Machine Learning for Robotics
Intelligent Systems Series

Georg Martius
Slides adapted from Christoph Lampert, IST Austria

MPI for Intelligent Systems, Tübingen, Germany

April 20, 2017

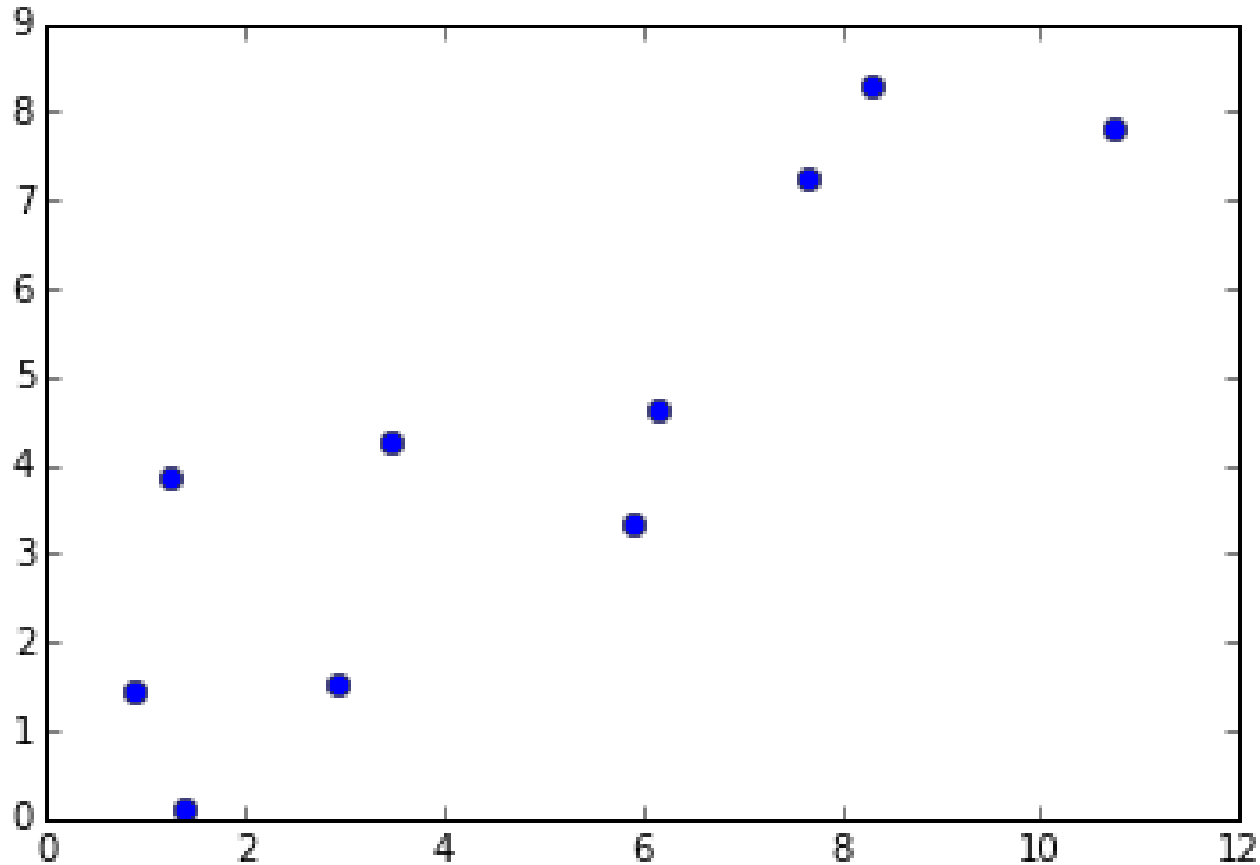
Regression

- Linear Regression
- Regularization
- Model Evaluation and Model Selection
- Nonlinear Regression
- Robust Regression

Given this data:

value	5.88	8.28	2.91	0.87	10.72	6.16	7.64	3.46	1.23	1.36
outcome	3.35	8.30	1.52	1.43	7.81	4.64	7.27	4.26	3.85	0.10

How to fit a model that can predict future outcomes?



Looks more or less straight...

Data:

x_i	5.88	8.28	2.91	0.87	10.72	6.16	7.64	3.46	1.23	1.36
y_i	3.35	8.30	1.52	1.43	7.81	4.64	7.27	4.26	3.85	0.10

Let's fit a linear model: $f(x) = ax$ for unknown $a \in \mathbb{R}$.

How to choose a? Least squares criterion: $\min_{a \in \mathbb{R}} \sum_{i=1}^{n} (ax_i - y_i)^2$

To find minimum, compute derivative:

$$\frac{d}{da} \sum_{i=1}^{n} (ax_i - y_i)^2 = 2 \sum_i x_i (ax_i - y_i) = 2a \sum_i (x_i)^2 - 2 \sum_i x_i y_i$$

Set derivative to 0 and solve for a:

$$a = \frac{\sum_i x_i y_i}{\sum_i (x_i)^2} = \frac{281.56}{339.06} = 0.83 \qquad \rightarrow \qquad f(x) = 0.83x$$

Data:

x_i	5.88	8.28	2.91	0.87	10.72	6.16	7.64	3.46	1.23	1.36
y_i	3.35	8.30	1.52	1.43	7.81	4.64	7.27	4.26	3.85	0.10
y_i/x_i	0.57	1.00	0.52	1.64	0.73	0.75	0.95	1.23	3.12	0.07

What else could we have done?

Since we want $f(x) \approx y$ for $f(x) = ax$. Since $a = \frac{f(x)}{x}$, how about

$$\min_a \sum_{i=1}^{n} \left(a - \frac{y_i}{x_i}\right)^2 \qquad \rightarrow \quad a = \frac{1}{n} \sum_{i=1}^{n} \frac{y_i}{x_i} = 1.06$$

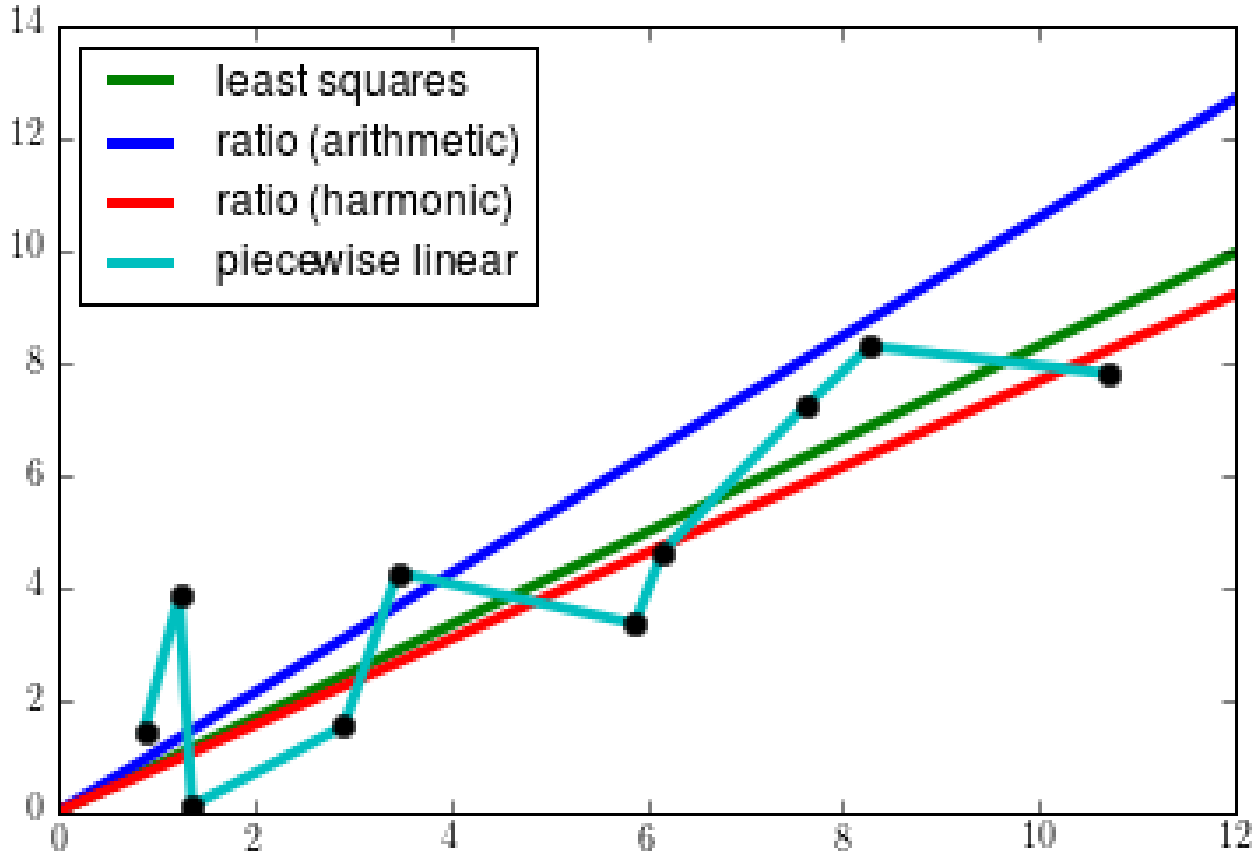
Arithmetic mean of ratios? Maybe rather geometric mean:

$$a = \sqrt[n]{\prod_i \frac{y_i}{x_i}} \quad = 0.77$$

Something completely different: piecewise linear?

Data:

x_i	5.88	8.28	2.91	0.87	10.72	6.16	7.64	3.46	1.23	1.36
y_i	3.35	8.30	1.52	1.43	7.81	4.64	7.27	4.26	3.85	0.10



Visual inspection: least squares (green) might be best...

Real data is large and high-dimensional: `Boston housing` dataset

$x \in \mathbb{R}^{13}$													y
0.01	18.00	2.31	0.00	0.54	6.58	65.20	4.09	1.00	296.00	15.30	396.90	4.98	24.00
0.03	0.00	7.07	0.00	0.47	6.42	78.90	4.97	2.00	242.00	17.80	396.90	9.14	21.60
0.03	0.00	7.07	0.00	0.47	7.18	61.10	4.97	2.00	242.00	17.80	392.83	4.03	34.70
0.03	0.00	2.18	0.00	0.46	7.00	45.80	6.06	3.00	222.00	18.70	394.63	2.94	33.40
0.07	0.00	2.18	0.00	0.46	7.15	54.20	6.06	3.00	222.00	18.70	396.90	5.33	36.20
0.03	0.00	2.18	0.00	0.46	6.43	58.70	6.06	3.00	222.00	18.70	394.12	5.21	28.70
0.09	12.50	7.87	0.00	0.52	6.01	66.60	5.56	5.00	311.00	15.20	395.60	12.43	22.90
0.14	12.50	7.87	0.00	0.52	6.17	96.10	5.95	5.00	311.00	15.20	396.90	19.15	27.10
0.21	12.50	7.87	0.00	0.52	5.63	100.00	6.08	5.00	311.00	15.20	386.63	29.93	16.50
0.17	12.50	7.87	0.00	0.52	6.00	85.90	6.59	5.00	311.00	15.20	386.71	17.10	18.90
0.22	12.50	7.87	0.00	0.52	6.38	94.30	6.35	5.00	311.00	15.20	392.52	20.45	15.00
0.12	12.50	7.87	0.00	0.52	6.01	82.90	6.23	5.00	311.00	15.20	396.90	13.27	18.90
0.09	12.50	7.87	0.00	0.52	5.89	39.00	5.45	5.00	311.00	15.20	390.50	15.71	21.70
0.63	0.00	8.14	0.00	0.54	5.95	61.80	4.71	4.00	307.00	21.00	396.90	8.26	20.40
0.64	0.00	8.14	0.00	0.54	6.10	84.50	4.46	4.00	307.00	21.00	380.02	10.26	18.20
0.63	0.00	8.14	0.00	0.54	5.83	56.50	4.50	4.00	307.00	21.00	395.62	8.47	19.90
1.05	0.00	8.14	0.00	0.54	5.93	29.30	4.50	4.00	307.00	21.00	386.85	6.58	23.10
0.78	0.00	8.14	0.00	0.54	5.99	81.70	4.26	4.00	307.00	21.00	386.75	14.67	17.50
0.80	0.00	8.14	0.00	0.54	5.46	36.60	3.80	4.00	307.00	21.00	288.99	11.69	20.20
0.73	0.00	8.14	0.00	0.54	5.73	69.50	3.80	4.00	307.00	21.00	390.95	11.28	18.20
1.25	0.00	8.14	0.00	0.54	5.57	98.10	3.80	4.00	307.00	21.00	376.57	21.02	13.60
0.85	0.00	8.14	0.00	0.54	5.96	89.20	4.01	4.00	307.00	21.00	392.53	13.83	19.60
1.23	0.00	8.14	0.00	0.54	6.14	91.70	3.98	4.00	307.00	21.00	396.90	18.72	15.20
0.99	0.00	8.14	0.00	0.54	5.81	100.00	4.10	4.00	307.00	21.00	394.54	19.88	14.50
0.75	0.00	8.14	0.00	0.54	5.92	94.10	4.40	4.00	307.00	21.00	394.33	16.30	15.60
0.84	0.00	8.14	0.00	0.54	5.60	85.70	4.45	4.00	307.00	21.00	303.42	16.51	13.90
0.67	0.00	8.14	0.00	0.54	5.81	90.30	4.68	4.00	307.00	21.00	376.88	14.81	16.60
0.96	0.00	8.14	0.00	0.54	6.05	88.80	4.45	4.00	307.00	21.00	306.38	17.28	14.80
0.77	0.00	8.14	0.00	0.54	6.50	94.40	4.45	4.00	307.00	21.00	387.94	12.80	18.40
1.00	0.00	8.14	0.00	0.54	6.67	87.30	4.24	4.00	307.00	21.00	380.23	11.98	21.00
1.13	0.00	8.14	0.00	0.54	5.71	94.10	4.23	4.00	307.00	21.00	360.17	22.60	12.70
1.35	0.00	8.14	0.00	0.54	6.07	100.00	4.17	4.00	307.00	21.00	376.73	13.04	14.50
1.39	0.00	8.14	0.00	0.54	5.95	82.00	3.99	4.00	307.00	21.00	232.60	27.71	13.20
1.15	0.00	8.14	0.00	0.54	5.70	95.00	3.79	4.00	307.00	21.00	358.77	18.35	13.10
1.61	0.00	8.14	0.00	0.54	6.10	96.90	3.76	4.00	307.00	21.00	248.31	20.34	13.50
							⋮						
0.08	0.00	5.96	0.00	0.50	5.85	41.50	3.93	5.00	279.00	19.20	396.90	8.77	11.90

Given $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i = (x_i^1, \ldots, x_i^d) \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$.

Linear model: $f(x) = w^\top x$ for $w^\top x = \sum_{j=1}^d w^j x^j$

$$\text{Least squares:} \quad \min_{w \in \mathbb{R}^d} \mathcal{L}(w) \quad \text{with } \mathcal{L}(w) = \sum_i (w^\top x_i - y_i)^2$$

$$\nabla_w \mathcal{L}(w) = 2 \sum_i x_i (x_i{}^\top w - y_i) = 2 \sum_i x_i x_i{}^\top w - 2 \sum_i x_i y_i$$

Setting the gradient to zero:

$$\underbrace{\sum_i x_i x_i{}^\top}_{\in \mathbb{R}^{d \times d}} w = \sum_i x_i y_i$$

We can solve for w if $\sum_i x_i x_i$ is full rank (at least: $n \geq d$),

$$w = \Big(\sum_i x_i x_i{}^\top \Big)^{-1} \sum_i x_i y_i$$

In matrix notation: $X = \big(x_1 | x_2 | \dots | x_n\big) \in \mathbb{R}^{d \times n}$, $Y \in \mathbb{R}^n$, $w \in \mathbb{R}^d$.

Least squares: $\min_{w \in \mathbb{R}^d} \mathcal{L}(w)$ with $\mathcal{L}(w) = \|X^\top w - Y\|^2$

$$\nabla_w \mathcal{L}(w) = \nabla_w \big(w^\top X X^\top w - Y^\top X^\top w - w^\top X Y + Y^\top Y\big) = 2XX^\top w - 2XY$$

Setting the gradient to zero:

$$XX^\top w = XY$$

If $XX^\top$ is full rank, we can solve for w:

$$w = \big(XX^\top\big)^{-1} XY \qquad \text{or} \qquad \begin{array}{l}\texttt{lhs = dot(X, X.T)} \\ \texttt{rhs = dot(X, Y)} \\ \texttt{w = solve(lhs, rhs)}\end{array}$$

Very useful to memorize: matrix calculus

$$\nabla_x a = 0 \qquad \nabla_x\ c^\top x = c \qquad \nabla_x\ x^\top c = c \qquad \nabla_x\ x^\top A x = (A^\top + A)x$$

`Boston housing` dataset

$x \in \mathbb{R}^{13}$													y
0.01	18.00	2.31	0.00	0.54	6.58	65.20	4.09	1.00	296.00	15.30	396.90	4.98	24.00
0.03	0.00	7.07	0.00	0.47	6.42	78.90	4.97	2.00	242.00	17.80	396.90	9.14	21.60
0.03	0.00	7.07	0.00	0.47	7.18	61.10	4.97	2.00	242.00	17.80	392.83	4.03	34.70
0.03	0.00	2.18	0.00	0.46	7.00	45.80	6.06	3.00	222.00	18.70	394.63	2.94	33.40
0.07	0.00	2.18	0.00	0.46	7.15	54.20	6.06	3.00	222.00	18.70	396.90	5.33	36.20
0.03	0.00	2.18	0.00	0.46	6.43	58.70	6.06	3.00	222.00	18.70	394.12	5.21	28.70
0.09	12.50	7.87	0.00	0.52	6.01	66.60	5.56	5.00	311.00	15.20	395.60	12.43	22.90
0.14	12.50	7.87	0.00	0.52	6.17	96.10	5.95	5.00	311.00	15.20	396.90	19.15	27.10
0.21	12.50	7.87	0.00	0.52	5.63	100.00	6.08	5.00	311.00	15.20	386.63	29.93	16.50
0.17	12.50	7.87	0.00	0.52	6.00	85.90	6.59	5.00	311.00	15.20	386.71	17.10	18.90
0.22	12.50	7.87	0.00	0.52	6.38	94.30	6.35	5.00	311.00	15.20	392.52	20.45	15.00
0.12	12.50	7.87	0.00	0.52	6.01	82.90	6.23	5.00	311.00	15.20	396.90	13.27	18.90
0.09	12.50	7.87	0.00	0.52	5.89	39.00	5.45	5.00	311.00	15.20	390.50	15.71	21.70
0.63	0.00	8.14	0.00	0.54	5.95	61.80	4.71	4.00	307.00	21.00	396.90	8.26	20.40
0.64	0.00	8.14	0.00	0.54	6.10	84.50	4.46	4.00	307.00	21.00	380.02	10.26	18.20
0.63	0.00	8.14	0.00	0.54	5.83	56.50	4.50	4.00	307.00	21.00	395.62	8.47	19.90
							⋮						
0.08	0.00	5.96	0.00	0.50	5.85	41.50	3.93	5.00	279.00	19.20	396.90	8.77	11.90

Learned linear least-square model:

$$f(x) = w^\top x$$

for

$$w = (0.11, 0.02, 0.08, -0.96, 19.2, -2.9, 0.09, 0.63, -0.07, 0, 0.64, 0)^\top$$

Is this a good model? What else could we do?

Linear model with bias term (= "intercept"):

$$f(x) = w^\top x + b \text{ for } w \in \mathbb{R}^d, b \in \mathbb{R}$$

Least squares: $\min_{w\in\mathbb{R}^d, b\in\mathbb{R}} \mathcal{L}(w,b)$ with $\mathcal{L}(w,b) = \sum_i (w^\top x_i + b - y_i)^2$

$$0 = \nabla_b \mathcal{L}(w,b) = 2\sum_i ({x_i}^\top w + b - y_i) \quad \rightarrow \quad b^{\text{opt}} = \bar{y} - \bar{x}^\top w$$

for $\bar{x} = \frac{1}{n}\sum_i x_i$ and $\bar{y} = \frac{1}{n}\sum_i y_i$.

$$\bar{\mathcal{L}}(w) = \bar{\mathcal{L}}(w, b^{\text{opt}}) = \sum_i \big(\ (x_i - \bar{x})^\top w - (y_i - \bar{y}) \ \big)^2$$

$$0 = \nabla_w \bar{\mathcal{L}}(w) = 2\sum_i \big(\ (x_i - \bar{x})(x_i - \bar{x})^\top w - (y_i - \bar{y}) \ \big)$$

Solve for w (if possible):

$$w = \big(\underbrace{\sum_i (x_i - \bar{x})(x_i - \bar{x})^\top}_{=n\text{Cov}(x_1,\ldots,x_n)} \big)^{-1} \underbrace{\sum_i (x_i - \bar{x})(y_i - \bar{y})}_{=n\text{Cov}(x_1,\ldots,x_n;y_1,,\ldots,y_n)}$$

$$= \text{Cov}(X)^{-1}\text{Cov}(X,Y)$$

Feature augmentation

Observation: centered data

If the training data is centered, i.e. $\frac{1}{n}\sum_i x_i = 0$, $\frac{1}{n}\sum_i y_i = 0$, we don't need a bias term $\rightarrow$ we can reuse code for linear regression without bias.

Alternative trick: feature augmentation

Adding a constant feature allows us to avoid models with explicit bias term:

- instead of $x = (x^1, \ldots, x^d) \in \mathbb{R}^d$, use $\tilde{x} = (x^1, \ldots, x^d, 1) \in \mathbb{R}^{d+1}$
- for any $\tilde{w} \in \mathbb{R}^{d+1}$, think $\tilde{w} = (w, b)$ with $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$

Linear model in $\mathbb{R}^{d+1}$:

$$f(\tilde{x}) = \tilde{w}^\top \tilde{x} = \sum_{i=1}^{d+1} \tilde{w}_i \tilde{x}_i = \sum_{i=1}^{d} \tilde{w}_i \tilde{x}_i + \tilde{w}_{d+1}\tilde{x}_{d+1} = w^\top x + b$$

Linear model with bias term in $\mathbb{R}^d$ $\equiv$ linear model with no bias term in $\mathbb{R}^{d+1}$

Boston housing dataset

$x \in \mathbb{R}^{13}$													y
0.01	18.00	2.31	0.00	0.54	6.58	65.20	4.09	1.00	296.00	15.30	396.90	4.98	24.00
0.03	0.00	7.07	0.00	0.47	6.42	78.90	4.97	2.00	242.00	17.80	396.90	9.14	21.60
0.03	0.00	7.07	0.00	0.47	7.18	61.10	4.97	2.00	242.00	17.80	392.83	4.03	34.70
0.03	0.00	2.18	0.00	0.46	7.00	45.80	6.06	3.00	222.00	18.70	394.63	2.94	33.40
0.07	0.00	2.18	0.00	0.46	7.15	54.20	6.06	3.00	222.00	18.70	396.90	5.33	36.20
0.03	0.00	2.18	0.00	0.46	6.43	58.70	6.06	3.00	222.00	18.70	394.12	5.21	28.70
0.09	12.50	7.87	0.00	0.52	6.01	66.60	5.56	5.00	311.00	15.20	395.60	12.43	22.90
0.14	12.50	7.87	0.00	0.52	6.17	96.10	5.95	5.00	311.00	15.20	396.90	19.15	27.10
0.21	12.50	7.87	0.00	0.52	5.63	100.00	6.08	5.00	311.00	15.20	386.63	29.93	16.50
0.17	12.50	7.87	0.00	0.52	6.00	85.90	6.59	5.00	311.00	15.20	386.71	17.10	18.90
0.22	12.50	7.87	0.00	0.52	6.38	94.30	6.35	5.00	311.00	15.20	392.52	20.45	15.00
0.12	12.50	7.87	0.00	0.52	6.01	82.90	6.23	5.00	311.00	15.20	396.90	13.27	18.90
0.09	12.50	7.87	0.00	0.52	5.89	39.00	5.45	5.00	311.00	15.20	390.50	15.71	21.70
0.63	0.00	8.14	0.00	0.54	5.95	61.80	4.71	4.00	307.00	21.00	396.90	8.26	20.40
0.64	0.00	8.14	0.00	0.54	6.10	84.50	4.46	4.00	307.00	21.00	380.02	10.26	18.20
0.63	0.00	8.14	0.00	0.54	5.83	56.50	4.50	4.00	307.00	21.00	395.62	8.47	19.90
							⋮						
0.08	0.00	5.96	0.00	0.50	5.85	41.50	3.93	5.00	279.00	19.20	396.90	8.77	11.90

Learned linear least-square model: $f(x) = w^\top x + b$

for $w = (0.10, 0.01, 0.01, -1.02, 5.20, -4.41, 0.09, 0.15, 0.05, 0, -0.12, 0)$

$b = 30.15$

Is this model better than the one without bias term?

Understanding model quality

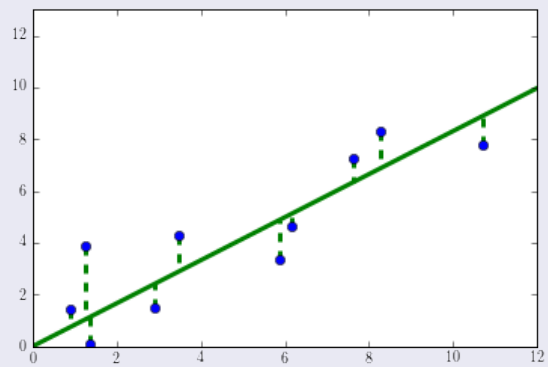
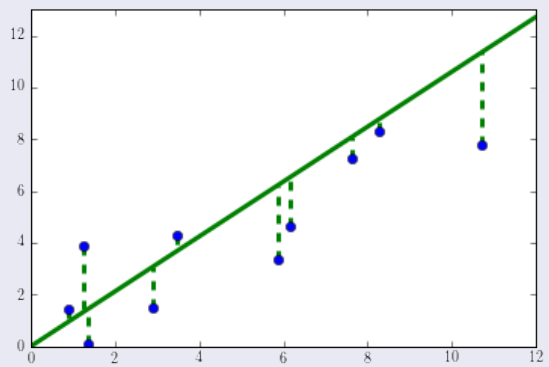
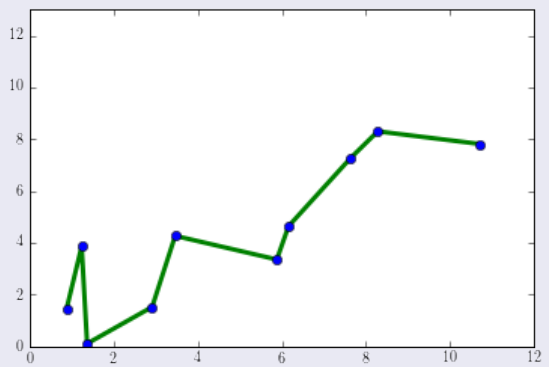
So you've trained a model. How good it is?

You've trained multiple models. Which one to choose?

Is the model 'good enough'? What would be the best possible model?

Wrong approach to evaluate predictive models: explained variance

Evaluate model error by checking how well it fits the training data.

Explained variance prefers complex models over simple ones (always!)

Right approach to model evaluation

The model is *not for you*, it's send out to be used by a *user*.

You: train the model on your data

input training data $\mathcal{D}_{trn}$
$f \leftarrow\!\shortmid$ some procedure using $\mathcal{D}_{trn}$
output predictive model $f : \mathcal{X} \to \mathbb{R}$

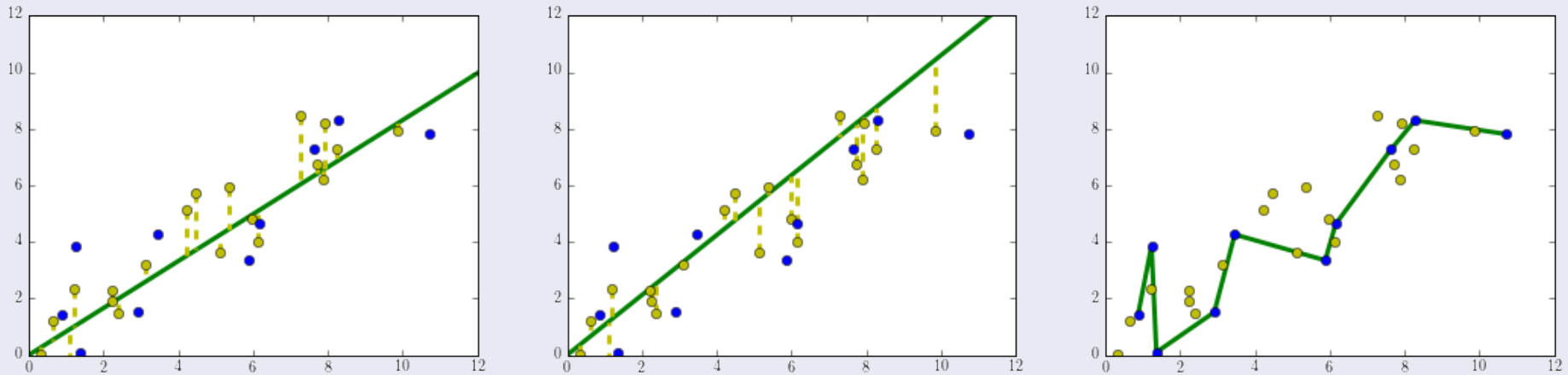
The user: make predictions on his/her own data

input trained predictive model $f : \mathcal{X} \to \mathbb{R}$
input new data $\mathcal{D}_{tst}$
use f to make predictions on $\mathcal{D}_{tst}$
output happy face or lawsuit

What matters isn't how good the model is on the training set, but on **new data**.

Evaluate predictive models: generalization performance

Evaluate model error on data that has not been used for training.



For trained model f and new data $\mathcal{D}_{tst} = \{(x'_1, y'_1), \ldots, (x'_m, y'_m)\}$,

$$E(f) = \frac{1}{m} \sum_{i=1}^{m} (f(x'_i) - y'_i)^2$$

Observation: A model can be perfect on training data (blue), but still not do great on new data (yellow).

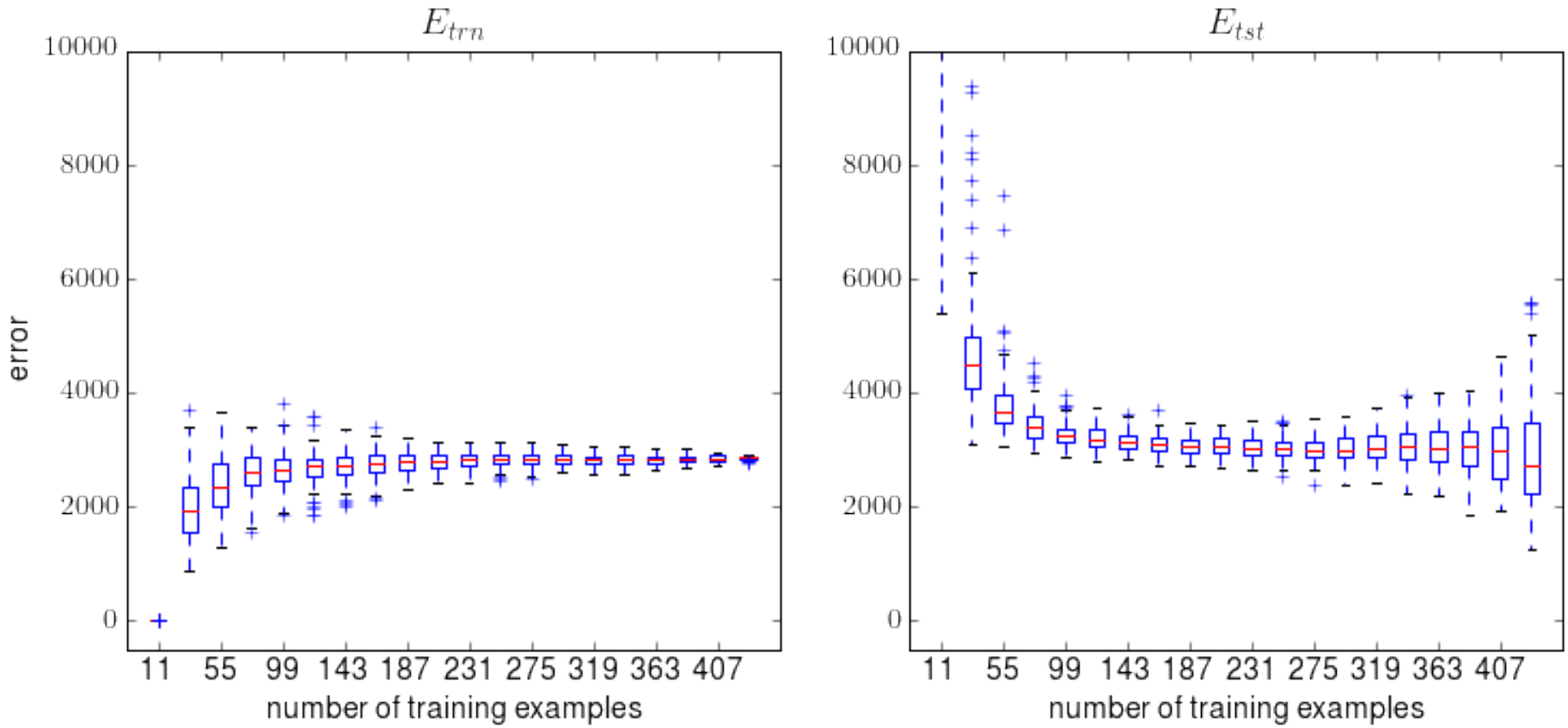
Model evaluation

In practice, we don't get "new" data. We'll have to use the available data both for training and for evaluation.

Model Training and Evaluation

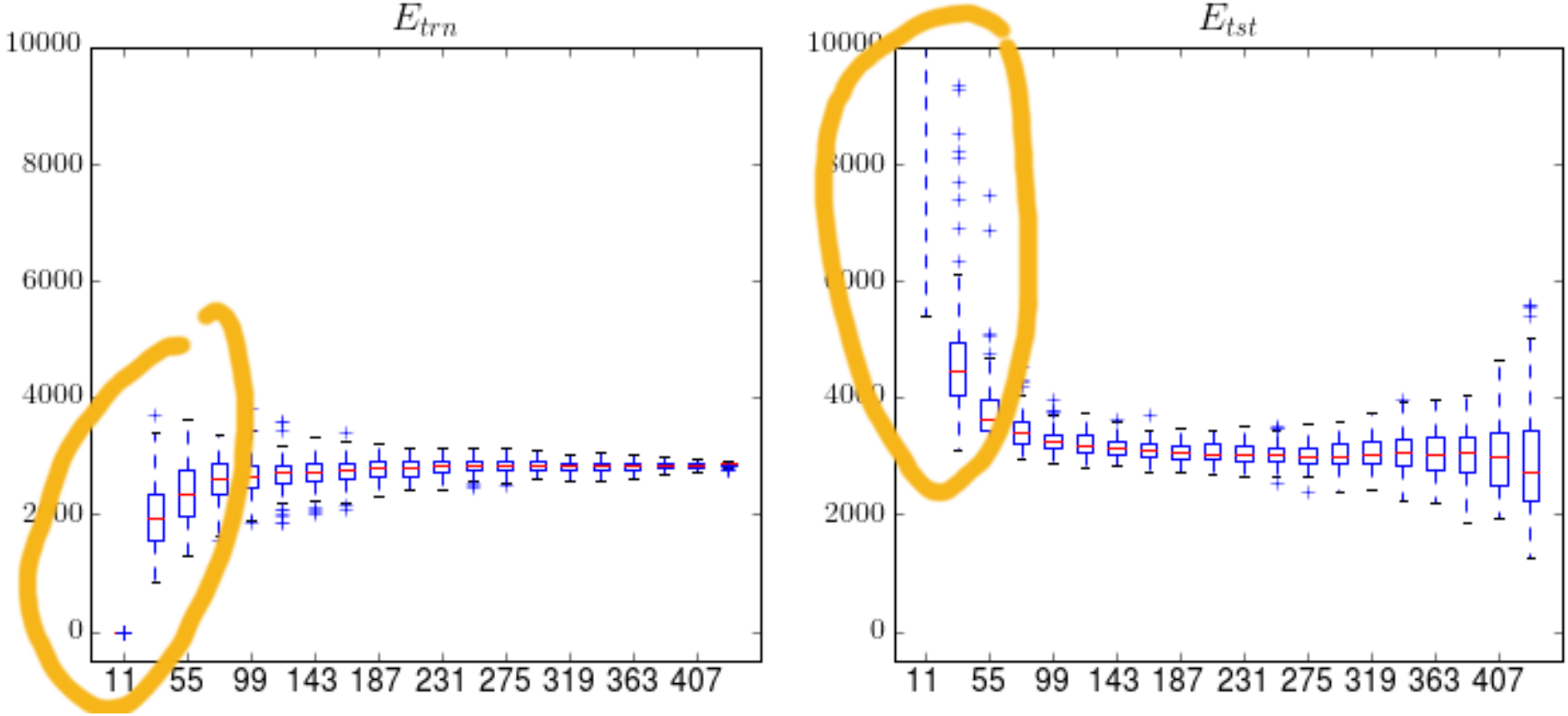
input data $\mathcal{D}$
input learning method A
  randomly split $\mathcal{D} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place // do not look at it!
  $f \leftarrow\!\shortmid A[\mathcal{D}_{trn}]$ // i.e. train model on training set
  $E(f) \leftarrow\!\shortmid$ performance of f on $\mathcal{D}_{tst}$
output trained model f, performance estimate $E(f)$

- Do not use $\mathcal{D}_{tst}$ for anything except the very last step.
- Do not look at $\mathcal{D}_{tst}$! Even if the learning algorithm doesn't see it, you looking at it can and will influence your model design or parameter selection (human overfitting).



Fact 1: Predictive models tend to get better when trained on more data.
Fact 2: A too small test set makes the quality estimate unreliable.

Guideline: $\mathcal{D}_{trn}$ should be as big as possible, but $\mathcal{D}_{tst}$ must be large enough to be convincing.



Fact 3: With very little training data, the training error is very small, but the test error is very large.

OVERFITTING!

The model has learned to reproduce idiosyncracies/noise. On new data, this causes a large error.

Observation: Least squares regression overfits if the number of example is not much bigger than the number of data dimensions.

How to avoid overfitting?

Feature Selection

Idea: reduce the dimensionality of the data by dropping some dimensions

Problems: 1) which ones? 2) simply throwing away data is rarely a good idea

Dimensionality Reduction

Idea: reduce data dimensionality differently, e.g. Principal Component Analysis.

Problems: 1) can destroy structure, 2) few dimensions might not be enough

Regularization

Idea: prevent the model from overfitting by making it **robust**

Problems: almost none. This is actually a good idea.

What do we mean by robustness?

Robustness of model parameters

How robust are the linear least squares model parameters

$$w = (XX^\top)^{-1}XY$$

against small changes in the training data X?

Robustness of predictions

How robust are the predictions

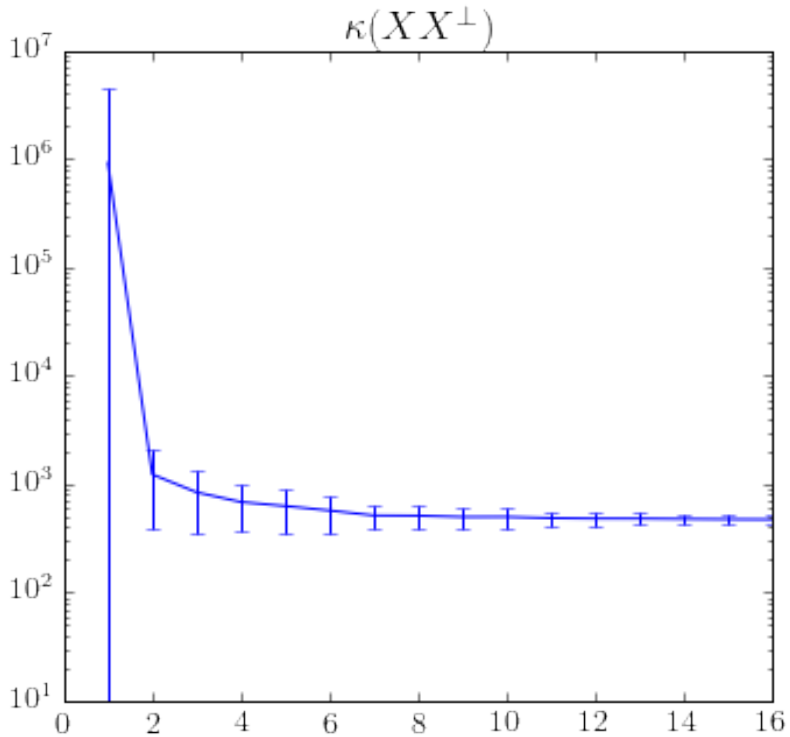
$$f(x) = w^\top x$$

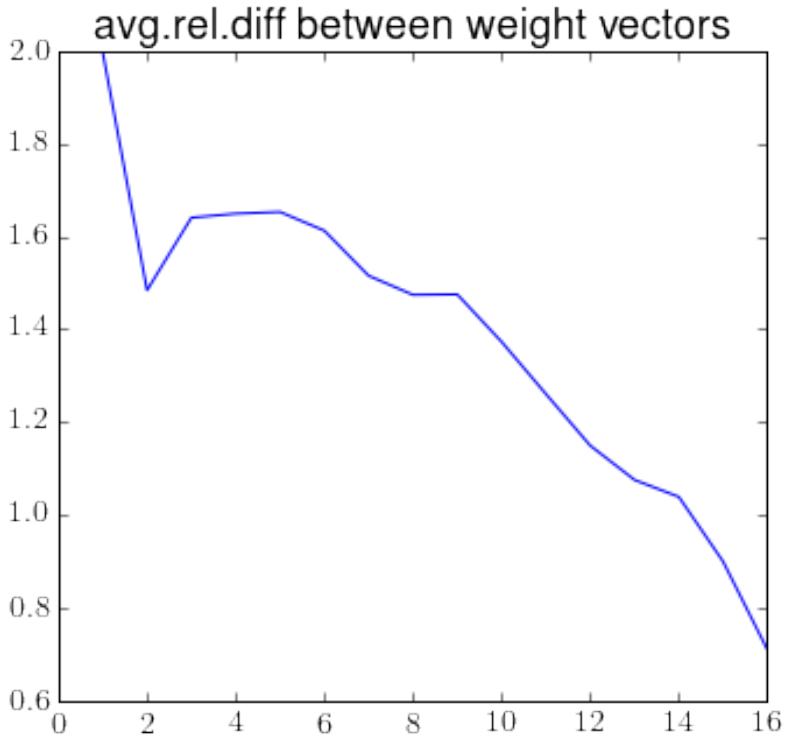
against small changes of x ?

Numerical Robustness

Robustness of the parameter vector $w = (XX^\top)^{-1}XY$ is determined by the **condition number** of the matrix $XX^\top$:

$$\kappa = \frac{\sigma_{\max}(XX^\top)}{\sigma_{\min}(XX^\top)}.$$





- for real data, κ is often large unless n is much larger than d
  $\rightarrow$ randomness in X can have large impact on w

Robustness of Predictions

For any linear model, the robustness of the predictions $f(x) = w^\top x$ is determined by the **norm** of the weight vector $\|w\|$:

$$\frac{f(x+\epsilon) - f(x)}{\|\epsilon\|} = \frac{\langle w, x+\epsilon\rangle - \langle w, x\rangle}{\|\epsilon\|} = \frac{\langle w, \epsilon\rangle}{\|\epsilon\|} \leq \frac{\|w\|\|\epsilon\|}{\|\epsilon\|} = \|w\|$$

Insight:

- if many different w work well on the training data, prefer the one with small $\|w\|$
- maybe even: allows for higher E_{trn} to avoid models with very large $\|w\|$

Regularization

So far:

- learn model parameters by minimizing error on training set

Now:

- take *robustness* into account as well when learning parameters

Regularized Least Squared Regression (= Ridge Regression)

For some $\lambda \geq 0$ (=regularization parameter), solve

$$\min_{w \in \mathbb{R}^d} \underbrace{\sum_i (w^\top x_i - y_i)^2}_{\text{training error}} + \lambda \underbrace{\|w\|^2}_{\text{"regularizer"}}$$

Observation:

- the bigger λ, the more emphasis we put on *robustness* versus *training error*
- $\lambda = 0 \quad \rightarrow$ no regularization, original least squares regression
- $\lambda \rightarrow \infty \quad \rightarrow$ training error ignored ($w \rightarrow 0$), but perfectly robust

Open question: what's the best value for λ?

Ridge Regression

Ridge regression is as easy to learn as least squares:

$$\min_{w \in \mathbb{R}^d} \; \mathcal{L}(w) + \lambda \Omega(w)$$

for $\mathcal{L}(w) = \sum_i (w^\top x_i - y_i)^2$ and $\Omega(w) = \|w\|^2$.

$$\nabla_w \Big[\mathcal{L}(w) + \lambda \Omega(w) \Big] = 2 \sum_i x_i x_i^\top w - 2 \sum_i x_i y_i + 2\lambda w$$

Set gradient to zero:

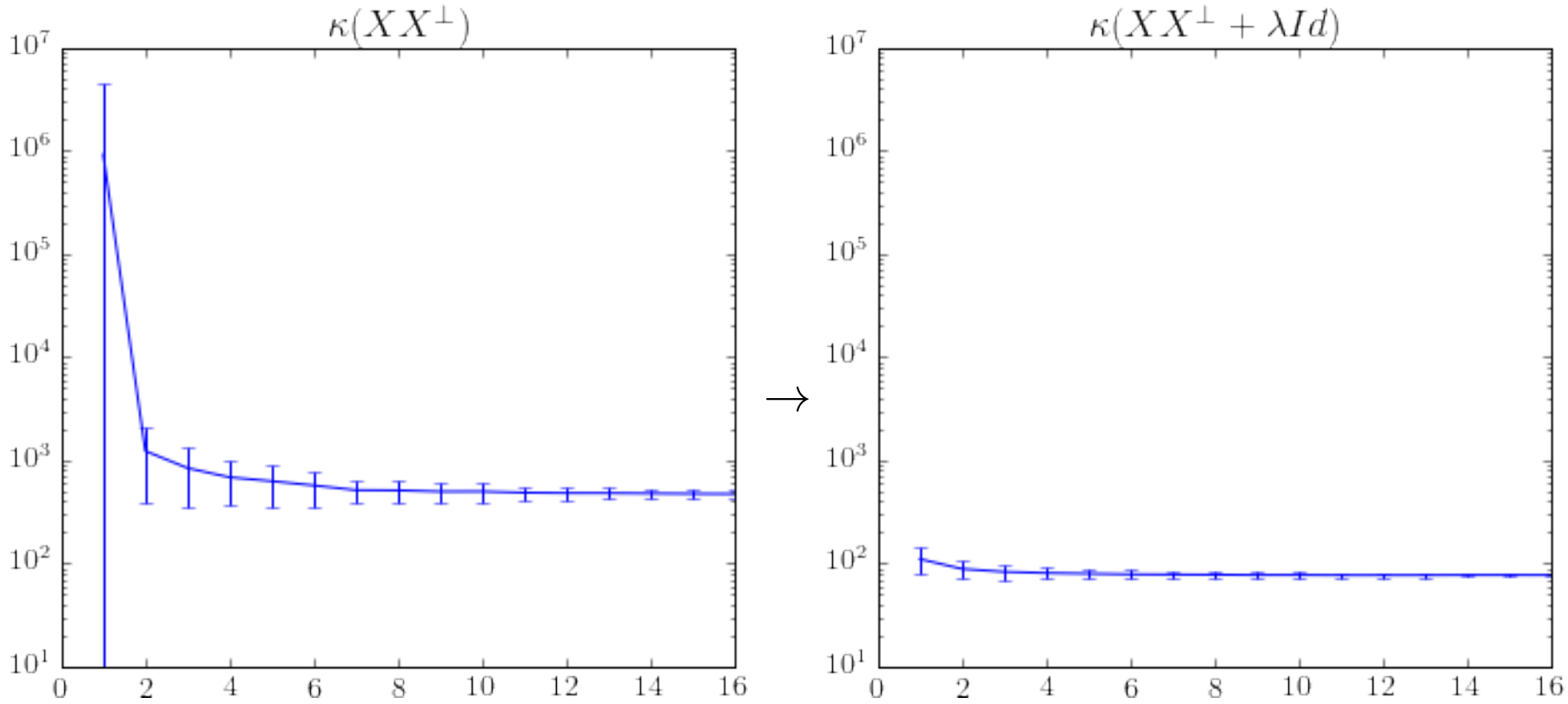
$$\underbrace{\sum_i x_i x_i^\top w + \lambda w}_{= (\sum_i x_i x_i^\top \, + \, \lambda \, \mathrm{Id}_{n \times n}) w} = \sum_i x_i y_i$$

For $\lambda > 0$, we can *always* solve for w (regardless if $n \geq d$),

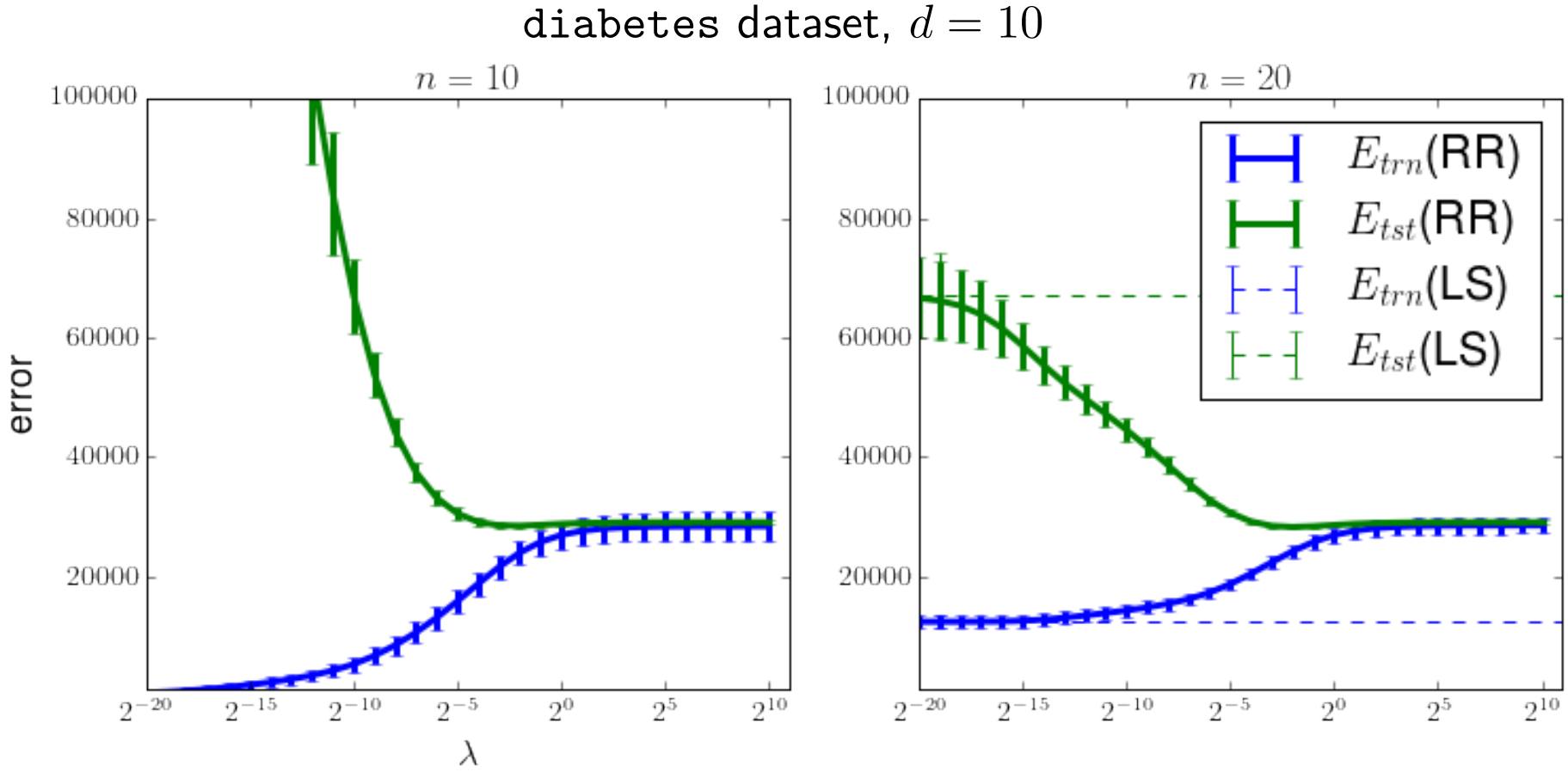
$$w = \Big(\sum_i x_i x_i^\top + \lambda \, \mathrm{Id} \Big)^{-1} \sum_i x_i y_i$$

Ridge Regression

Already small regularization strongly increases robustness (here: $\lambda = 0.0001n$)



Training and test error for different regularization constants:



Model Selection

Question:

- How to select one model from many possible ones.
- How to set free parameters of a model (e.g. regularization)?

Training and Choosing between Multiple Models (suboptimal, don't use)

input data $\mathcal{D}$, set of method $\mathcal{A} = \{A_1, \ldots, A_K\}$
  randomly split $\mathcal{D} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  for all possible procedures $A_i \in \mathcal{A}$ **do**
    $f_i \leftarrow\!\shortmid A_i[\mathcal{D}_{trn}]$
    $E(f_i) \leftarrow\!\shortmid$ performance of f_i on $\mathcal{D}_{tst}$
  end for
output $f \leftarrow\!\shortmid f_i$ for $i = \mathbf{argmin}_i\, E(f_i)$ // pick best performing f_i

Problem: How good is the selected model? We don't know!
$\mathcal{D}_{tst}$ was used to select f, so it became part of the *training* stage.

Model Selection

Proper model selection: We simulate the model evaluation step during the training procedure. This requires one additional data split:

Training and Selecting between Multiple Models

input data $\mathcal{D}$
input set of method $\mathcal{A} = \{A_1, \ldots, A_K\}$
  randomly split $\mathcal{D} = \mathcal{D}_{trnval} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place (and do not look at it)

  randomly split $\mathcal{D}_{trnval} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{val}$ disjointly
  for all possible procedures $A_i \in \mathcal{A}$ **do**
    $f_i \leftarrow\!\shortmid A_i[\mathcal{D}_{trn}]$
    $E_{val}(f_i) \leftarrow\!\shortmid$ performance of f_i on $\mathcal{D}_{val}$
  end for
  $f \leftarrow\!\shortmid f_i$ for $i = \mathbf{argmin}_i \, E_{val}(f_i)$ // pick best performing f_i

  (optional) $f \leftarrow\!\shortmid A_i[\mathcal{D}_{trnval}]$ // retrain best method on full data

  $E_{tst}(f) \leftarrow\!\shortmid$ performance of f on $\mathcal{D}_{tst}$
output trained model f, performance estimate $E_{tst}(f)$

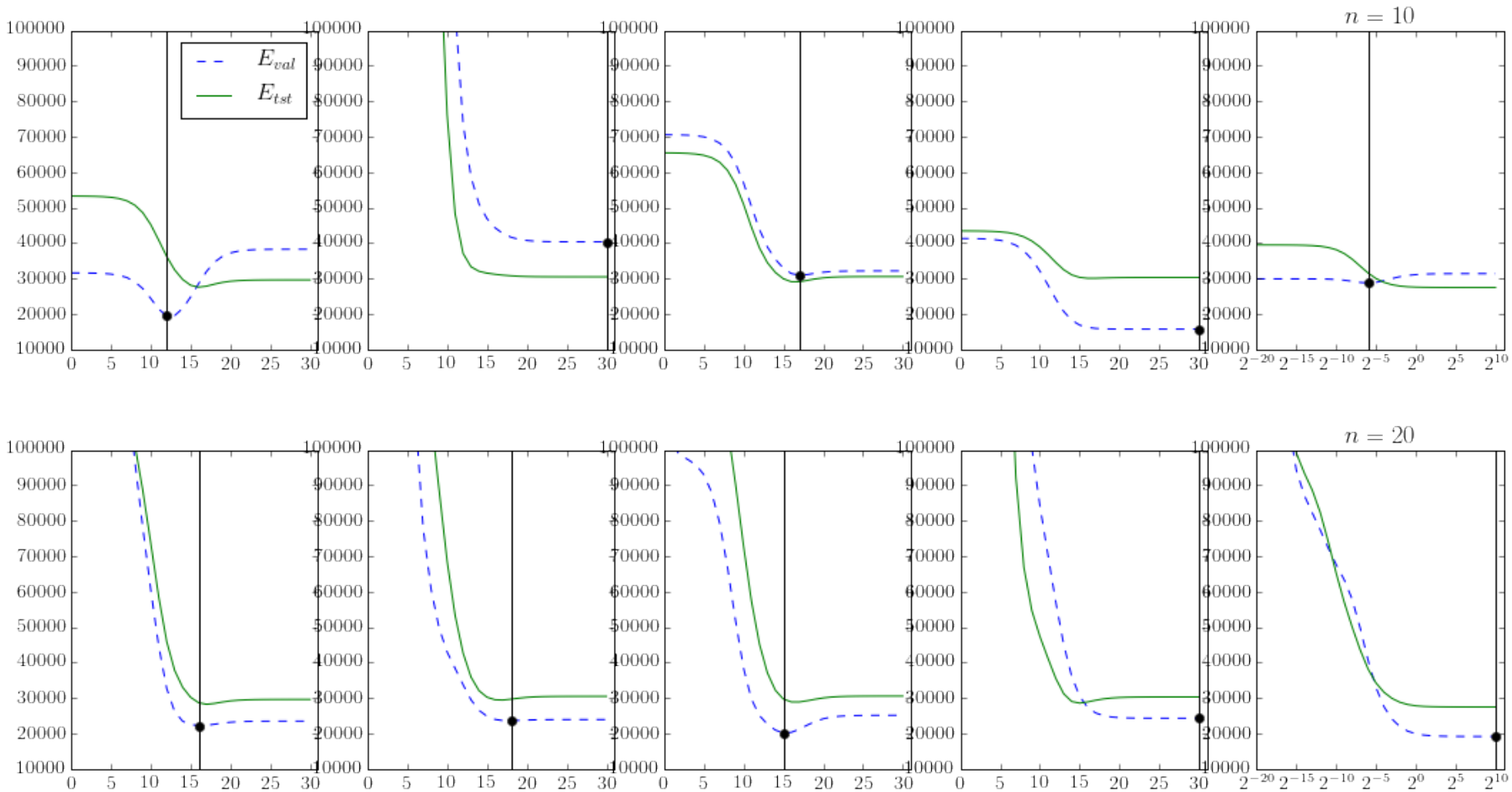
Discussion.

- Each model is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- Which model is selected depends on $\mathcal{D}_{trn}$ and $\mathcal{D}_{val}$ ✓
- Only then the "new" $\mathcal{D}_{tst}$ is used to evaluate the single final model ✓

Problems.

- small $\mathcal{D}_{val}$ is bad: E_{val} could be bad estimate of f_i's true performance, and we might pick a suboptimal method.
- large $\mathcal{D}_{val}$ is bad: $\mathcal{D}_{trn}$ is much smaller than $\mathcal{D}_{trnval}$, so the classifier learned on $\mathcal{D}_{trn}$ might be much worse than necessary.
- retraining the best model on $\mathcal{D}_{trnval}$ might overcome that, but that comes at a risk: just because a model/parameter was the best for $\mathcal{D}_{trn}$, does not mean it is also the best for the larger $\mathcal{D}_{trnval}$.

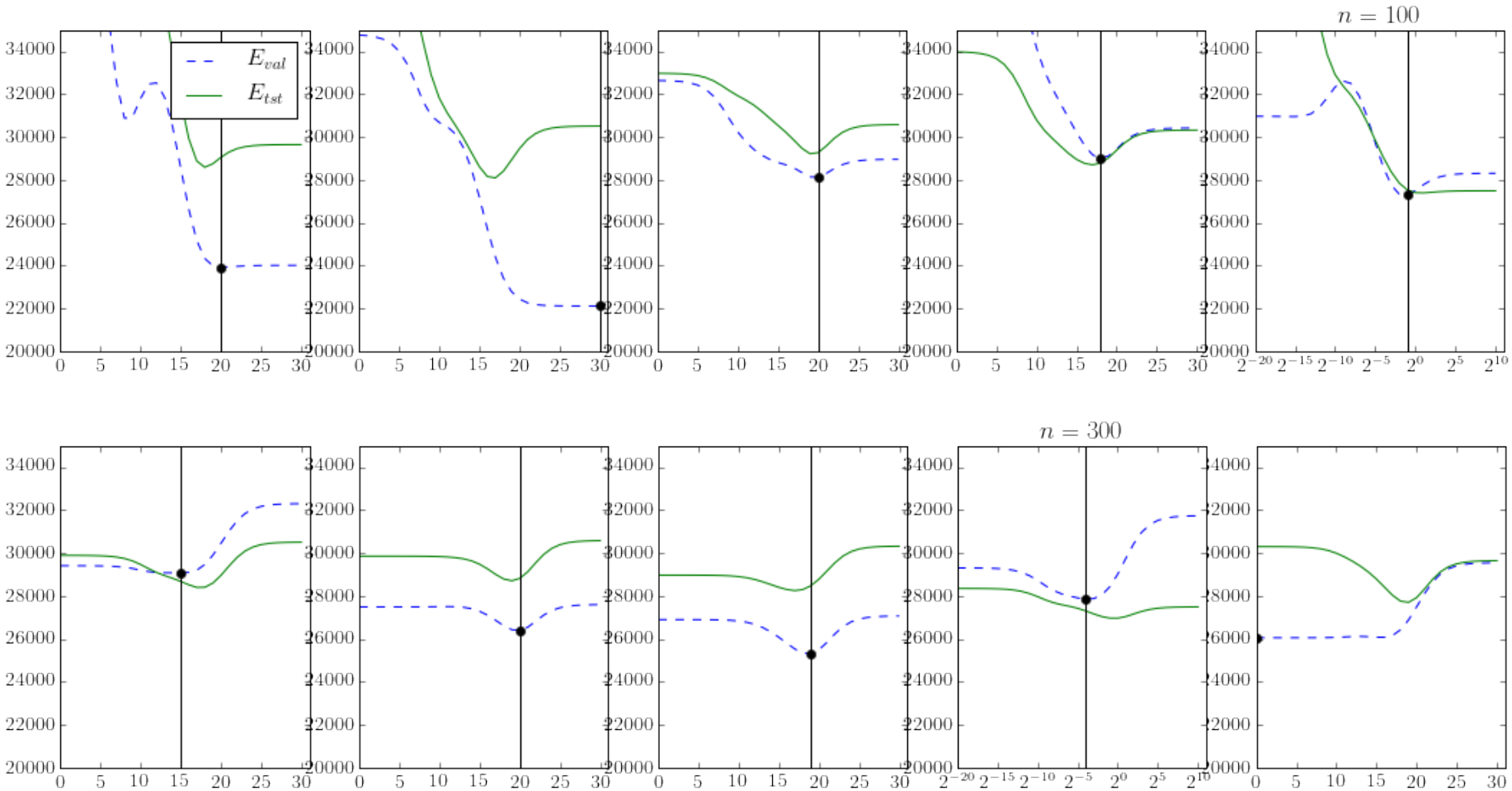
Validation error and test error for different regularization constants:



Model Selection

Validation error and test error for different regularization constants:

`diabetes` dataset, $d = 10$



Parameters from model selection are rarely optimal, but usually reasonable.

Can we use *all data* for training as well as model selection?

Leave-One-Out Evaluation (for a single model/algorithm)

input algorithm A
input data $\mathcal{D}$ (*trnval* part only: *tst* part was set aside earlier)
  for all $(x_i, y_i) \in \mathcal{D}$ **do**
    $f_{\neg i} \leftarrow\!\shortmid A[\ \mathcal{D} \setminus \{(x_i, y_i)\}\ ]$ // $\mathcal{D}_{trn}$ is $\mathcal{D}$ with i-th example removed
    $r_i \leftarrow\!\shortmid$ performance of f_i on (x_i, y_i) // i.e. $\mathcal{D}_{val}$ is $\{(x_i, y_i)\}$
  end for
output $E_{loo}(f) = \frac{1}{n}\sum_{i=1}^{n} r_i$ // average leave-one-out risk

Properties.

- Each r_i is a unbiased (but high variance) estimate of the quality of $f_{\neg i}$
- $\mathcal{D} \setminus \{(x_i, y_i)\}$ is almost the same as $\mathcal{D}$, so we can hope that each $f_{\neg i}$ is almost the same as $f = A[\mathcal{D}]$.
- Therefore, E_{loo} can be expected a good estimate of E on new data

Problem: slow, trains n times on $n-1$ examples instead of once on n
Problem: all training sets are almost the same, r_i are correlated

Compromise: use fixed number of small $\mathcal{D}_{val}$

K-fold Cross Validation (CV)

input algorithm A, loss function ℓ, data $\mathcal{D}$ (trnval part)
  split $\mathcal{D} = \dot{\bigcup}_{k=1}^{K} \mathcal{D}_k$ into K equal sized disjoint parts
  for $k = 1, \ldots, K$ **do**
    $f_{\neg k} \leftarrow\!\shortmid A[\, \mathcal{D} \setminus \mathcal{D}_k \,]$
    $r_k \leftarrow\!\shortmid$ performance of $f_{\neg k}$ on $\mathcal{D}_k$
  end for
output $R_{K\text{-CV}} = \frac{1}{K} \sum_{k=1}^{K} r_k$ (K-fold cross-validation risk)

Observation.

- for $K = |\mathcal{D}|$ same as leave-one-out error.
- approximately k times increase in runtime.
- most common: $k = 10$ or $k = 5$.

Remaining problem: training sets overlap, so the error estimates are not independent, and it's hard to interpret error bars or design statistical tests