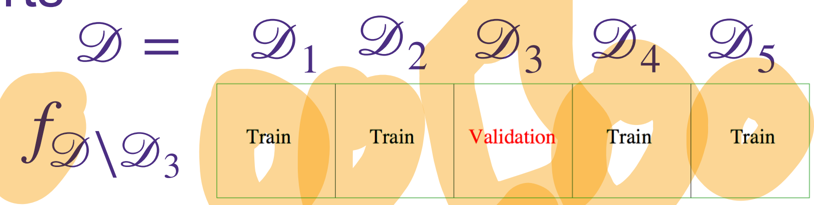
Use *k*-fold cross validation

- Randomly divide training data into *k* equal parts
  - $D_1, \ldots, D_k$
- For each *i*
  - Learn model $f_{\mathscr{D} \setminus \mathscr{D}_i}$ using data point not in $\mathscr{D}_i$
  - Estimate error of $f_{\mathscr{D} \setminus \mathscr{D}_i}$ on validation set $\mathscr{D}_i$:



$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j, y_j) \in \mathcal{D}_i} (y_j - f_{\mathcal{D} \setminus \mathcal{D}_i}(x_j))^2$$

Use *k*-fold cross validation

- Randomly divide training data into *k* equal parts
  - $D_1, \ldots, D_k$
- For each *i*
  - Learn model $f_{\mathscr{D} \backslash \mathscr{D}_i}$ using data point not in $\mathscr{D}_i$
  - Estimate error of $f_{\mathscr{D} \backslash \mathscr{D}_i}$ on validation set $\mathscr{D}_i$:

$$\mathscr{D} = \quad \mathscr{D}_1 \quad \mathscr{D}_2 \quad \mathscr{D}_3 \quad \mathscr{D}_4 \quad \mathscr{D}_5$$

$f_{\mathscr{D} \backslash \mathscr{D}_3}$	Train	Train	Validation	Train	Train

$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j, y_j) \in \mathcal{D}_i} (y_j - f_{\mathcal{D} \backslash \mathcal{D}_i}(x_j))^2$$

- k-fold cross validation error is average over data splits:

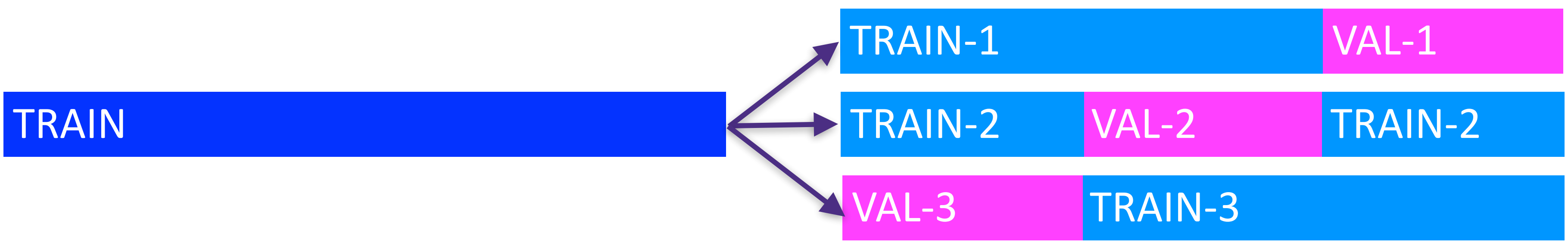
$$\text{error}_{k\text{-fold}} = \frac{1}{k} \sum_{i=1}^{k} \text{error}_{\mathscr{D}_i}$$

- k-fold cross validation properties:
  - Much faster to compute than LOO as $k \ll n$
  - More (pessimistically) biased – using much less data, only $n - \frac{n}{k}$
  - Usually, k = 10

Recap

> Given a dataset, begin by splitting into

> Model selection: Use k-fold cross-validation on TRAIN to train predictor and choose hyper-parameters such as λ



> Model assessment: Use TEST to assess the accuracy of the model you output

- Never ever ever ever ever train or choose parameters based on the test data

Model selection using cross validation

- For $\lambda \in \{0.001, 0.01, 0.1, 1, 10\}$
  - For $j \in \{1, \ldots, k\}$
    - $$\hat{w}_{\lambda,\text{Train}-j} \leftarrow \arg\min_{w} \sum_{i \in \text{Train}-j} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$
- $$\hat{\lambda} \leftarrow \arg\min_{\lambda} \frac{1}{k} \sum_{j=1}^{k} \sum_{i \in \text{Val}-j} (y_i - \hat{w}_{\lambda,\text{Train}-j}^T x_i)^2$$

Example 1

- You wish to predict the stock price of zoom.us given historical stock price data y_i's (for each i-th day) and the historical news articles x_i's
- You use all daily stock price up to Jan 1, 2020 as TRAIN and Jan 2, 2020 - April 13, 2020 as TEST
- What's wrong with this procedure?

Training + test are not identically distributed!!

Example 2

- Given 10,000-dimensional data and n examples, we pick a subset of 50 dimensions that have the highest correlation with labels in the training set:

50 indices j that have largest $$\frac{\left|\sum_{i=1}^{n} x_{i,j} y_i\right|}{\sqrt{\sum_{i=1}^{n} x_{i,j}^2}}$$

- After picking our 50 features, we then use CV with the training set to train ridge regression with regularization λ
- What's wrong with this procedure?

Recap

- Learning is…
  - Collect some data
    - E.g., housing info and sale price
  - Randomly split dataset into TRAIN, VAL, and TEST
    - E.g., 80%, 10%, and 10%, respectively
  - Choose a hypothesis class or model
    - E.g., linear with non-linear transformations
  - Choose a loss function
    - E.g., least squares with ridge regression penalty on TRAIN
  - **Choose an optimization procedure**
    - **E.g., set derivative to zero to obtain estimator, cross-validation on VAL to pick num. features and amount of regularization**
  - Justifying the accuracy of the estimate
    - E.g., report TEST error

Simple variable selection: LASSO for sparse regression

Sparsity

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2$$

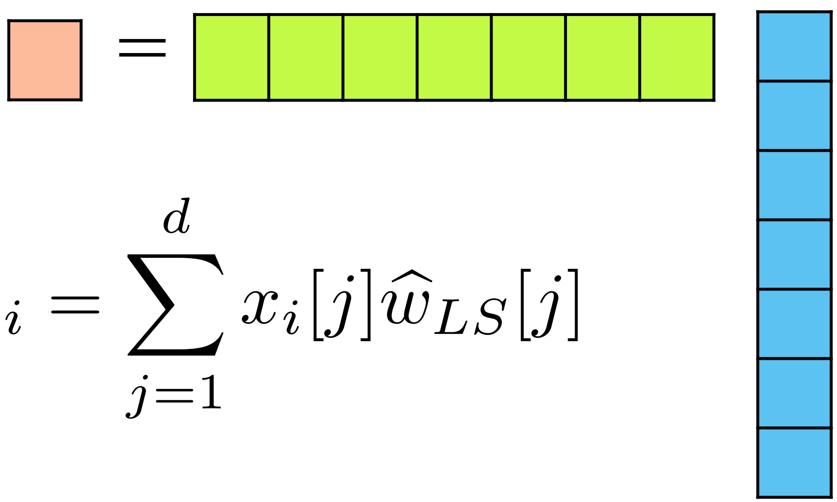
[Ridge $\lambda \|w\|_2^2$]

- Vector w is **sparse**, if many entries are zero

Sparsity

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

- Vector w is **sparse**, if many entries are zero
  - **Efficiency**: If size(w) = 100 Billion, each prediction $w^T x$ is expensive:
    - If w is sparse, prediction computation only depends on number of non-zeros in w



$$\widehat{y}_i = \widehat{w}_{LS}^\top x_i = \sum_{j=1}^{d} x_i[j]\widehat{w}_{LS}[j]$$

Sparsity

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

- Vector w is **sparse**, if many entries are zero
  - **Interpretability**: What are the relevant features to make a prediction?

Lot size
Single Family
Year built
Last sold price
Last sale price/sqft
Finished sqft
Unfinished sqft
Finished basement sqft
floors
Flooring types
Parking type
Parking amount
Cooling
Heating
Exterior materials
Roof type
Structure style

Dishwasher
Garbage disposal
Microwave
Range / Oven
Refrigerator
Washer
Dryer
Laundry location
Heating type
Jetted Tub
Deck
Fenced Yard
Lawn
Garden
Sprinkler System

- How do we find "best" subset of features useful in predicting the price among all possible combinations?

Finding best subset: Exhaustive

- Try all subsets of size 1, 2, 3, … and one that minimizes validation error
- Problem?

Computationally prohibitive

Finding best subset: Greedy

Forward stepwise:
Starting from simple model and iteratively add features most useful to fit

Backward stepwise:
Start with full model and iteratively remove features least useful to fit

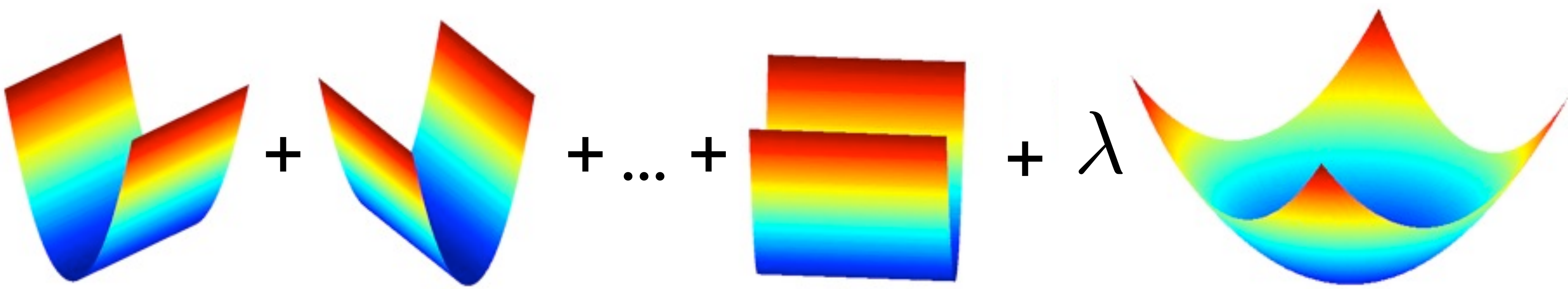
Combining forward and backward steps:
In forward algorithm, insert steps to remove features no longer as important

Lots of other variants, too.

Finding best subset: Regularize

Ridge regression makes coefficients small

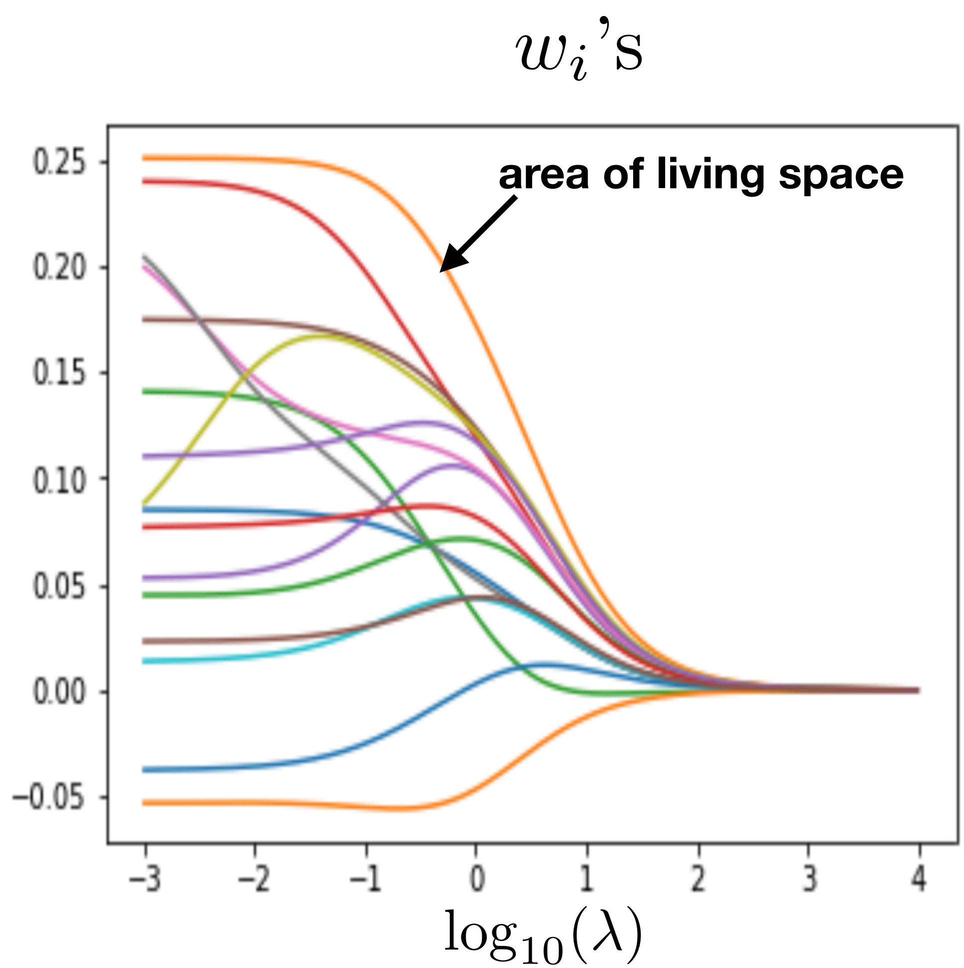
$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



Finding best subset: Regularize

Ridge regression makes coefficients small

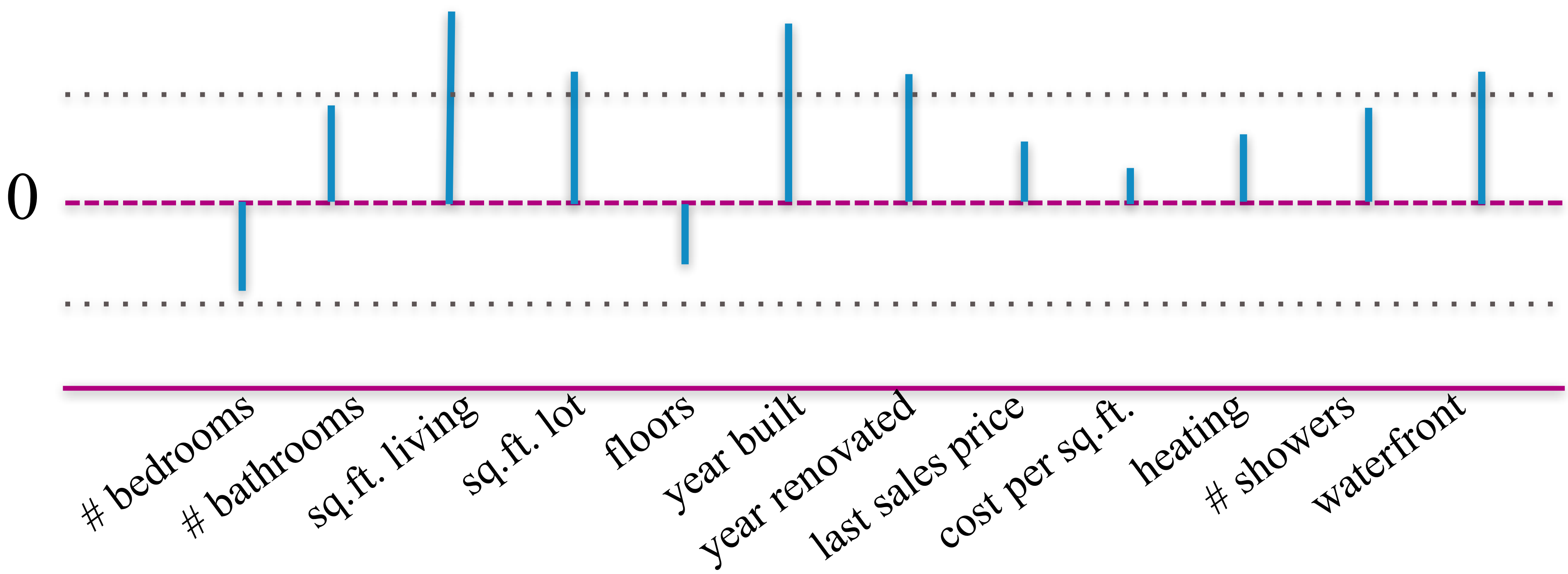
$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



Thresholded Ridge Regression

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

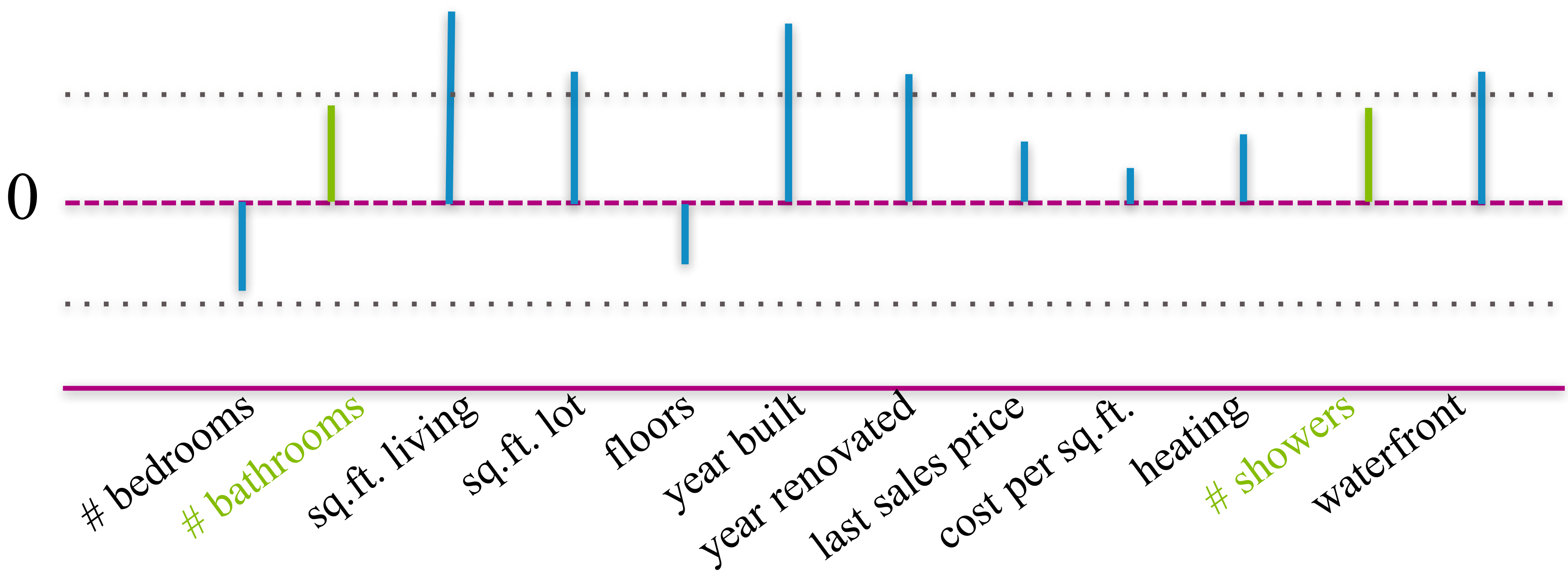
Why don't we just set **small** ridge coefficients to 0?



Thresholded Ridge Regression

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

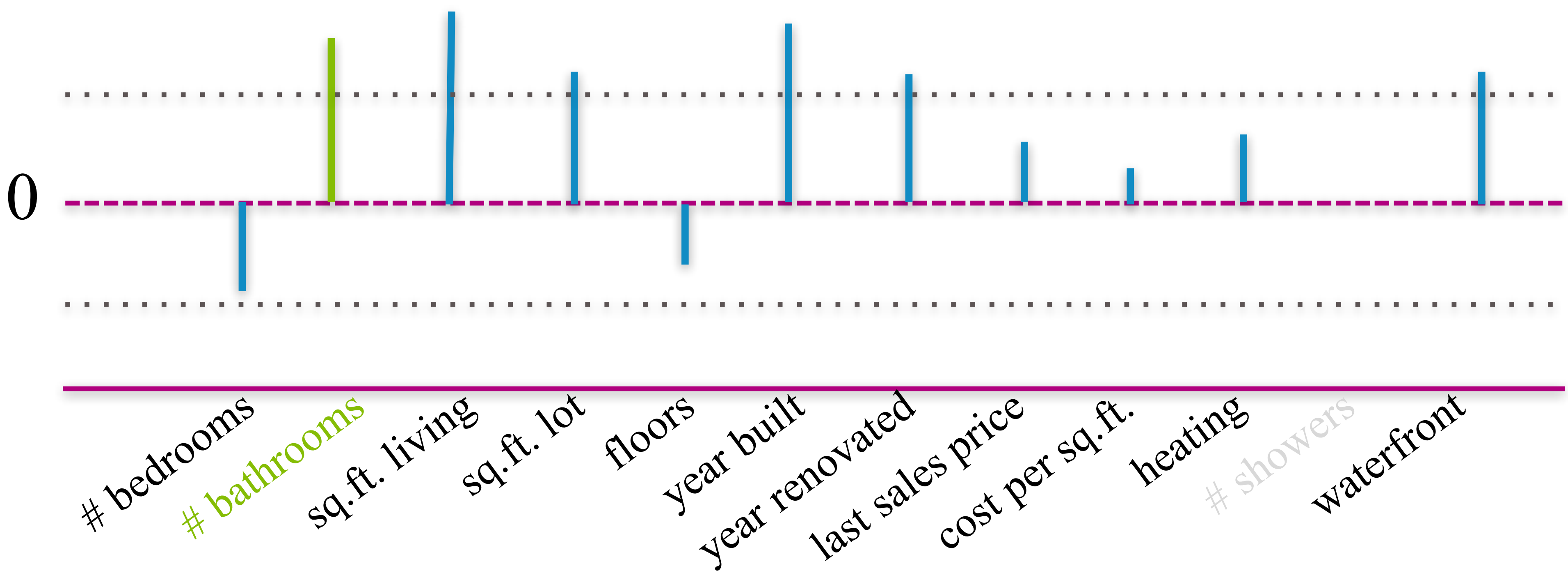
Consider two related features (bathrooms, showers)



Thresholded Ridge Regression

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

What if we **didn't** include showers? Weight on bathrooms increases!

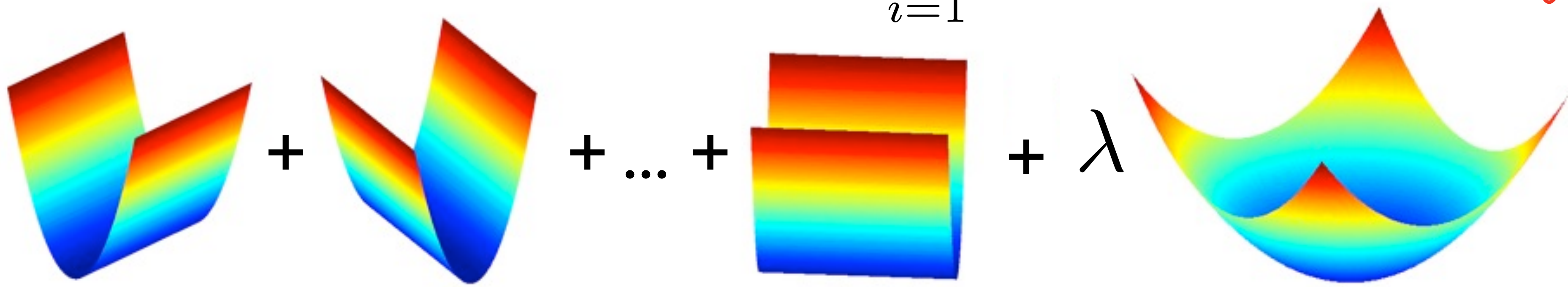


Can another regularizer perform selection automatically?

Recall Ridge Regression

- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda||w||_2^2$$

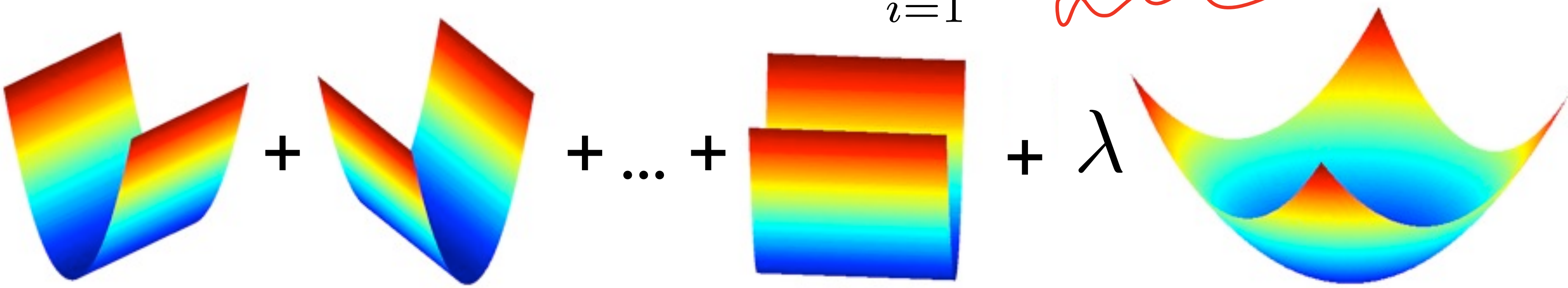


$$\|w\|_p = \left(\sum_{i=1}^{d} |w|^p \right)^{1/p}$$

Ridge vs. Lasso Regression

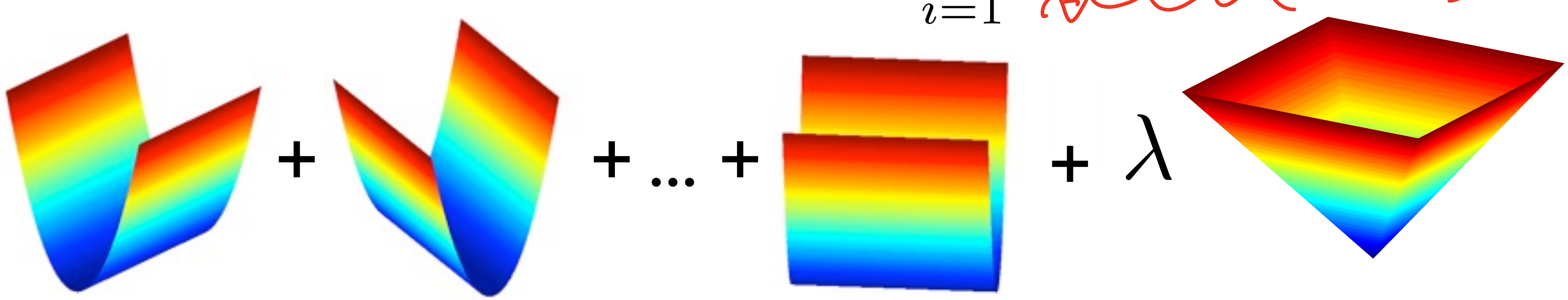
- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



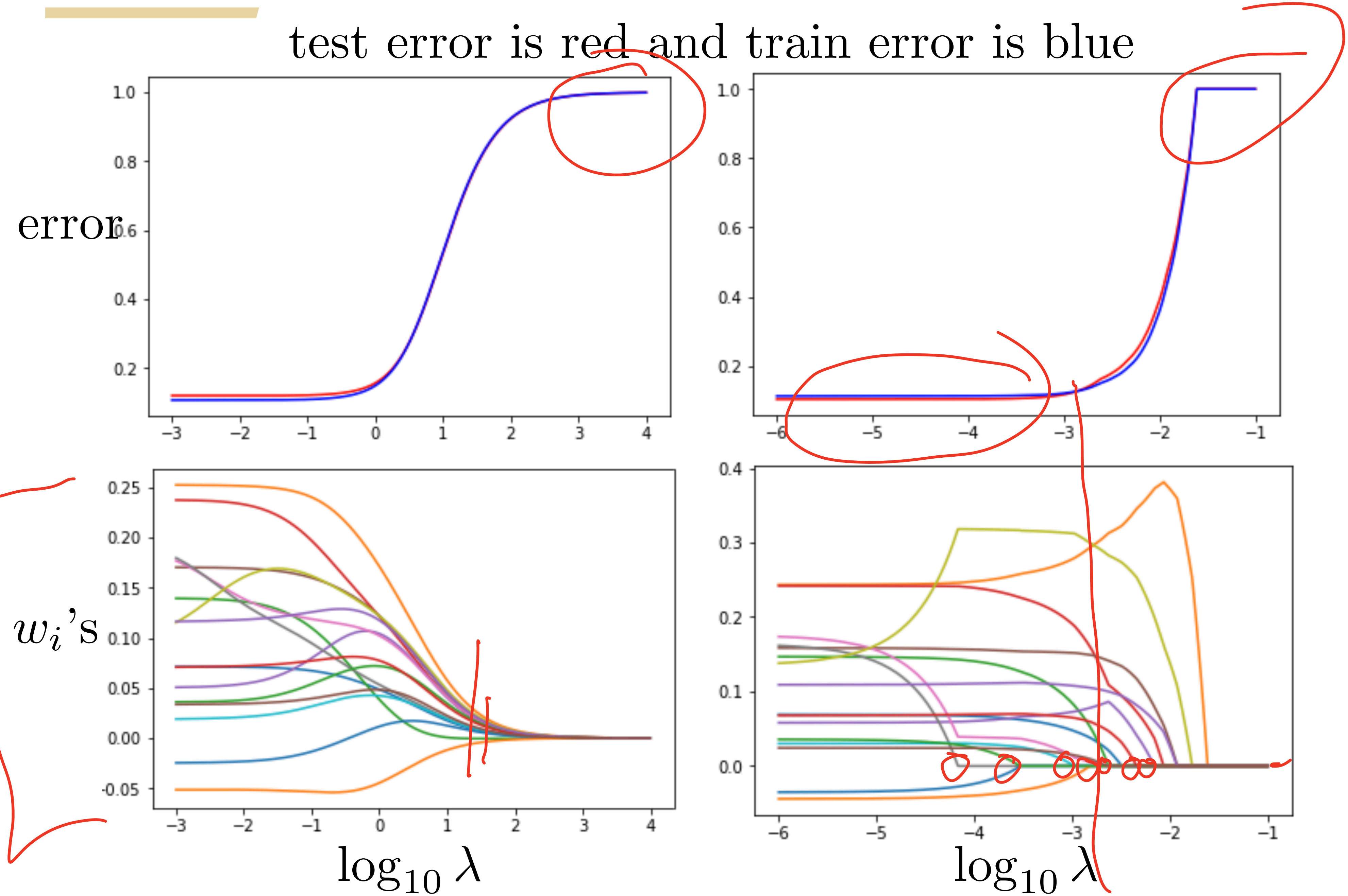
- Lasso objective:

$$\widehat{w}_{lasso} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1$$



Example: house price with 16 features

test error is red and train error is blue



Ridge regression

Lasso regression

Lasso regression naturally gives sparse features

- **feature selection** with Lasso regression

    1. choose λ based on cross validation error
    2. keep only those features with non-zero (or not-too-small) parameters in w at optimal λ
    3. **retrain** with the sparse model and $\lambda = 0$

only on features

Example: piecewise-linear fit

$$h_0(x) = 1$$
$$h_i(x) = [x + 1.1 - 0.1i]^+$$

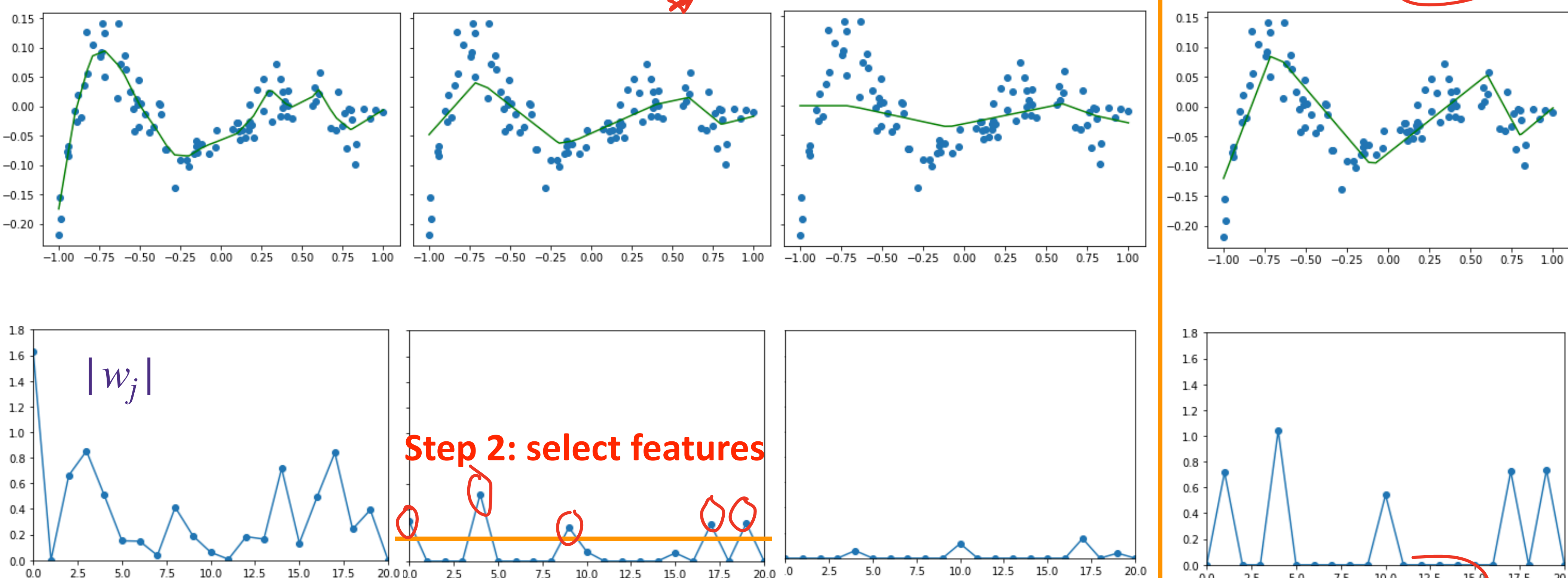
- We use Lasso on the piece-wise linear example

Step 1: find optimal λ^*

$$\text{minimize}_w \ \mathcal{L}(w) + \lambda \|w\|_1$$

Step 3: retrain

$$\text{minimize}_w \ \mathcal{L}(w)$$



$\lambda = 10^{-8}$ $\lambda = 10^{-4}$ $\lambda = 2 \times 10^{-4}$ $\lambda = 0$

but only use selected features

- de-biasing (via re-training) is critical!

Penalized Least Squares

- Regularized optimization:

(1) $$\widehat{w}_r = \arg\min_w \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \lambda r(w)$$

Penalized LS objective [unconstrained]

$$\text{Ridge} : r(w) = ||w||_2^2$$
$$\text{Lasso} : r(w) = ||w||_1$$

Optimal Solution for (1)

- For any $\lambda^* \geq 0$ for which $\hat{w}_r$ achieves the minimum, there exists a $\mu^* \geq 0$ such that the solution of the constrained optimization, $\widehat{w}_c$, is the same as the solution of the regularized optimization , $\widehat{w}_r$, where

(2) $$\widehat{w}_C = \arg\min_w \sum_{i=1}^{n} (y_i - x_i^T w)^2 \quad \text{subject to } r(w) \leq \mu^*$$

- so there are pairs of (λ, μ) whose optimal solution $\widehat{w}_r$ are the same for the regularizes optimization and constrained optimization

Why does Lasso give sparse solutions?

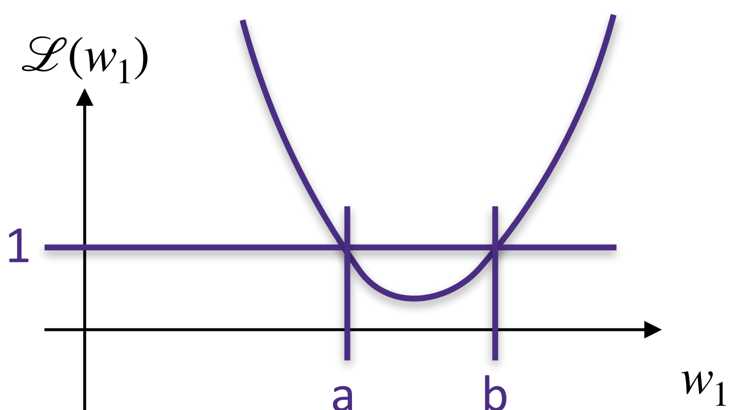
$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

→ Least squares

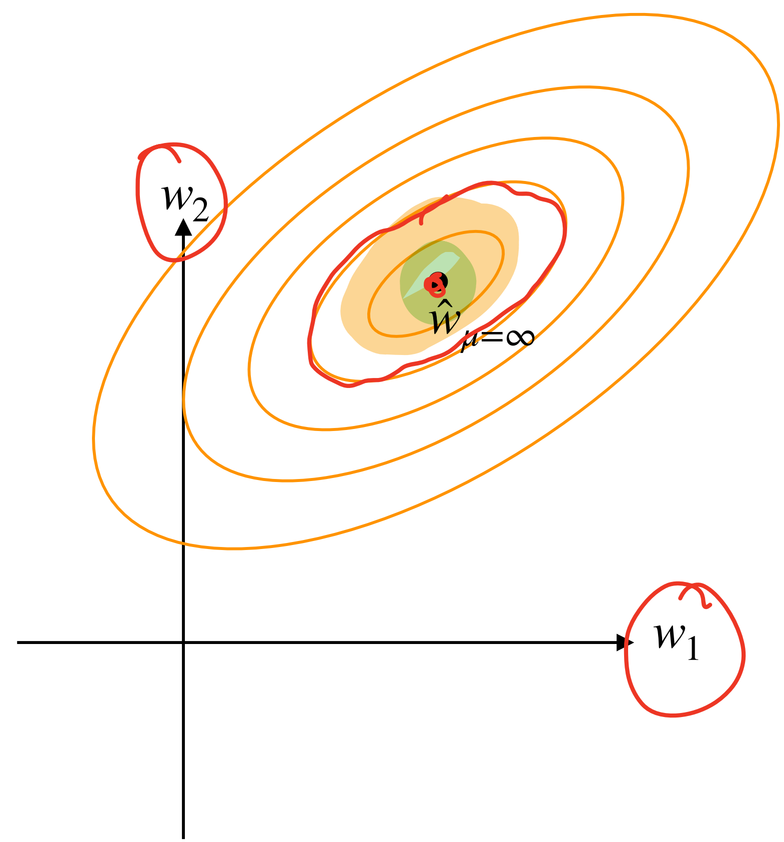
$$\text{subject to} \quad \|w\|_1 \leq \mu$$

- the **level set** of a function $\mathcal{L}(w_1, w_2)$ is defined as the set of points (w_1, w_2) that have the same function value
- the level set of a quadratic function is an oval
- the center of the oval is the least squares solution $\hat{w}_{\mu=\infty} = \hat{w}_{\text{LS}}$

1-D example with quadratic loss



Level set of $\mathcal{L}(w_1)$ at value 1 is [a,b]



Why does Lasso give sparse solutions?

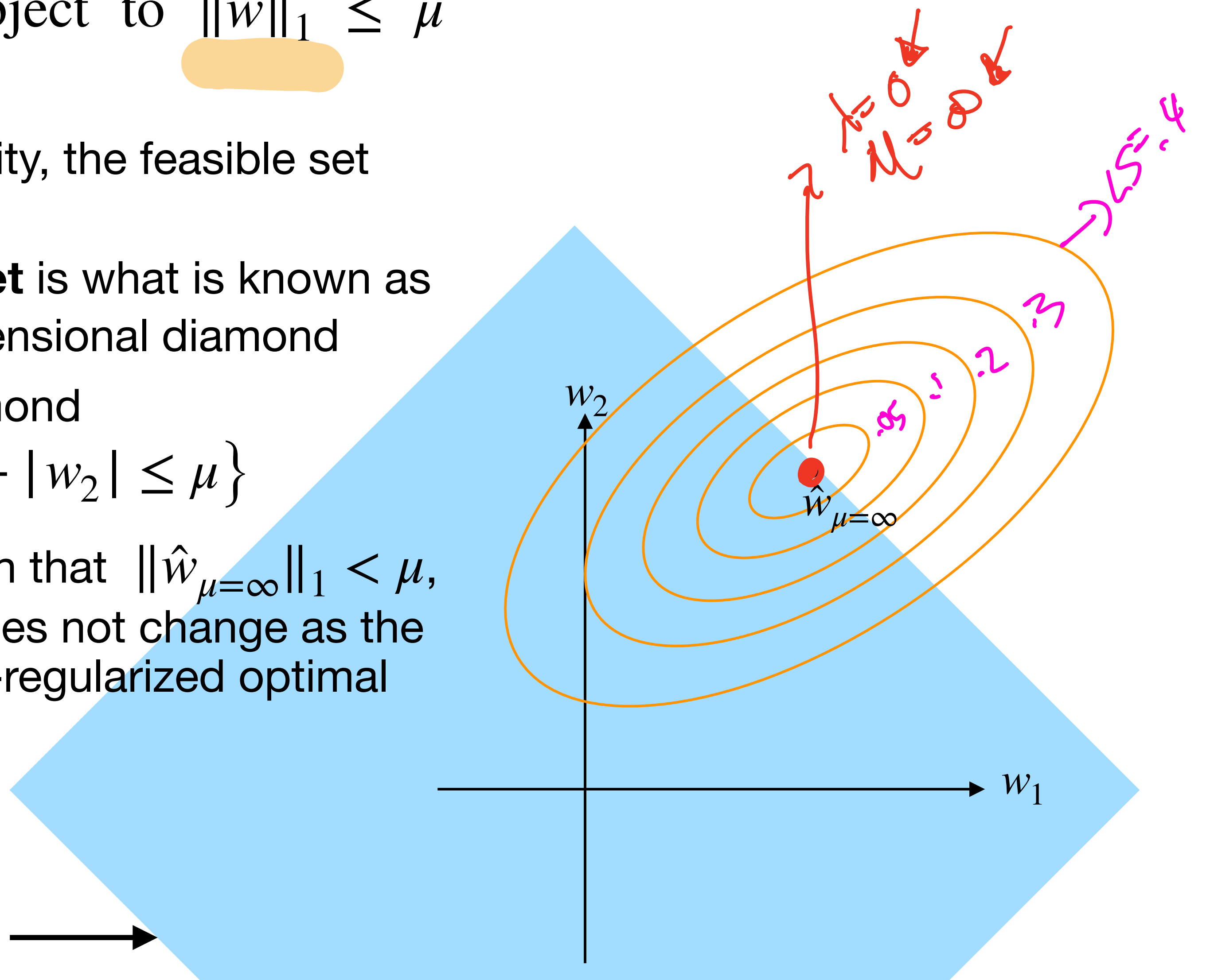
$$\text{minimize}_w \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \leq \mu$$

- as we decrease μ from infinity, the feasible set becomes smaller
- the shape of the **feasible set** is what is known as L_1 ball, which is a high dimensional diamond
- In 2-dimensions, it is a diamond

$$\left\{(w_1, w_2) \,\middle|\, |w_1| + |w_2| \leq \mu \right\}$$

- when μ is large enough such that $\|\hat{w}_{\mu=\infty}\|_1 < \mu$, then the optimal solution does not change as the feasible set includes the un-regularized optimal solution



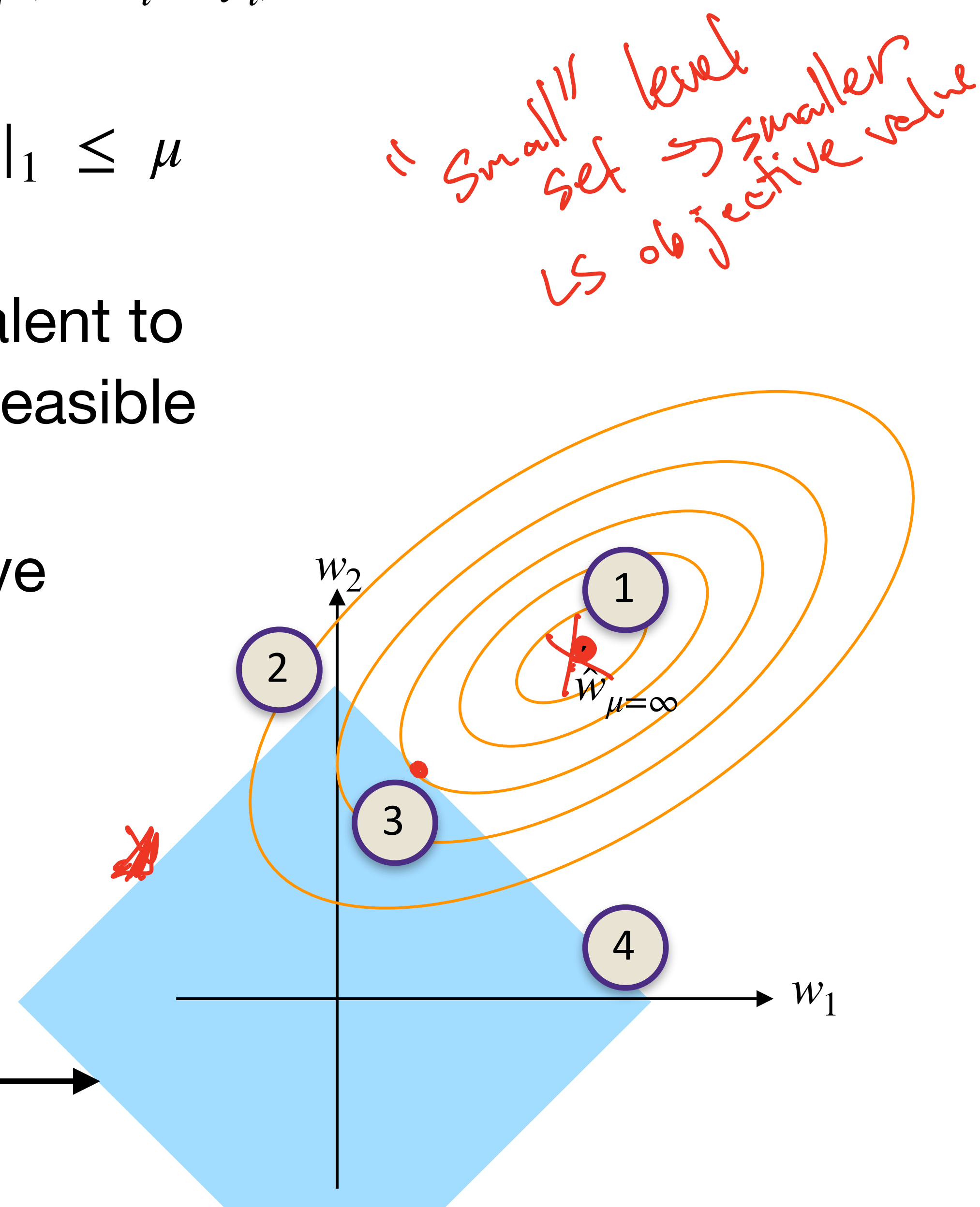
feasible set: $\{w \in \mathbb{R}^2 \mid \|w\|_1 \leq \mu\}$ →

Why does Lasso give sparse solutions?

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \leq \mu$$

- As μ decreases (which is equivalent to increasing regularization λ) the feasible set (blue diamond) shrinks
- The optimal solution of the above optimization is ?



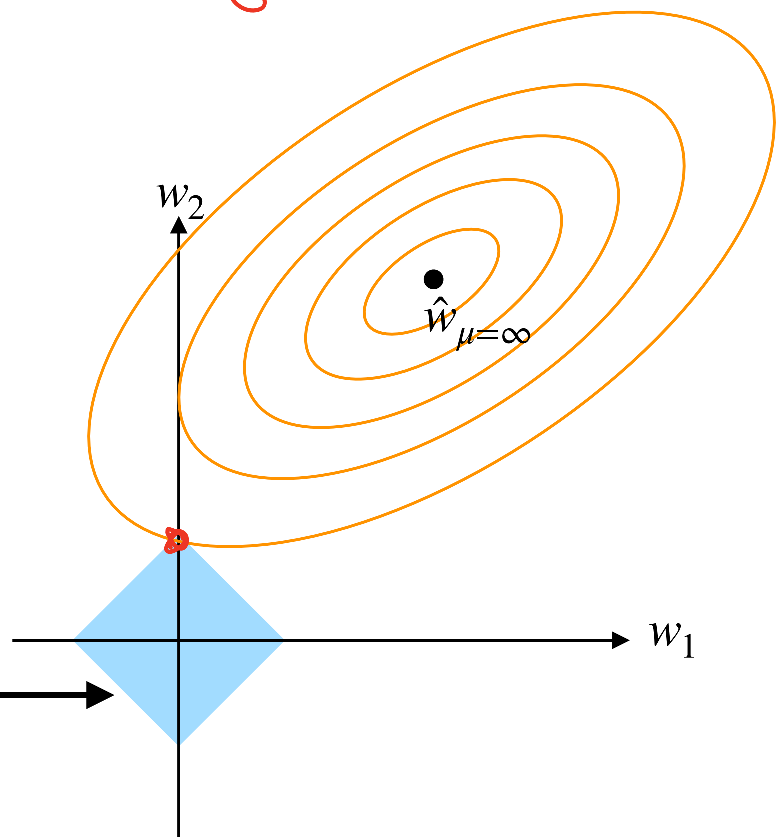
feasible set: $\{w \in \mathbb{R}^2 \mid \|w\|_1 \leq \mu\}$

Why does Lasso give sparse solutions?

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \leq \mu$$

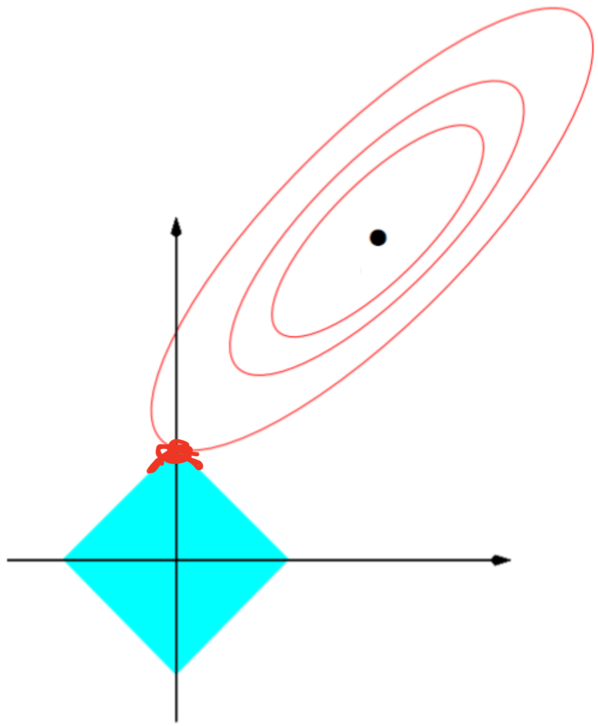
- For small enough μ, the optimal solution becomes **sparse** (large enough λ)
- This is because the L_1-ball is "pointy",i.e., has sharp edges aligned with the axes

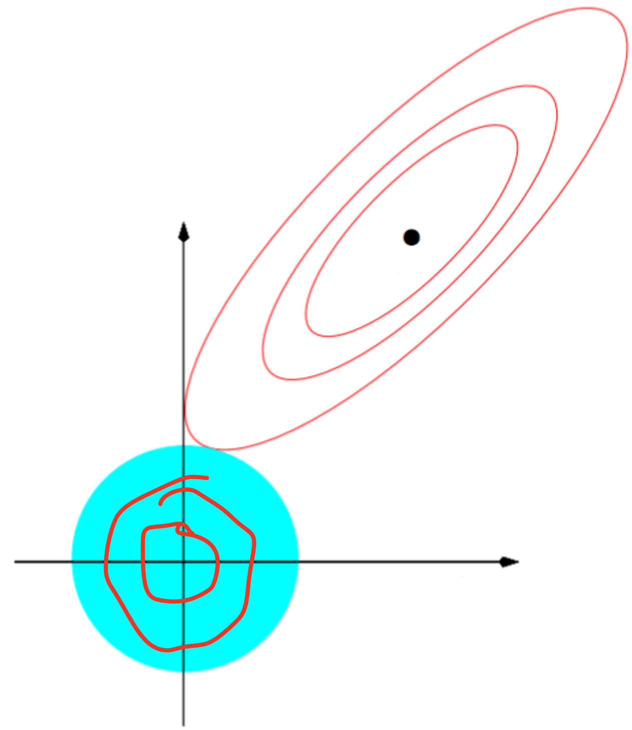


feasible set: $\{w \in \mathbb{R}^2 \mid \|w\|_1 \leq \mu\}$

Penalized Least Squares

- Lasso regression finds sparse solutions, as L_1-ball is "pointy"
- Ridge regression finds dense solutions, as L_2-ball is "smooth"





$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \leq \mu$$

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_2^2 \leq \mu$$

Ridge vs. Lasso

- **Ridge**
  - Very fast:
    - Closed form solution if used with linear models
    - Even with non-linear and complex loss, optimization is fast for squared ℓ_2 regularization (to be taught later)
  - Gives regularized parameters that avoid overfitting

- **Lasso**
  - Slower than Ridge:
    - No closed form!
    - A non-smooth optimization which is slower
      - (to be taught later)
  - Gives sparse parameters

Questions?