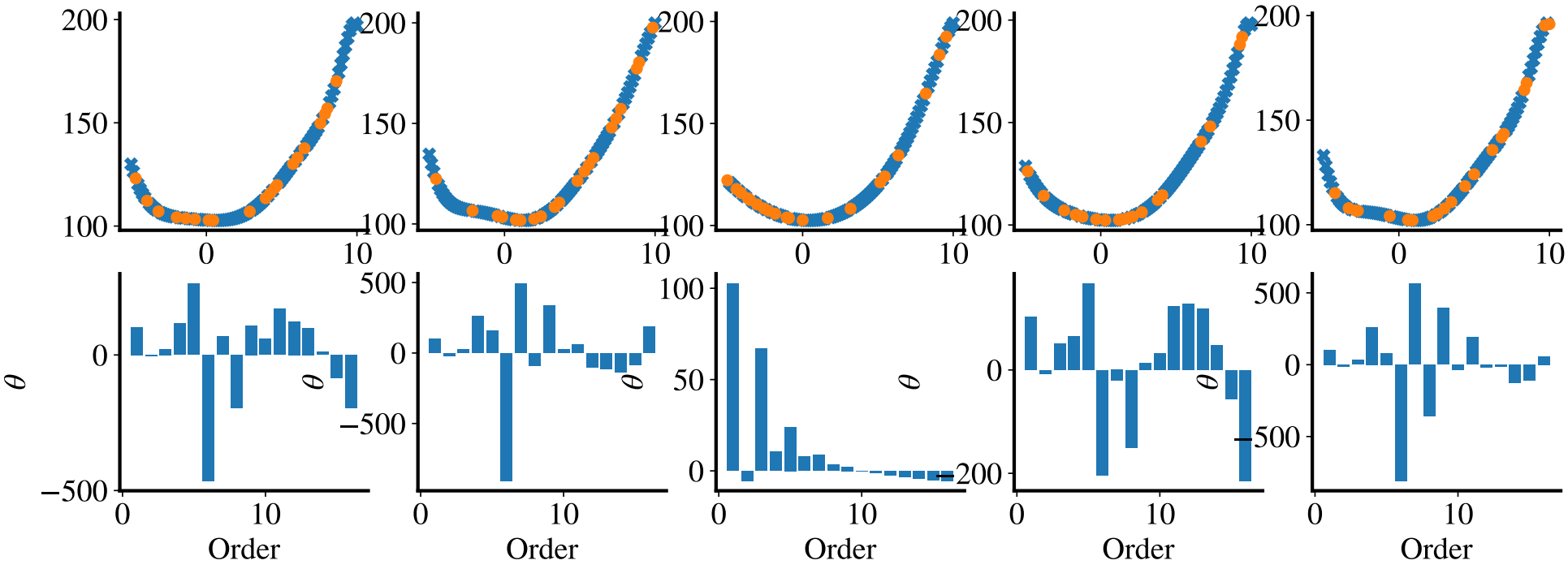
- Remember, extrapolation always problematic so randomize before split





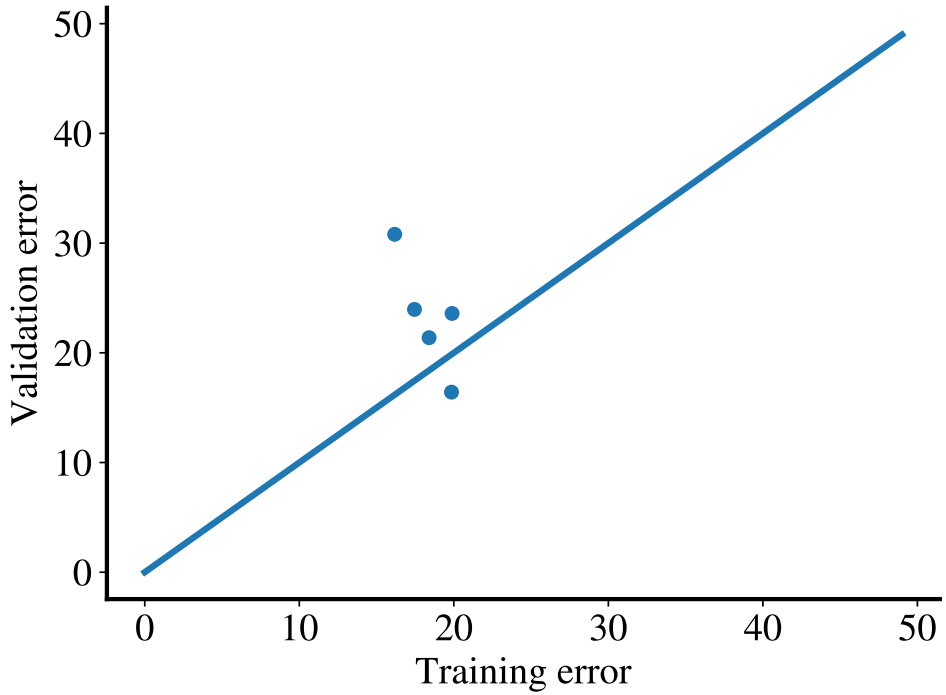
Example: fit 15th–order polynomial to x^2 data

- 5-fold cross-validation. Fits all look good, but completely different polynomials!



Example: fit 15^{th}–order polynomial to x^2 data

- 5-fold cross-validation: Validation errors generally larger than training errors:
- Overfitting!



Regularization

Ridge & Lasso Regularized Numeric or Logistic Regression

Problem Statement

- Remember, in both numerical & logistic regression we fit $y^{(i)} = f(\sum_j \theta_j x_j^{(i)})$, often for many features, *n*, whether 'independent' or polynomial.
- In ML, we generally don't know optimal *n* features *a priori*.
- When choosing too few features: underfitting/bias
- When choosing too many: overfitting/variance

- **Objective**: How do we find sufficient but fewest features x_j with parameters θ_j?
- **Alternatively**: how do we ***penalize*** too many features.
- Or, how to find fit with low MSE on θ_j but also smallest possible θ_j.

Conceptual Idea

- Remember that all ML algorithms fit some hypothesis model by minimizing error in fitting parameters θ_i, e.g., MSE: $J = \sum_{i=0}^{n} (\theta_i x_i - y)^2$
- But this does not penalize overly complex models with all $|\theta_i| \gg 0$
- How to improve?

Regularization

- Again, embarrassingly simple solution: add extra penalty/cost/error for large θ_i, with a user-tunable **regularization strength** $\boldsymbol{\lambda}$:
- Lasso, **L**east **A**bsolute **S**hrinkage and **S**election **O**perator:

$$J = \sum_{i=0}^{n} [(\theta_i x_i - y)^2 + \lambda |\theta_i| \,]$$

- Ridge regularization:

$$J = \sum_{i=0}^{n} [(\theta_i x_i - y)^2 + \lambda \theta_i^2 \,]$$

Full Python code for Ridge regression

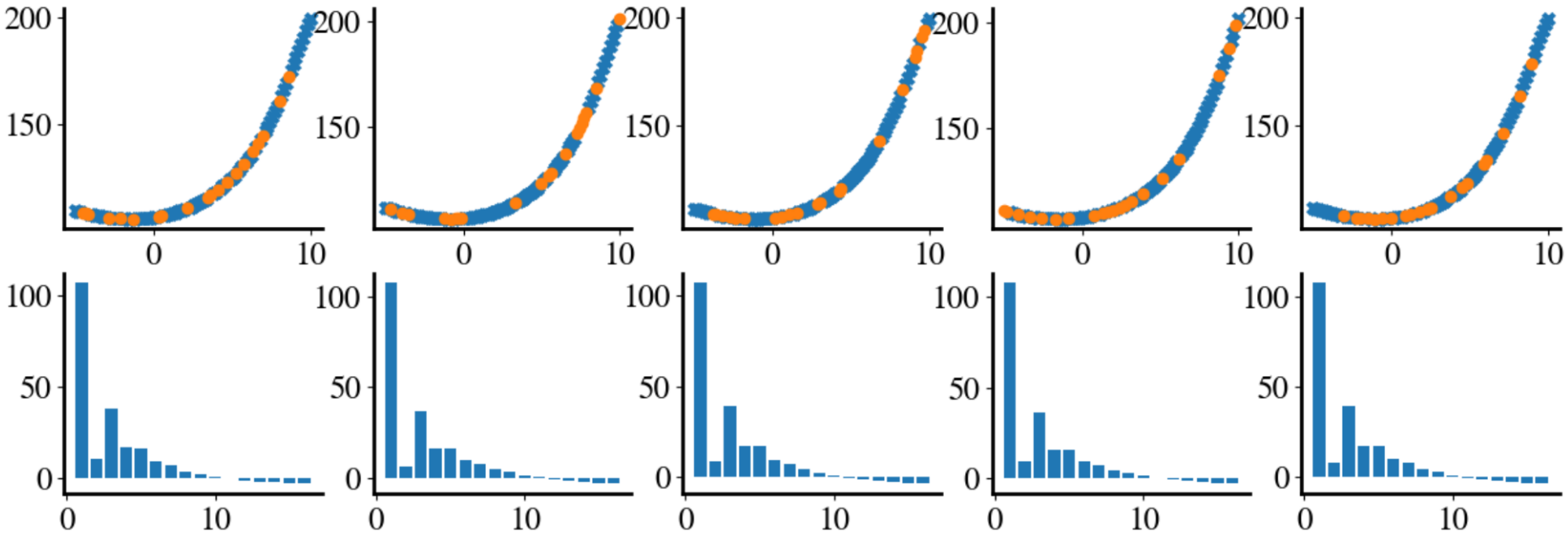
- Cost and cost gradient

```
1  def cost_ridge(theta, x, y, reglambda):
2    # Usual mean squared error cost:
3    cost = mean_squared_error(x.dot(theta),y) /2
4    # Additional regularization/penalty to keep theta small:
5    cost += reglambda*np.sum(theta[1:]**2)/(2*len(y))
6    return cost
7
8  def gradient_ridge(theta, x, y,reglambda):
9    grads = (x.dot(theta) - y).dot(x)/len(y)
10   # Regularization of theta_1 and higher:
11   grads[1:] += (reglambda/len(y)) * theta[1:]
12   return grads
13
```

```
13
14  def gradient_descent_ridge(theta,x, y,alpha,tolerance,reglambda):
15      nrsteps = 100000
16      m = len(y)
17
18      i=0
19      converged = False
20      diverged = False
21      while i < nrsteps-1 and not converged and not diverged:
22          costi = cost_ridge(theta,x,y,reglambda)
23          theta = theta -  alpha * gradient_ridge(theta,x,y,reglambda)
24
25          if i>0:
26              improvement = np.abs(1- costi/cost_sv)
27              if improvement < tolerance:
28                  converged = True
29              elif improvement > 1:
30                  print(i, improvement, "Errors are diverging:", cost, ">", cost_sv)
31                  diverged = True
32          i+=1
33          cost_sv = costi
34
35      if i==nrsteps-1:
36          print(i-1, costi, "Did not converge in", nrsteps, "steps. Increase nr of steps and try again.")
37      return theta
```

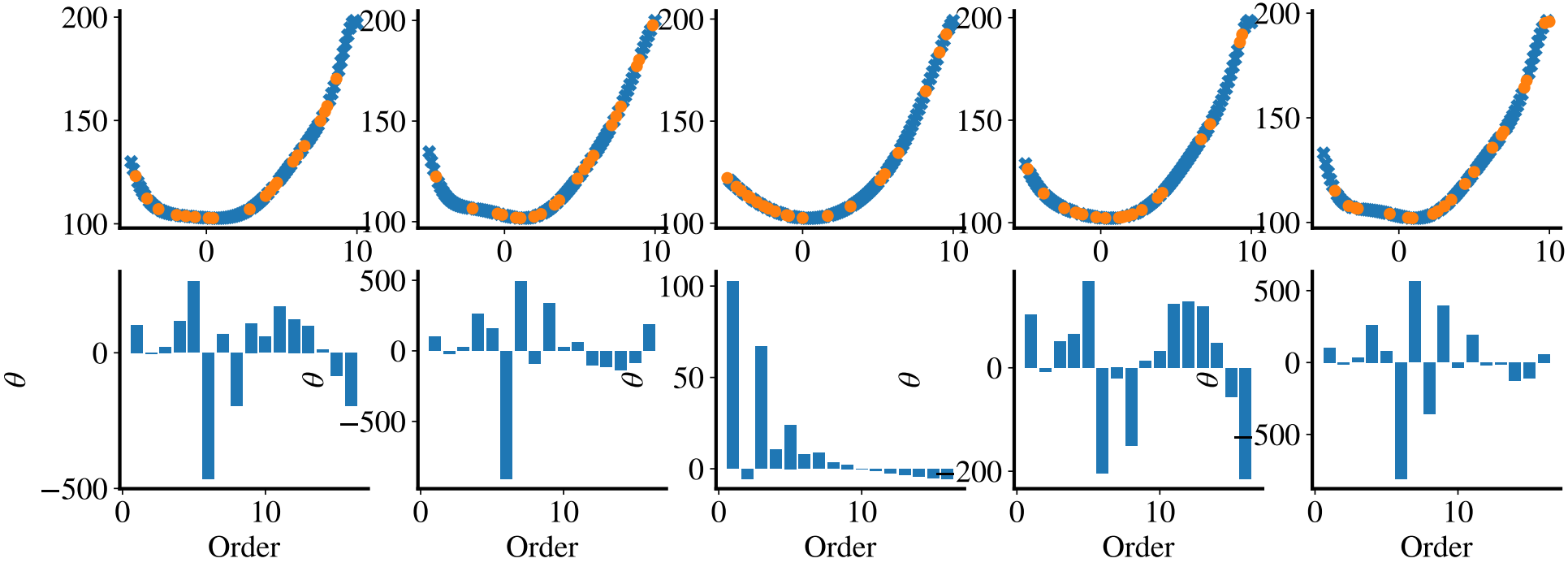
Example: Ridge Regression (still 15th order poly)

- 5-fold cross-validation with λ=1
- See how higher-order terms are much smaller (but not zero)



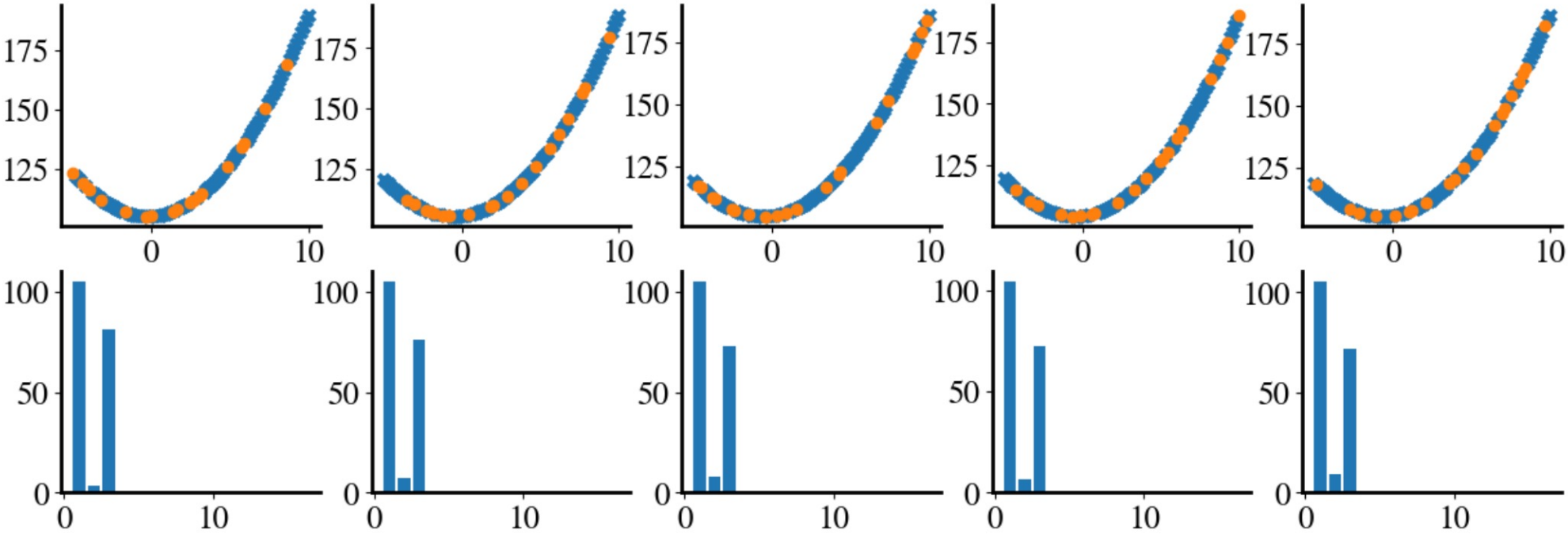
Remember, without regularization

- 5-fold cross-validation. Fits all look good, but completely different polynomials!



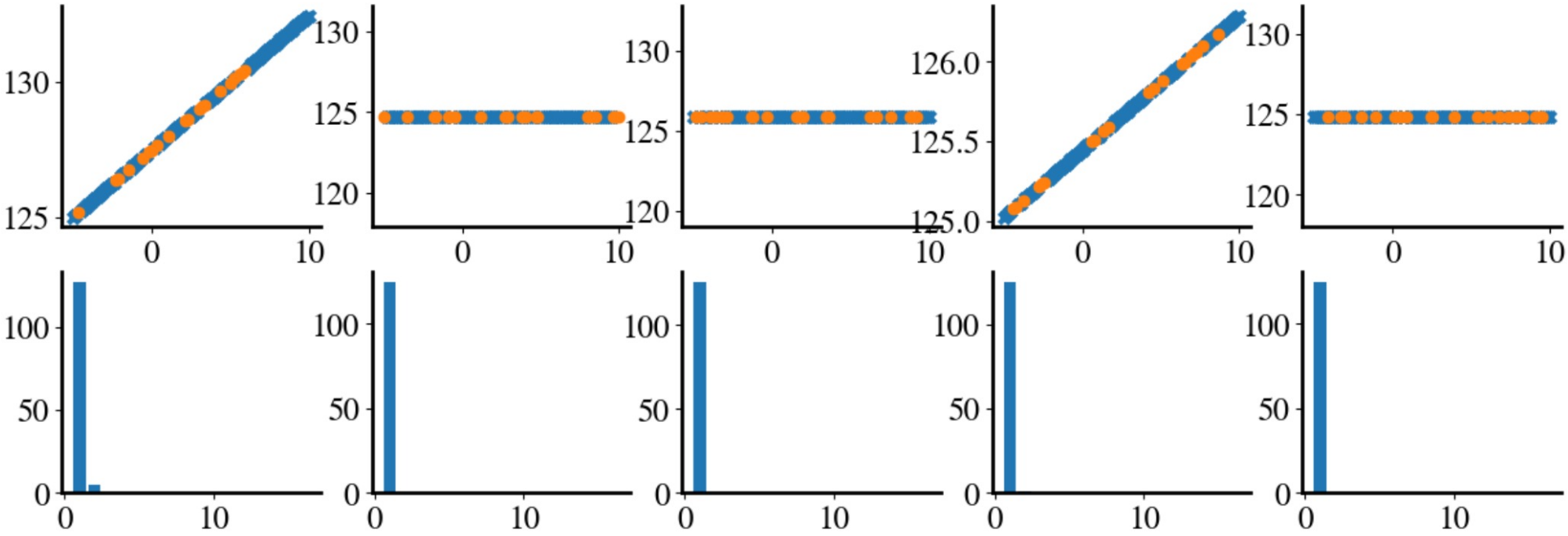
Example: Lasso Regression

- For λ=1, all terms beyond quadratic are exactly zero
- True **automatic feature selection!!**



Example: Lasso Regression

- For λ=10, all terms beyond linear are zero…
- Too much regularization: needs tuning



Subtlety

- What regularization strength should you pick?
- **Answer**: use cross-validation to determine

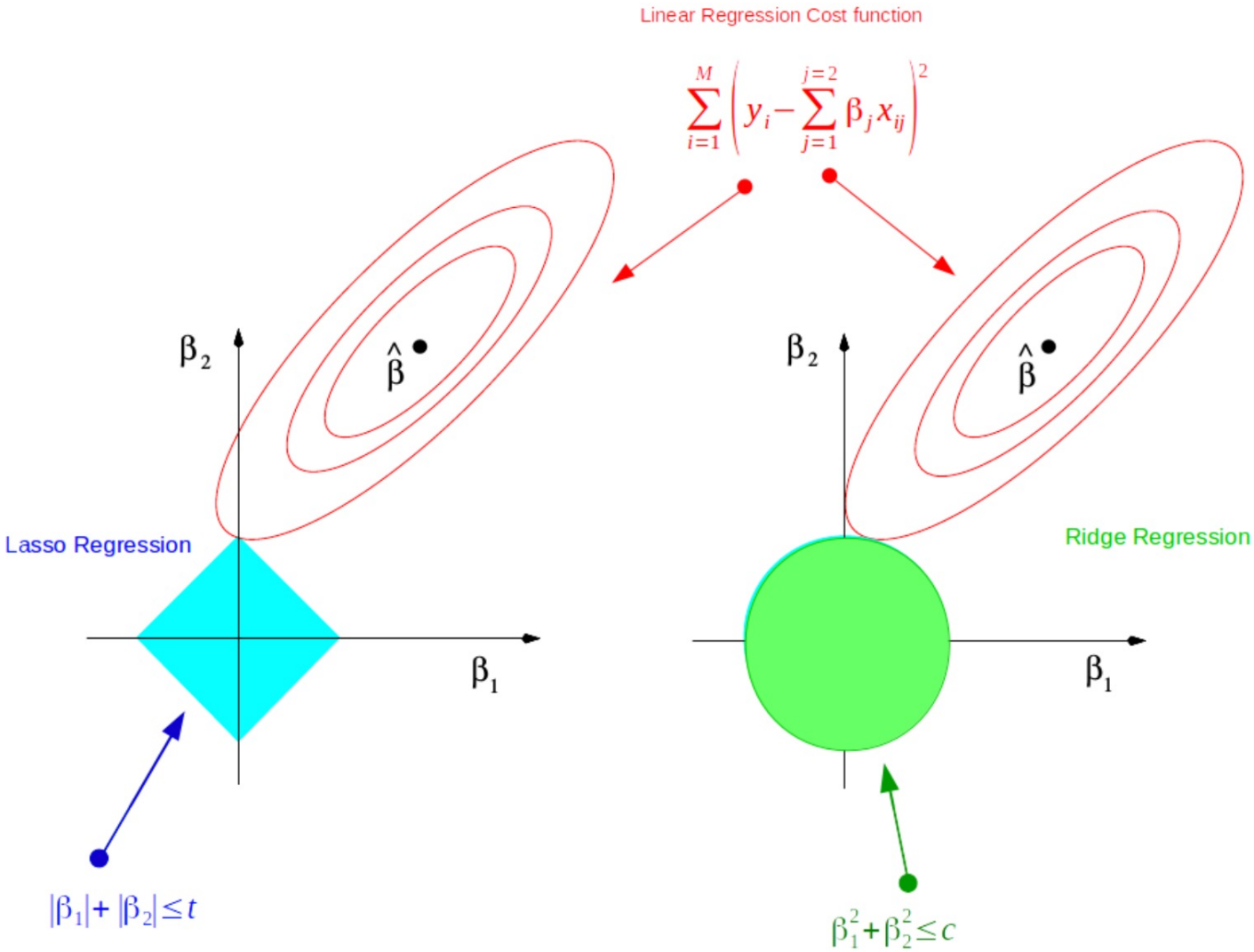
- Important subtlety: tuning hyperparameters like λ through cross-validation 'pollutes' the validation data

- You need a **triple** split: **training**:**validation** for cross-validation and hyperparameter tuning and another **test split** that is never seen in fitting model.

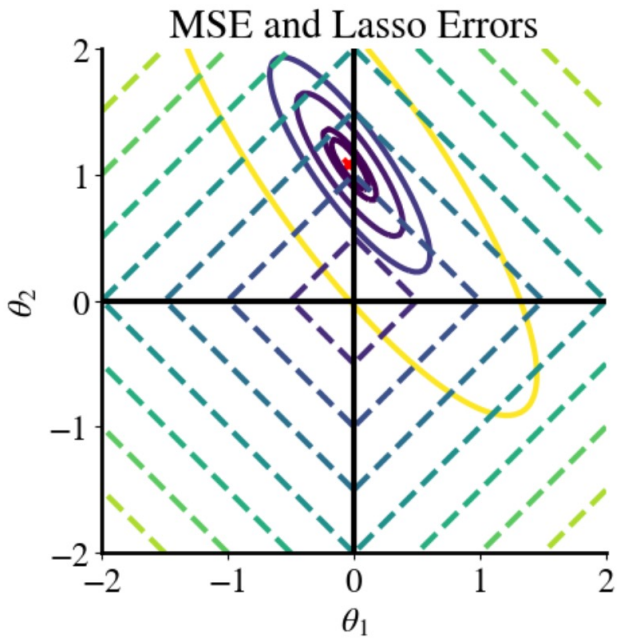
So why can Lasso, but not Ridge, eliminate features?

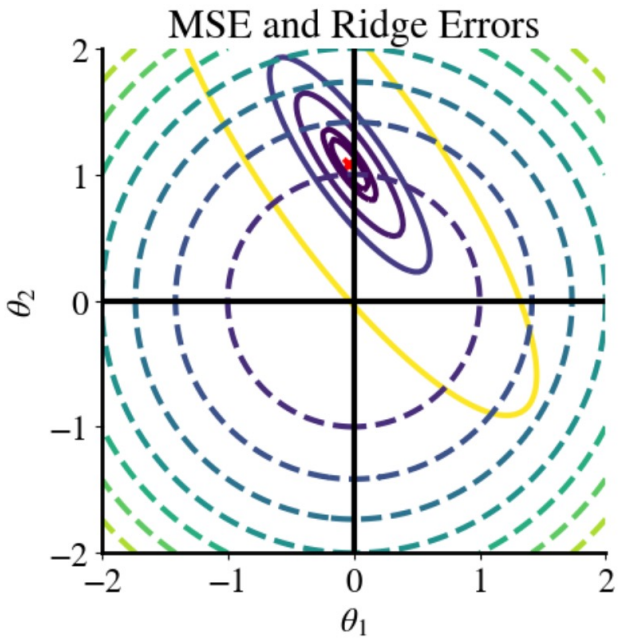
Dimension Reduction of Feature Space with LASSO

Least Absolute Shrinkage and Selection Operator



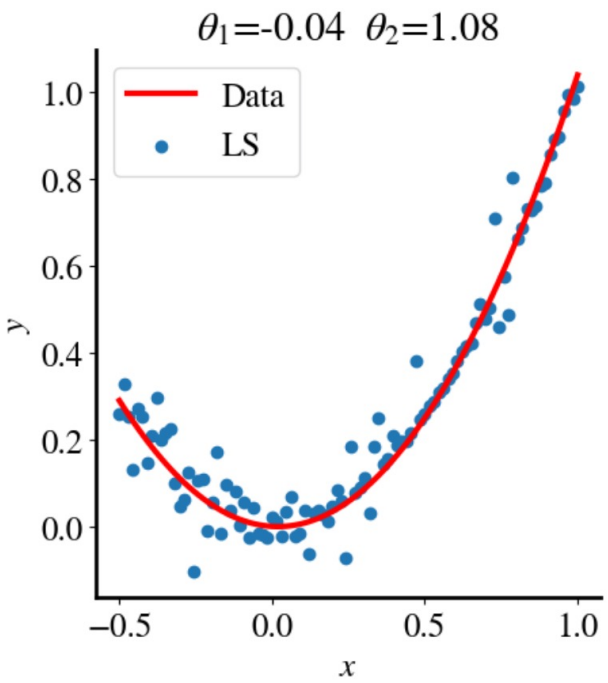
Without regularization



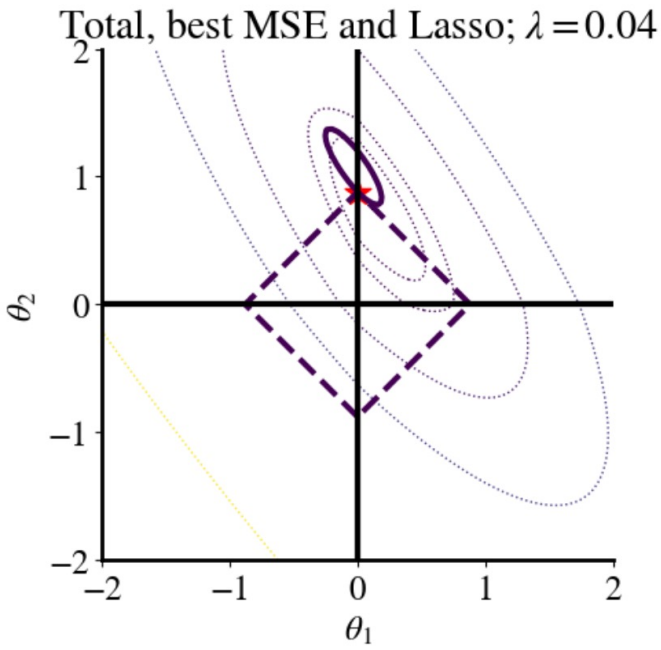


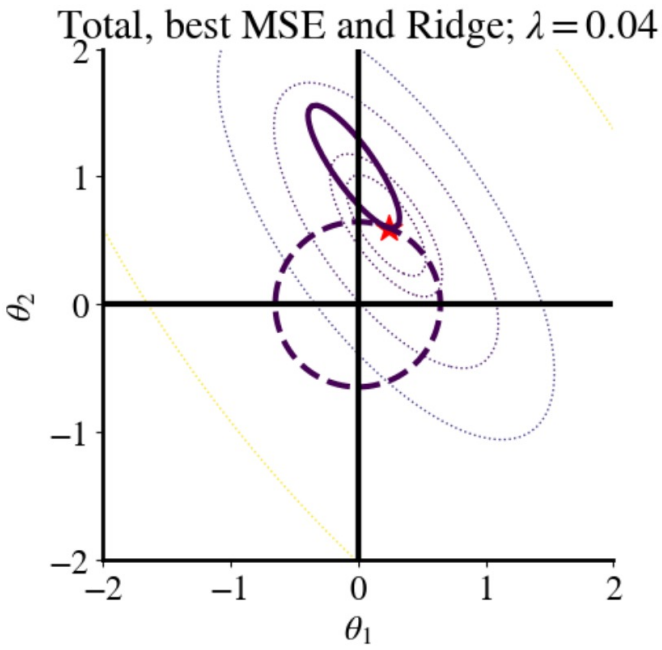
Linear term

Quadratic term

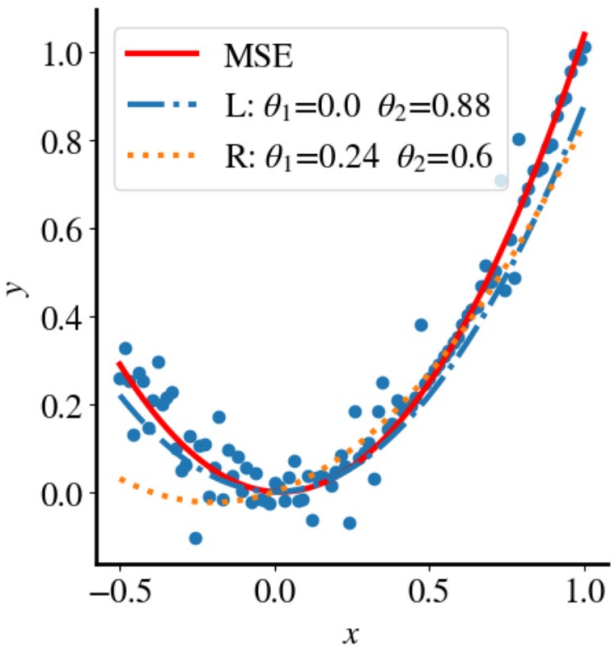


Regularization with $\lambda = 0.04$

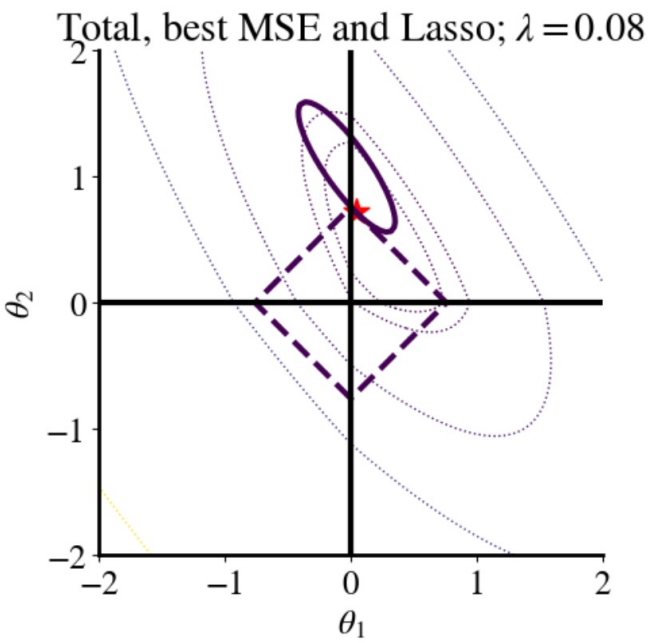


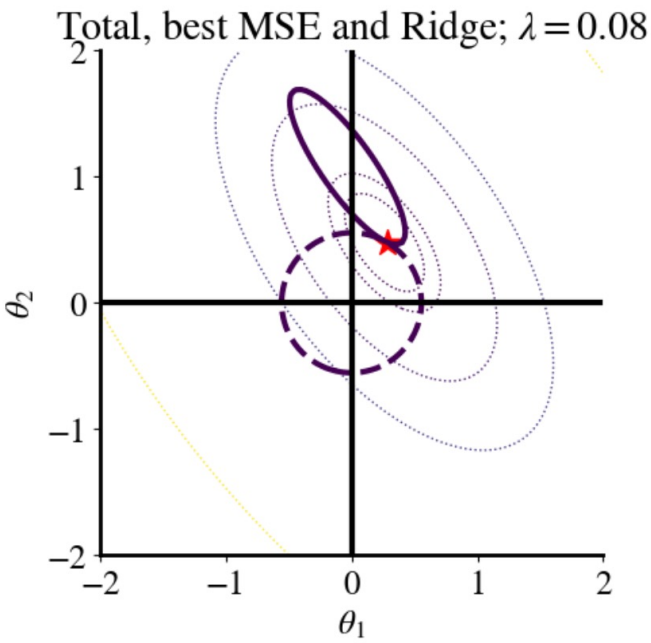


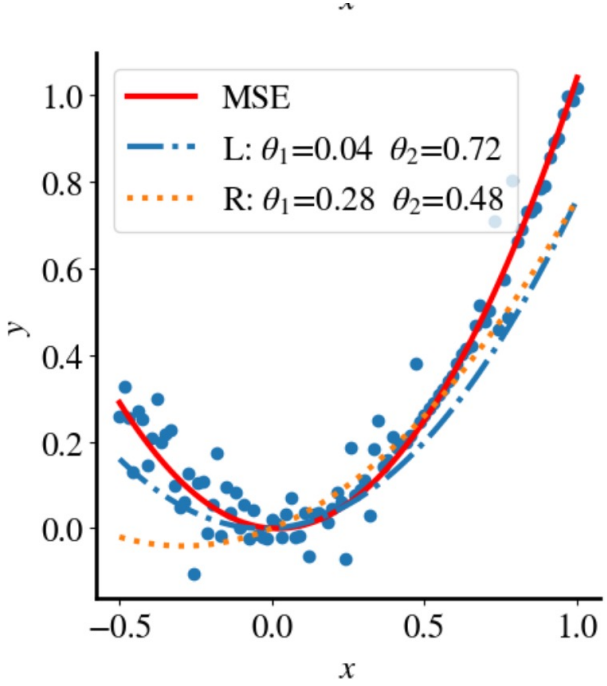
Lasso eliminated linear term!



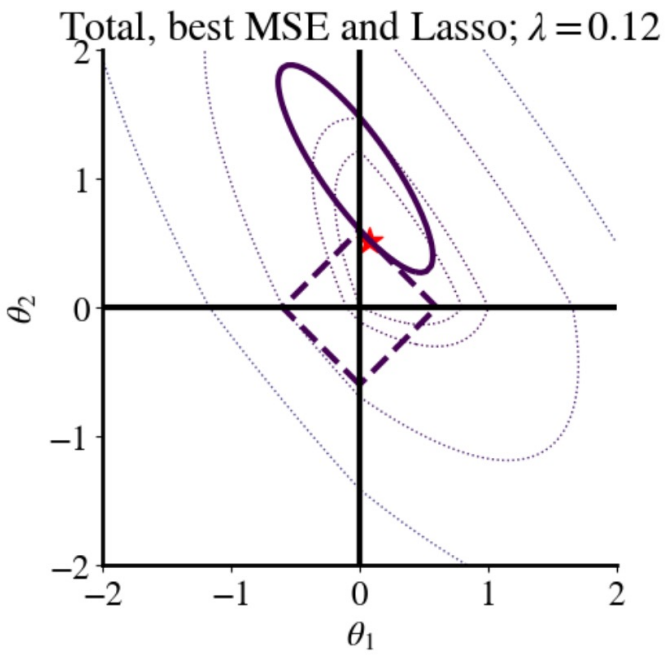
Regularization with $\lambda = 0.08$

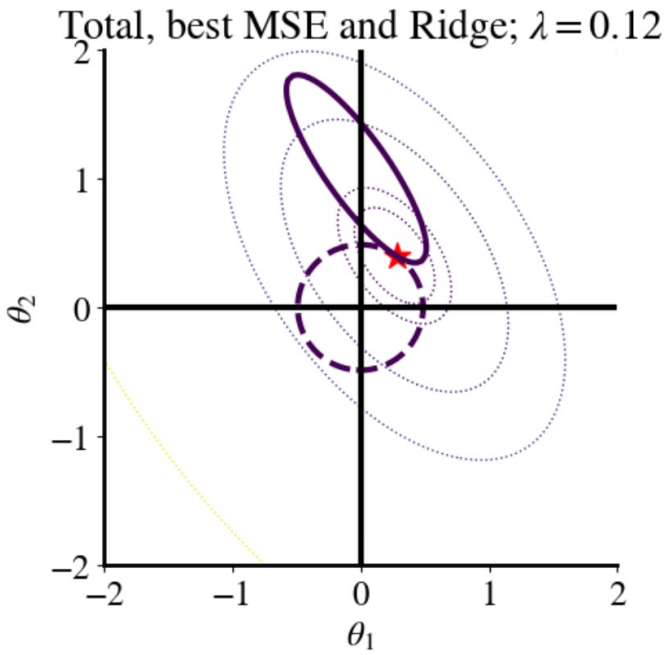


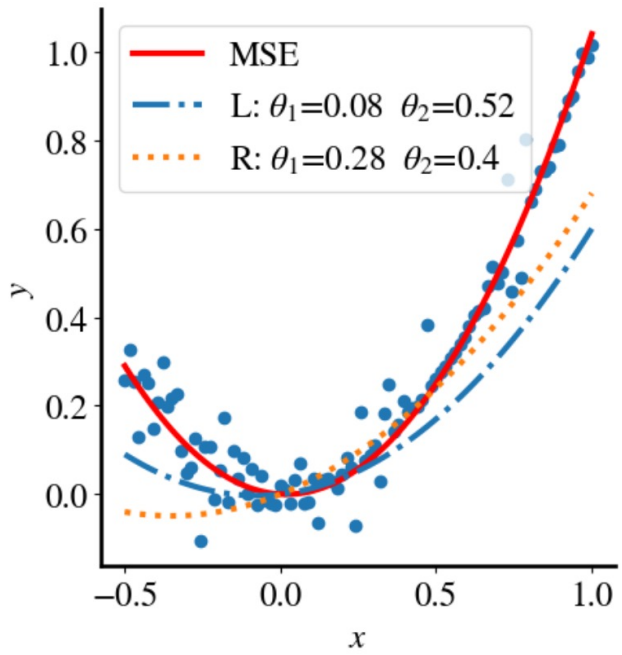




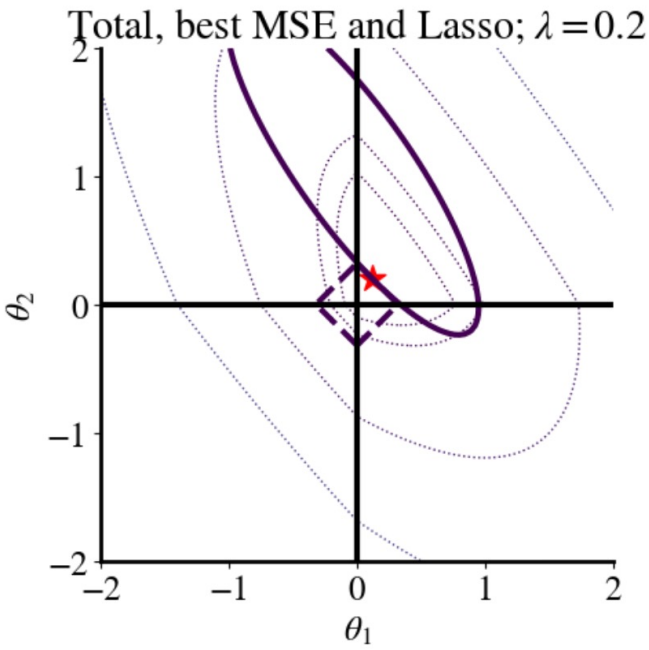
Regularization with $\lambda = 0.12$

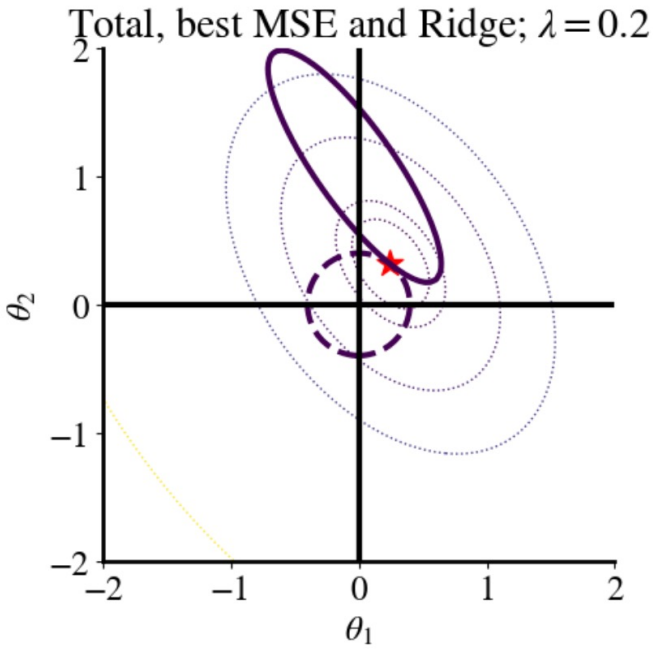


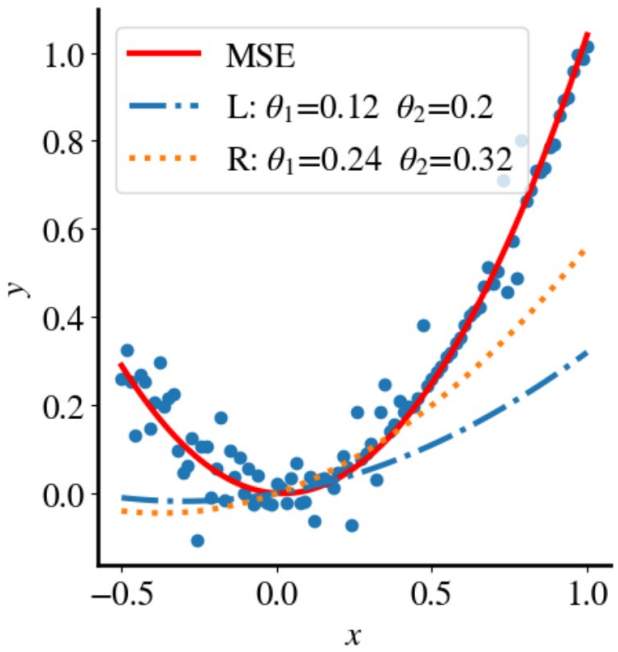




Regularization with $\lambda = 0.2$

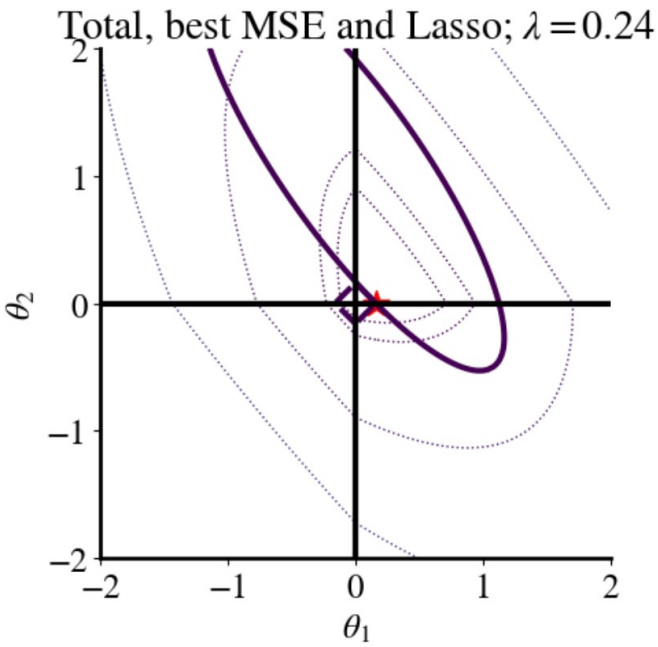


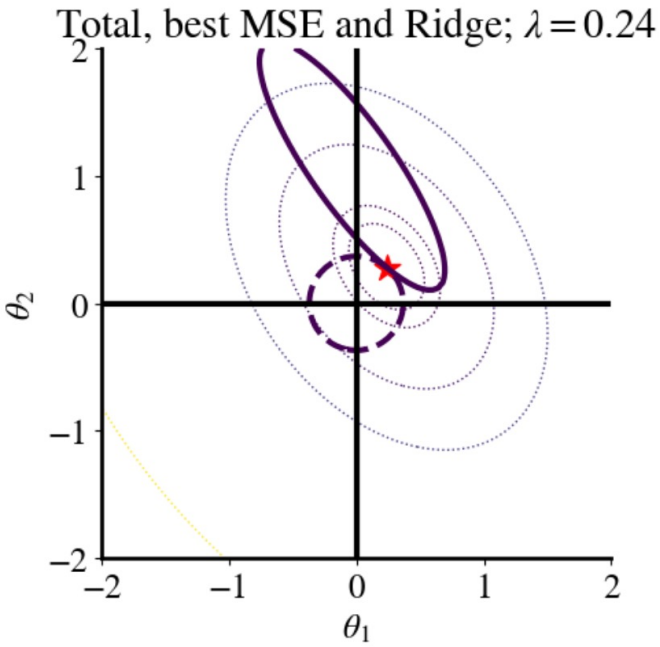


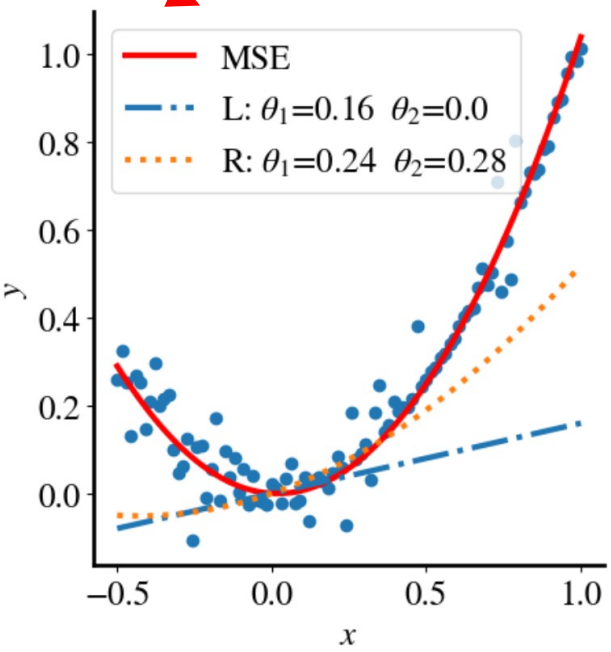


Regularization with $\lambda = 0.24$

Lasso eliminated quadratic term!







Summary

- **Bias**: offset of average model predictions vs 'true' values
- When predictions from a ML model show a high **bias**, model is too simple (underfitting). **Remedy**: higher complexity in model and/or nr of features.
- **Variance**: spread in predictions from repeated data sampling.
- When ML predictions show high variance, model too complex (overfitting).
- **Remedy**: **Regularization**, penalty for large fitting parameters.

- **Ridge**: easy to implement, reduces all θ_j, doesn't eliminate features
- **Lasso**: not differentiable but does true automatic feature selection!