

Regression

Purdue ME 597, Distributed Energy Resources

Kevin J. Kircher

Outline

Regression overview

Example: Polynomial interpolation

Example: Thermal resistance from thermostat data

What is machine learning?

- extracting information from data, usually to make predictions
- includes two general tasks:
 1. build a **model** from data
 2. **validate** the model (assess performance on **unseen** data)
- a **supervised** learning model predicts **targets** given **features**
 - ◇ **regression** predicts real-valued targets
 - ◇ **classification** predicts targets from finite sets such as $\{-1, 1\}$
- **unsupervised** learning creates a model of the data

Training and validation

- our true goal is to predict unseen data
- good practice: divide the full dataset into
 1. **training** data for choosing model **parameters**
 2. **validation** data for evaluating candidate models
- **overfit** models
 - ◇ perform well on training data, but not on validation data
 - ◇ likely won't **generalize** well to unseen data

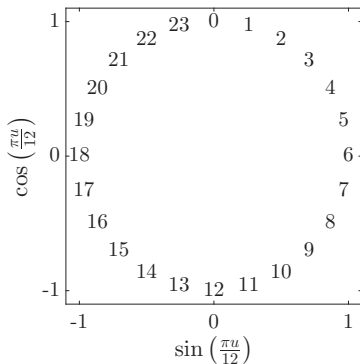
Embeddings

- denote raw input, output data by $u^{(i)} \in \mathcal{U}$, $v^{(i)} \in \mathcal{V}$
 - ◊ \mathcal{U} is the set of possible input data
 - ◊ \mathcal{V} is the set of possible output data
- we often transform each
 - ◊ raw input $u^{(i)}$ into a feature $x^{(i)} = \phi(u^{(i)}) \in \mathbf{R}^{n_x}$
 - ◊ raw output $v^{(i)}$ into a target $y^{(i)} = \psi(v^{(i)}) \in \mathbf{R}^{n_y}$
- $\phi : \mathcal{U} \rightarrow \mathbf{R}^{n_x}$ and $\psi : \mathcal{V} \rightarrow \mathbf{R}^{n_y}$ are called **embeddings**
- we'll assume $n_y = 1$ from now on, but methods extend to vector $y^{(i)}$

Faithful embeddings satisfy $\phi(u) \approx \phi(\tilde{u}) \iff u \approx \tilde{u}$

example: suppose $\mathcal{U} = \{0, \dots, 23\}$ contains hour-of-day data

- problem: midnight \approx 11 PM, but $0 \not\approx 23$
- idea: choose $\phi(u) = \sin(\frac{\pi u}{12})$, so $\phi(0) \approx \phi(23)$
- new problem: midnight $\not\approx$ noon, but $\sin(0) = \sin(\pi)$
- faithful embedding: $\phi(u) = (\sin(\frac{\pi u}{12}), \cos(\frac{\pi u}{12}))$



Standardization and normalization

- model training tends to work better when feature scales are similar
- **standardization** and **normalization** are two ways to rescale data
- standardization embedding, $x_j^{(i)} = (u_j^{(i)} - \mu_j)/\sigma_j$ with

$$\mu_j = \frac{1}{n} \sum_{i=1}^n u_j^{(i)}, \quad \sigma_j = \sqrt{\frac{1}{n} \sum_{i=1}^n (u_j^{(i)} - \mu_j)^2},$$

makes $\frac{1}{n} \sum_{i=1}^n x_j^{(i)} = 0$, $\sqrt{\frac{1}{n} \sum_{i=1}^n (x_j^{(i)})^2} = 1$

- normalization embedding, $x_j^{(i)} = (u_j^{(i)} - \underline{u}_j)/(\bar{u}_j - \underline{u}_j)$ with

$$\underline{u}_j = \min \{u_j^{(1)}, \dots, u_j^{(n)}\}, \quad \bar{u}_j = \max \{u_j^{(1)}, \dots, u_j^{(n)}\},$$

makes each $x_j^{(i)}$ lie between zero and one

Models

- a model is a function $f_\theta : \mathbf{R}^{n_x} \rightarrow \mathbf{R}$
- notation f_θ emphasizes dependence on parameters $\theta \in \mathbf{R}^{n_\theta}$
- a good model predicts $f_\theta(x) \approx y$ for unseen (x, y) data
- many possible model structures:
 - ◇ linear models
 - ◇ nearest neighbors
 - ◇ neural networks
 - ◇ regression trees
 - ◇ support vector machines
 - ◇ ...

Linear models

- **linear** models have the form $f_{\theta}(x) = \theta^{\top}x$ (so $n_{\theta} = n_x$)
- usually, we embed with a constant feature $x_1 = 1$, so

$$f_{\theta}(x) = \theta_1 + \theta_2 x_2 + \dots + \theta_{n_x} x_{n_x}$$

- in training, we choose θ such that each $f_{\theta}(x^{(i)}) \approx y^{(i)}$:

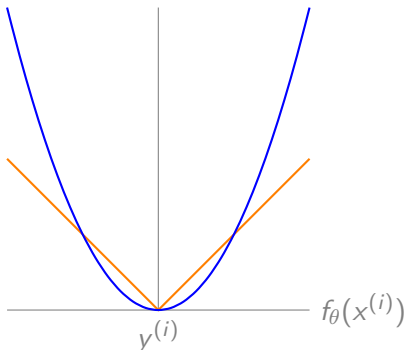
$$\begin{bmatrix} \theta^{\top} x^{(1)} \\ \vdots \\ \theta^{\top} x^{(n)} \end{bmatrix} = \begin{bmatrix} (x^{(1)})^{\top} \theta \\ \vdots \\ (x^{(n)})^{\top} \theta \end{bmatrix} = \begin{bmatrix} (x^{(1)})^{\top} \\ \vdots \\ (x^{(n)})^{\top} \end{bmatrix} \theta \approx \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

- can write this as $X\theta \approx Y$, where

$$X = \begin{bmatrix} (x^{(1)})^{\top} \\ \vdots \\ (x^{(n)})^{\top} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & \dots & x_{n_x}^{(1)} \\ \vdots & & \vdots \\ x_1^{(n)} & \dots & x_{n_x}^{(n)} \end{bmatrix}, \quad Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Loss functions

- a loss function $\ell_i : \mathbf{R}^{n_\theta} \rightarrow \mathbf{R}$ quantifies how badly $f_\theta(x^{(i)})$ misses $y^{(i)}$
- square loss: $\ell_i(\theta) = (f_\theta(x^{(i)}) - y^{(i)})^2$
- absolute loss: $\ell_i(\theta) = |f_\theta(x^{(i)}) - y^{(i)}|$



Mean loss

- the **mean loss** $\mathcal{L} : \mathbf{R}^{n_\theta} \rightarrow \mathbf{R}$ is

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell_i(\theta)$$

- with square loss, \mathcal{L} is the mean square error (MSE):

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (f_\theta(x^{(i)}) - y^{(i)})^2$$

- with absolute loss, \mathcal{L} is the mean absolute error (MAE):

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n |f_\theta(x^{(i)}) - y^{(i)}|$$

Regularization

- a model f_θ is **insensitive** if $f_\theta(x) \approx f_\theta(\tilde{x})$ when $x \approx \tilde{x}$
- insensitive models tend to generalize better
- a **regularizer** $r : \mathbf{R}^{n_\theta} \rightarrow \mathbf{R}$ quantifies the sensitivity of f_θ
- for linear model $f_\theta(x) = \theta^\top x$,

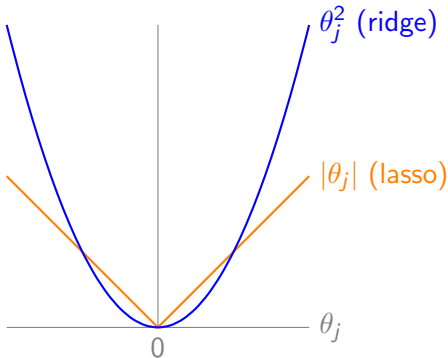
$$\frac{\partial f_\theta(x)}{\partial x_j} = \frac{\partial}{\partial x_j} (\theta_1 x_1 + \cdots + \theta_j x_j + \cdots + \theta_{n_x} x_{n_x}) = \theta_j$$

$\implies f_\theta$ should be less sensitive if θ is smaller

- two common regularizers:
 - ◇ **ridge** or ℓ_2 : $r(\theta) = \theta_1^2 + \cdots + \theta_{n_\theta}^2$
 - ◇ **lasso** or ℓ_1 : $r(\theta) = |\theta_1| + \cdots + |\theta_{n_\theta}|$
- with constant feature $x_1 = 1$, typically omit θ_1 from regularization

Lasso regularization promotes sparsity

- a vector is **sparse** if most of its elements are zero
- if $\theta_j = 0$ in linear model $f_{\theta}(x) = \theta_1 x_1 + \dots + \theta_{n_x} x_{n_x}$, then
 - ◊ feature x_j has no influence on predictions
 - ◊ might as well omit x_j and simplify model
- lasso regularizer $|\theta_1| + \dots + |\theta_{n_x}|$ promotes sparsity of θ



Training models via regularized mean loss minimization

- choosing θ is called **training** the model
- two competing goals:
 1. fit training data well by making $\mathcal{L}(\theta)$ small
 2. generalize well by making $r(\theta)$ small
- to balance them, choose θ to minimize $\mathcal{L}(\theta) + \lambda r(\theta)$
- regularization **hyperparameter** $\lambda \geq 0$ governs tradeoff: 1 vs. 2

Training linear models, $f_{\theta}(x) = \theta^{\top} x$

- with linear model, square loss, and ℓ_2 regularization,

$$\begin{aligned}\mathcal{L}(\theta) + \lambda r(\theta) &= (X\theta - Y)^{\top}(X\theta - Y)/n + \lambda\theta^{\top}\theta \\ &= (\theta^{\top}X^{\top}X\theta - 2Y^{\top}X\theta + Y^{\top}Y)/n + \lambda\theta^{\top}I\theta \\ &= [\theta^{\top}(X^{\top}X + n\lambda I)\theta - 2Y^{\top}X\theta - Y^{\top}Y]/n\end{aligned}$$

- since $\mathcal{L}(\theta) + \lambda r(\theta) = \theta^{\top}P\theta + q^{\top}\theta + r$, gradient is $2P\theta + q$

$$2P\theta^* + q = 0 \iff \theta^* = (X^{\top}X + n\lambda I)^{-1}X^{\top}Y$$

- inverse always exists if $\lambda > 0$
- if $\lambda = 0$ (no regularization), X needs linearly independent columns
- to omit θ_1 regularization, replace I_{n_x} by $E^{\top}E$, where

$$E = \begin{bmatrix} 0 & I_{n_x-1} \end{bmatrix} \in \mathbf{R}^{n_x-1 \times n_x}$$

- no formula for absolute loss or ℓ_1 regularization, but convex problems

Validation

- true goal: choose embeddings ϕ , ψ and model f_θ such that

$$\psi^{-1}(f_\theta(\phi(u))) \approx v$$

for unseen (u, v) data

- we can't test this until unseen data are revealed
- instead, validation
 - ◇ holds back some data from training
 - ◇ evaluates performance on held-back data

Out-of-sample validation

- split data into training and validation (say, 70/30, 80/20, or 90/10)
- choose starter embeddings, model structure, hyperparameters
- repeat:
 - ◇ change embeddings, model structure, or hyperparameters
 - ◇ train model in training data
 - ◇ if performance degrades in validation data, revert changeuntil performance in validation data is acceptable

K-folds validation

- divide data into K **folds** (for some integer K between 2 and n)
- for $k = 1, \dots, K$
 - ◊ train on all data except fold k
 - ◊ compute mean loss on fold k , $\mathcal{L}_k(\theta)$
- evaluate performance using mean (over all folds) of mean losses,

$$\bar{\mathcal{L}}(\theta) = \frac{1}{K} \sum_{i=1}^K \mathcal{L}_k(\theta)$$

Summary: How to build regression models

- choose a loss function to rate how badly predictions miss targets
- choose a regularizer to rate model sensitivity
- split data into training and validation
- repeat:
 - ◇ choose how to embed raw data into features and targets
(**feature engineering**)
 - ◇ choose a model structure and hyperparameters
(**model selection** and **hyperparameter tuning**)
 - ◇ choose model parameters to minimize regularized mean training loss
(**training**)until validation performance is acceptable
- retrain model on all (training + validation) data

Outline

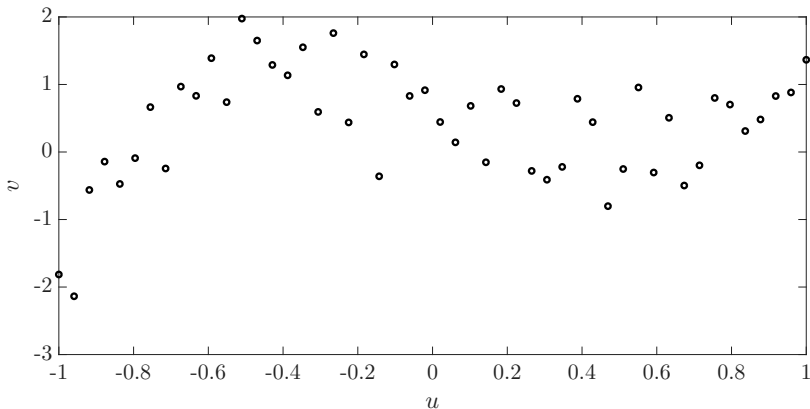
Regression overview

Example: Polynomial interpolation

Example: Thermal resistance from thermostat data

Problem

- goal: approximate an unknown function $g : [-1, 1] \rightarrow \mathbf{R}$
- $n = 50$ noisy raw data points:
 - ◊ inputs $u^{(1)}, \dots, u^{(50)}$ are evenly spaced on $[-1, 1]$
 - ◊ outputs $v^{(1)}, \dots, v^{(50)}$ are noisy observations of $g(u^{(1)}), \dots, g(u^{(50)})$
- want to predict unseen function values between the 50 points



Solution approach

- linear model, square loss
- high-order polynomial feature embeddings with normalization
- standardization target embedding
- K -folds validation with $K = 10$
- two stages of regularization with hyperparameter tuning:
 - ◇ lasso to select important features
 - ◇ ridge to reduce model sensitivity to selected features

Embeddings

- start with 25th degree polynomial feature embedding:

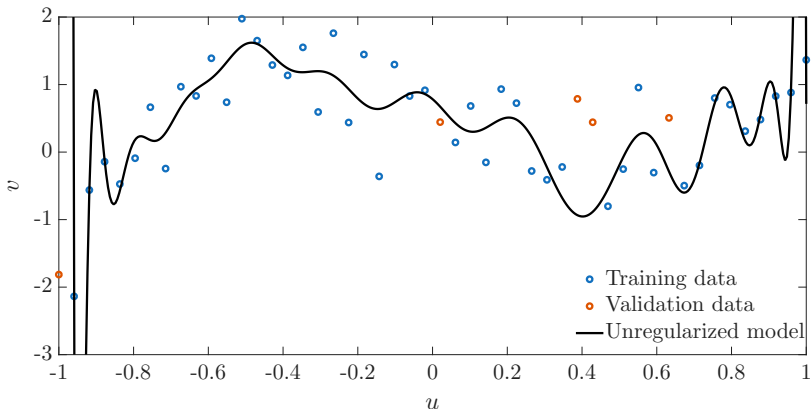
$$z^{(i)} = \begin{bmatrix} 1 \\ u^{(i)} \\ \vdots \\ (u^{(i)})^{25} \end{bmatrix}$$

- normalize the $z_2^{(i)}, \dots, z_{26}^{(i)}$:

$$x^{(i)} = \begin{bmatrix} 1 \\ (z_2^{(i)} - \underline{z}_2)/(\bar{z}_2 - \underline{z}_2) \\ \vdots \\ (z_j^{(i)} - \underline{z}_{26})/(\bar{z}_{26} - \underline{z}_{26}) \end{bmatrix}$$

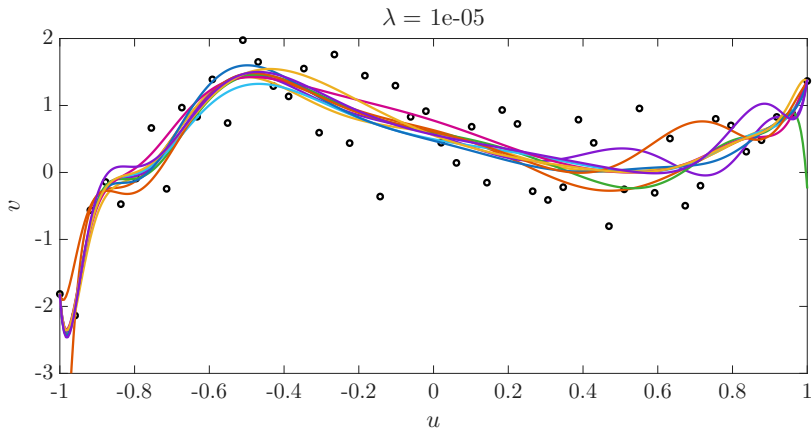
- standardize targets: $y^{(i)} = (v^{(i)} - \mu)/\sigma$

Unregularized model overfits training data

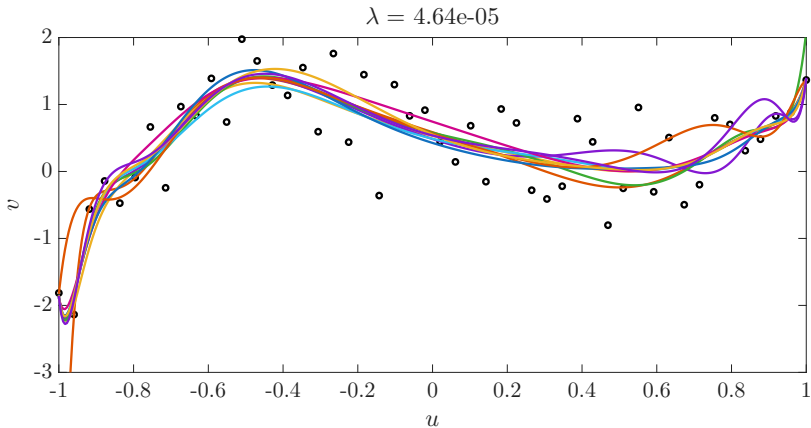


mean losses in fold 1: 0.2 training, 183,850 validation

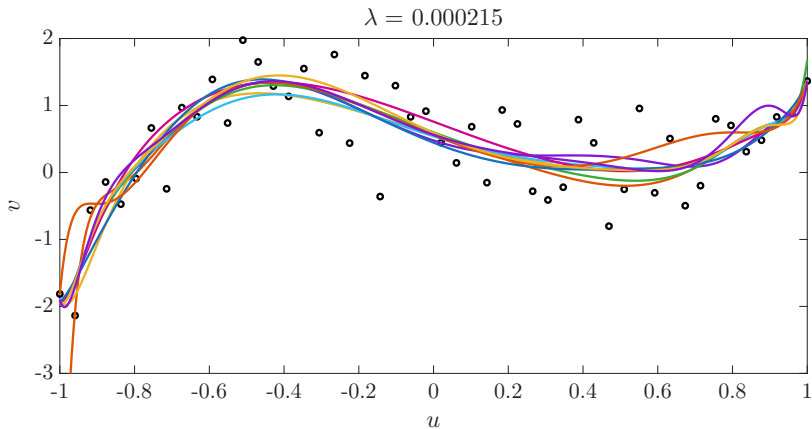
Lasso-regularized models



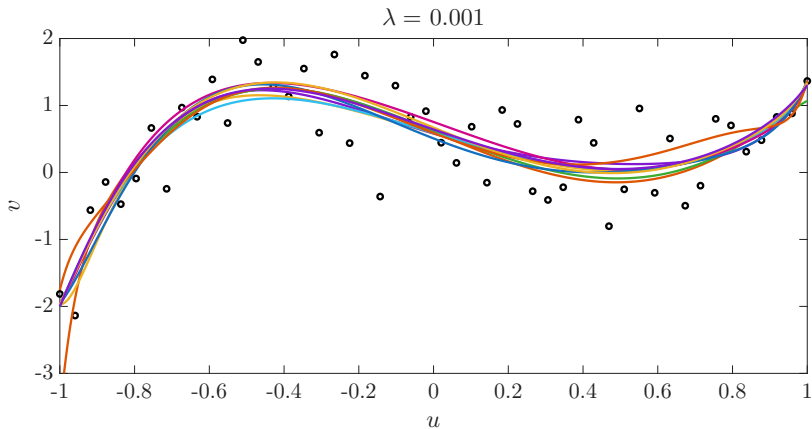
Lasso-regularized models



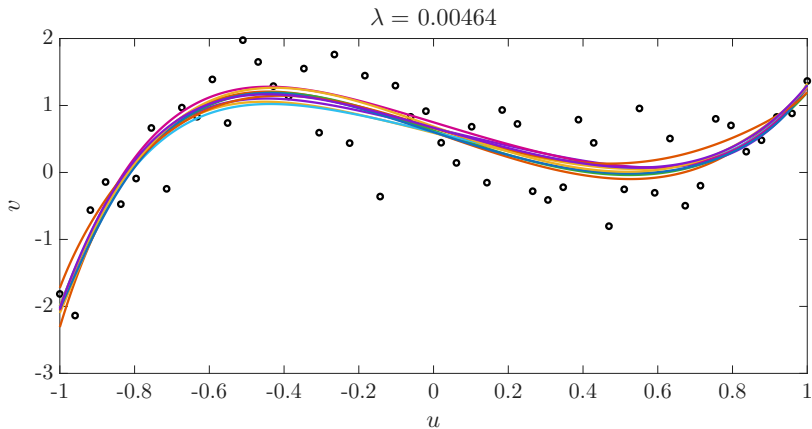
Lasso-regularized models



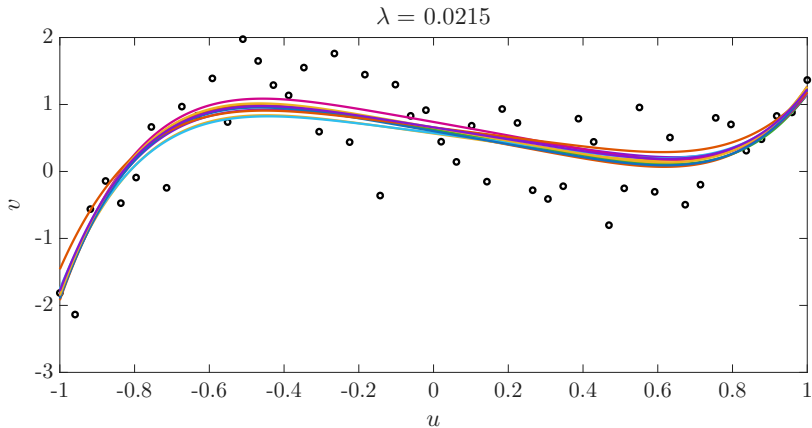
Lasso-regularized models



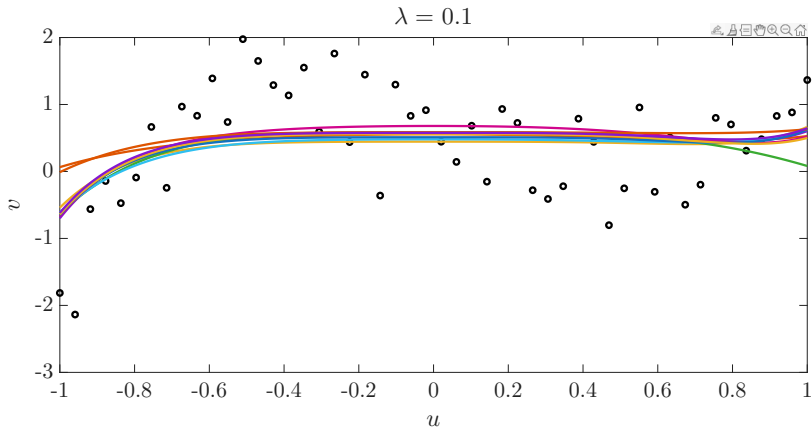
Lasso-regularized models



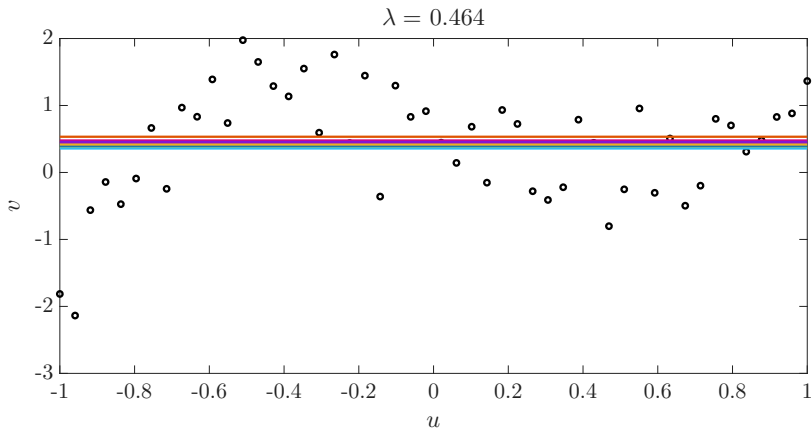
Lasso-regularized models



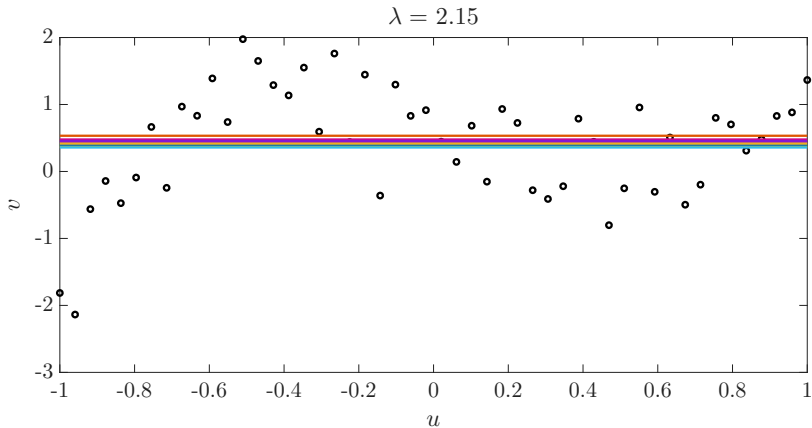
Lasso-regularized models



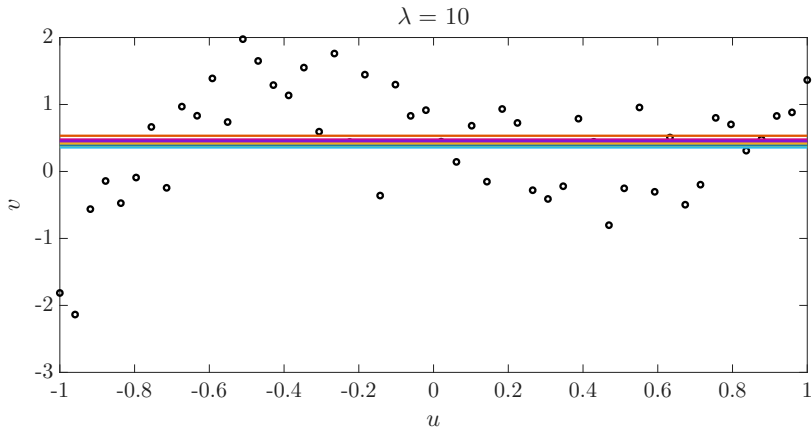
Lasso-regularized models



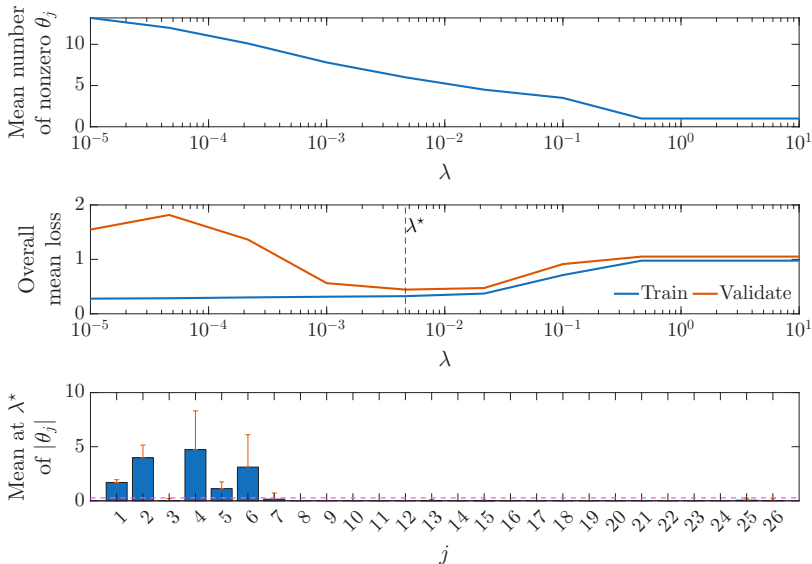
Lasso-regularized models



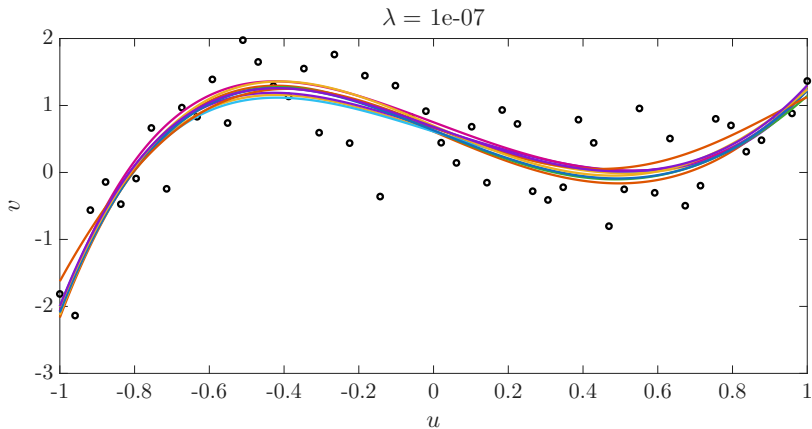
Lasso-regularized models



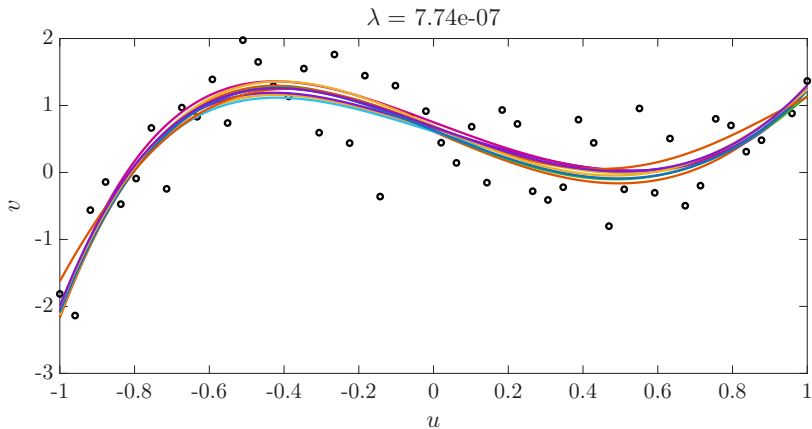
Feature selection via lasso regularization



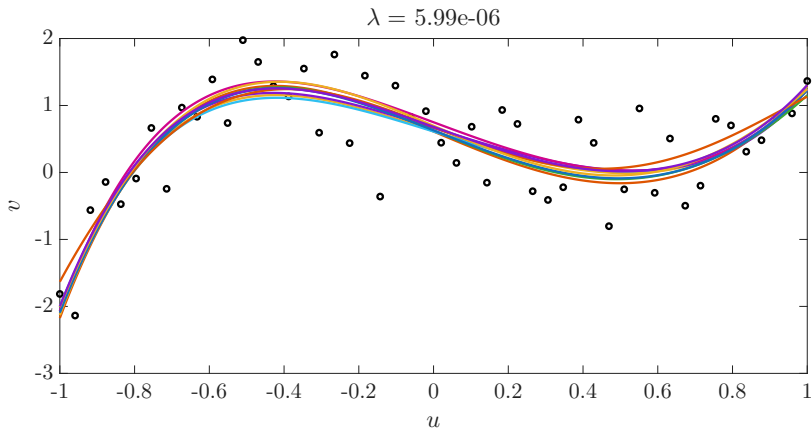
Ridge-regularized models with selected features



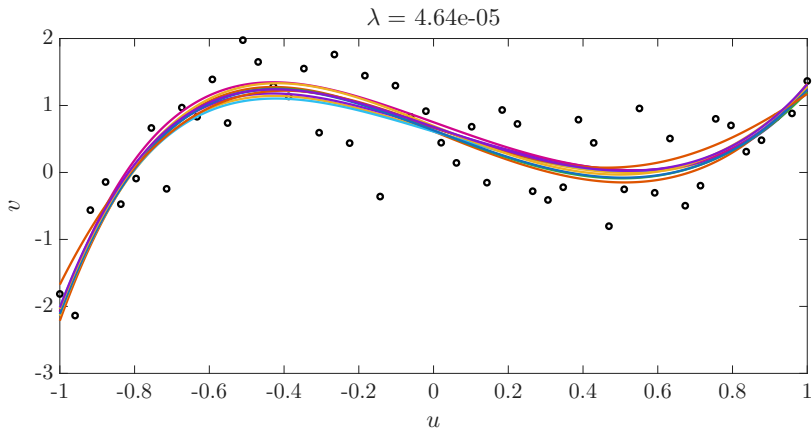
Ridge-regularized models with selected features



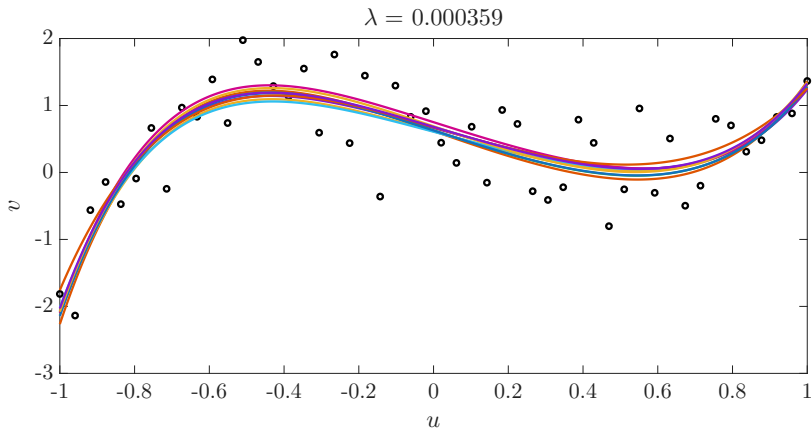
Ridge-regularized models with selected features



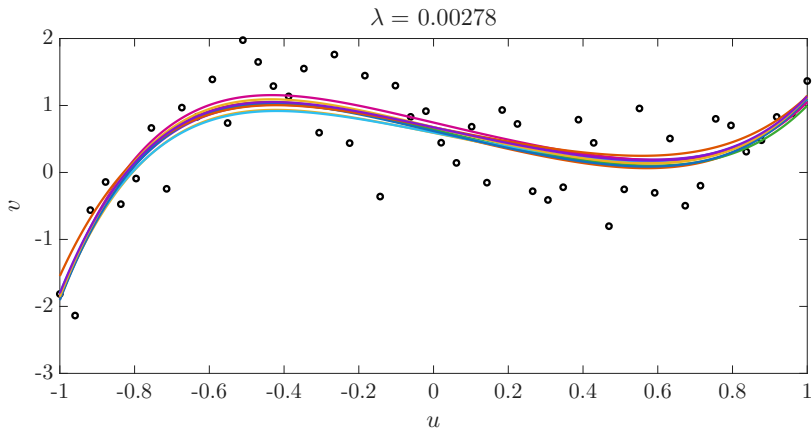
Ridge-regularized models with selected features



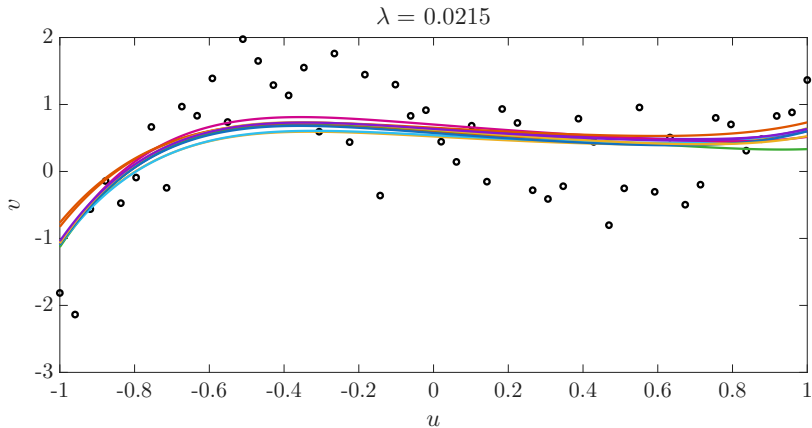
Ridge-regularized models with selected features



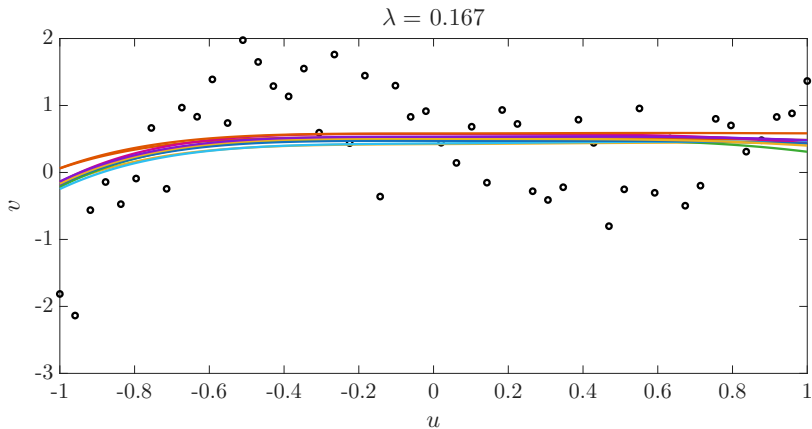
Ridge-regularized models with selected features



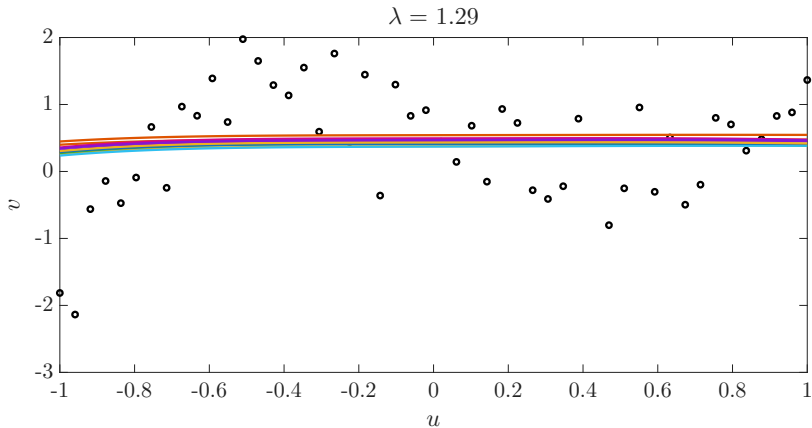
Ridge-regularized models with selected features



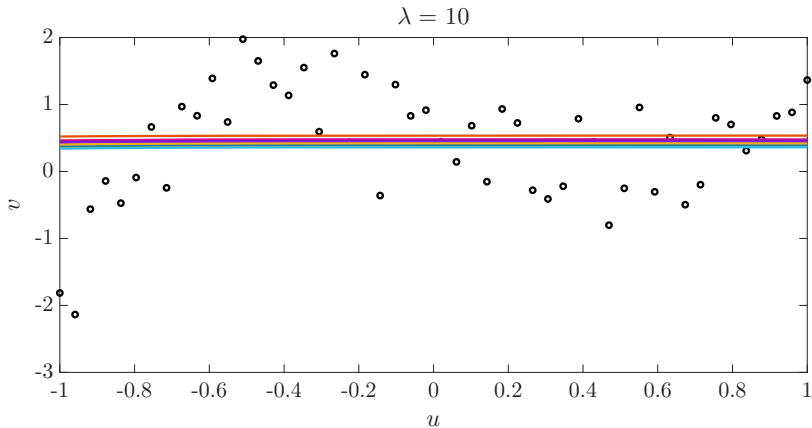
Ridge-regularized models with selected features



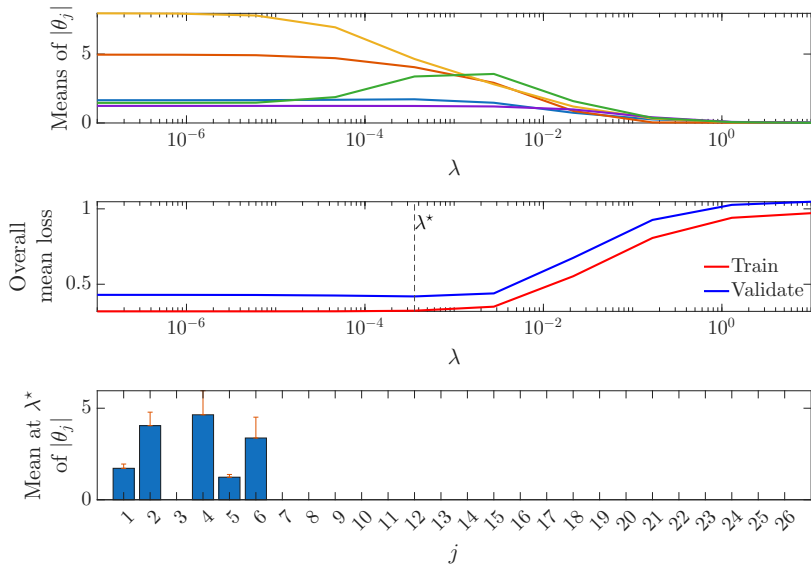
Ridge-regularized models with selected features



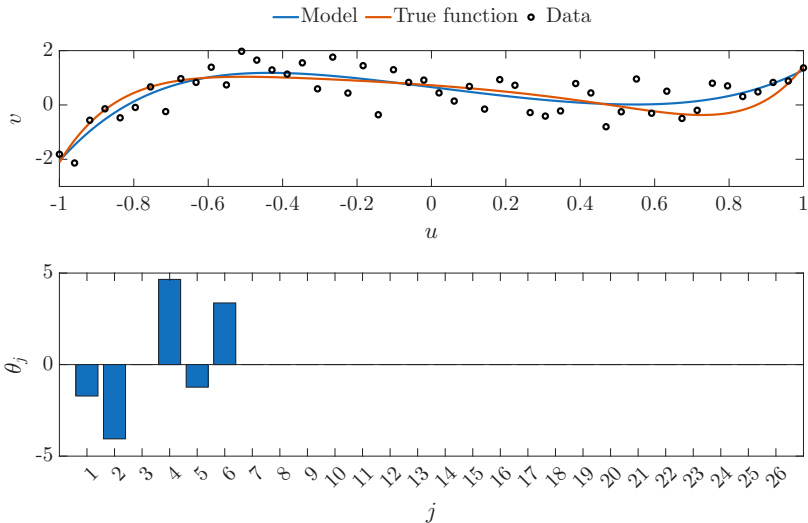
Ridge-regularized models with selected features



Hyperparameter tuning via ridge regularization



Final model



Summary

- 'linear models' are linear in θ and x , can be highly nonlinear in u
- general approach works for many problems
 1. embed raw inputs into a large number of candidate features
 2. select features via lasso regularization and K -folds validation
 3. 'polish' model with selected features and ridge regularization
- step 3 is fast (exact formula for square loss and ridge regularization)
- step 2 is slower (lasso regularization requires numerical optimization)
- approach also works for nonlinear models, but usually much slower

Outline

Regression overview

Example: Polynomial interpolation

Example: Thermal resistance from thermostat data

Problem

- estimate thermal resistance R in 1R1C building model,

$$C \frac{dT(t)}{dt} = \frac{T_{\text{out}}(t) - T(t)}{R} + q(t) + w(t)$$

- available measurements (5-minute time step, 6-week duration):
 - ◇ indoor temperature $T(t)$
 - ◇ outdoor temperature $T_{\text{out}}(t)$
 - ◇ heater thermal power $q(t)$

Prior information

- time-average of 1R1C model with $T(0) \approx T(\tau)$:

$$\frac{C}{\tau} \int_0^\tau \frac{dT(t)}{dt} dt = \frac{1}{\tau} \int_0^\tau \left[\frac{T_{\text{out}}(t) - T(t)}{R} + q(t) + w(t) \right] dt$$
$$\frac{C(T(\tau) - T(0))}{\tau} = \frac{\bar{T}_{\text{out}} - \bar{T}}{R} + \bar{q} + \bar{w}$$
$$\implies \bar{q} \approx \frac{\bar{T} - \bar{T}_{\text{out}}}{R} - \bar{w}$$

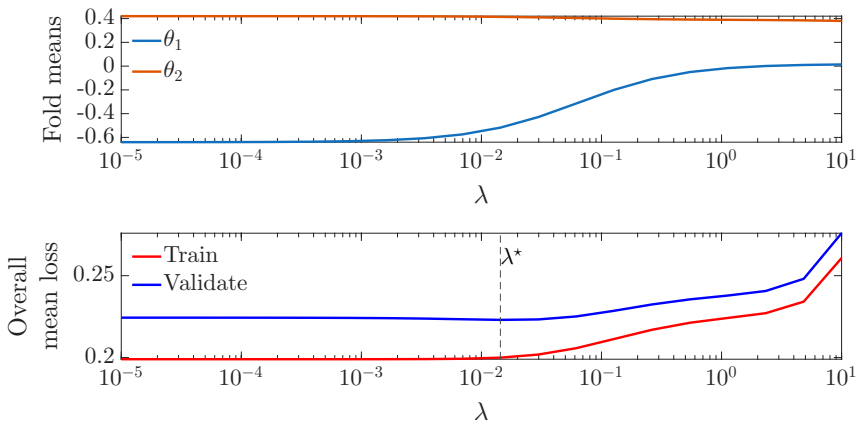
where $\bar{q} = \frac{1}{\tau} \int_0^\tau q(t) dt$ and so on for \bar{T} , \bar{T}_{out} , \bar{w}

- suggests linear model $y \approx \theta_1 x_1 + \theta_2 x_2$ with
 - target $y = \bar{q}$
 - features $x_1 = 1$, $x_2 = \bar{T} - \bar{T}_{\text{out}}$
 - parameters $\theta_1 = -\bar{w}$, $\theta_2 = 1/R$
- to avoid solar effects, embed raw data into nightly averages

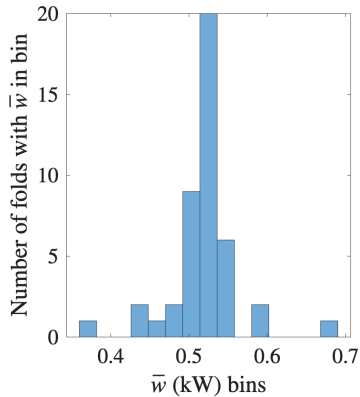
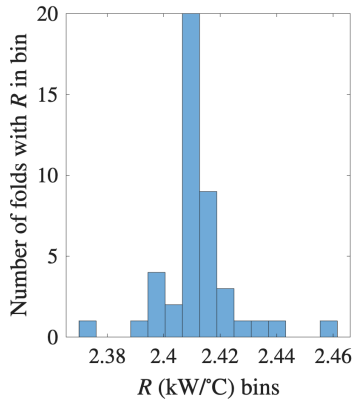
Solution approach

- linear model, square loss
- ridge regularization (including on constant feature)
- K -folds validation with $K = n = 44$
- only 2 features, so no need for a feature selection step

Hyperparameter tuning



Histograms of parameter estimates at λ^*



Final model, retrained on all (training + validation) data

