

Is the test error unbiased for these programs?

```
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
mu = np.mean(X, axis=0)
X = X - mu

idx = np.random.permutation(1000)
TRAIN = idx[0:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]

# Solve for argmin_w ||Xtrain*w - ytrain||_2
w = np.linalg.solve( np.dot(Xtrain.T, Xtrain),
                    np.dot(Xtrain.T, ytrain) )
b = np.mean(ytrain)

ytest = y[TEST]
Xtest = X[TEST,:]

train_error = np.dot( np.dot(Xtrain, w)+b - ytrain,
                    np.dot(Xtrain, w)+b - ytrain )/len(TRAIN)
test_error = np.dot( np.dot(Xtest, w)+b - ytest,
                    np.dot(Xtest, w)+b - ytest )/len(TEST)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

```
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
idx = np.random.permutation(1000)
TRAIN = idx[0:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]
Xtrain_avg = np.mean(Xtrain, axis=0)
Xtrain = Xtrain - Xtrain_avg

# Solve for argmin_w ||Xtrain*w - ytrain||_2
w = np.linalg.solve( np.dot(Xtrain.T, Xtrain),
                    np.dot(Xtrain.T, ytrain) )
b = np.mean(ytrain)

ytest = y[TEST]
Xtest = X[TEST,:]
Xtest_avg = np.mean(Xtest, axis=0)
Xtest = Xtest - Xtest_avg

train_error = np.dot( np.dot(Xtrain, w)+b - ytrain,
                    np.dot(Xtrain, w)+b - ytrain )/len(TRAIN)
test_error = np.dot( np.dot(Xtest, w)+b - ytest,
                    np.dot(Xtest, w)+b - ytest )/len(TEST)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

Is the test error unbiased for this program?

```
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
idx = np.random.permutation(1000)
TRAIN = idx[0:800]
VAL = idx[800:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]
yval = y[VAL]
Xval = X[VAL,:]

err = np.zeros(50)
for d in range(1,51):
    w, b = fit(Xtrain[:,0:d], ytrain)
    yval_hat = predict(w, b, Xval[:,0:d])
    err[d-1] = np.mean((yval_hat-yval)**2)
d_best = np.argmin(err)+1

Xtot = np.concatenate((Xtrain, Xval), axis=0)
ytot = np.concatenate((ytrain, yval), axis=0)
w, b = fit(Xtot[:,0:d_best], ytot)

ytest = y[TEST]
Xtest = X[TEST,:]

ytot_hat = predict(w, b, Xtot[:,0:d_best])
tot_train_error = np.mean((ytot_hat-ytot)**2)
ytest_hat = predict(w, b, Xtest[:,0:d_best])
test_error = np.mean((ytest_hat-ytest)**2)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

```
def fit(Xin, Yin):
    mu = np.mean(Xin, axis=0)
    Xin = Xin - mu
    w = np.linalg.solve( np.dot(Xin.T, Xin),
                          np.dot(Xin.T, Yin) )
    b = np.mean(Yin) - np.dot(w, mu)
    return w, b
def predict(w, b, Xin):
    return np.dot(Xin, w)+b
```

Cross-Validation

How... How... How????????

- > How do we pick the regularization constant λ ...
- > How do we pick the number of basis functions...

- > We could use the test data, but...

(LOO) Leave-one-out cross validation

- > Consider a validation set with 1 example:
 - D – training data
 - $D \setminus j$ – training data with j th data point (x_j, y_j) moved to validation set
- > Learn classifier $f_{D \setminus j}$ with $D \setminus j$ dataset
- > Estimate true error as squared error on predicting y_j :
 - Unbiased estimate of error $\text{error}_{\text{true}}(f_{D \setminus j})!$
- > **LOO cross validation:** Average over all data points j :
 - For each data point you leave out, learn a new classifier $f_{D \setminus j}$
 - Estimate error as:

(LOO) Leave-one-out cross validation

- > Consider a validation set with 1 example:
 - D – training data
 - $D \setminus j$ – training data with j th data point (x_j, y_j) moved to validation set
- > Learn classifier $f_{D \setminus j}$ with $D \setminus j$ dataset
- > Estimate true error as squared error on predicting y_j :
 - Unbiased estimate of error_{true}($f_{D \setminus j}$)!
- > LOO cross validation: Average over all data points j :
 - For each data point you leave out, learn a new classifier $f_{D \setminus j}$
 - Estimate error as:

$$\text{error}_{LOO} = \frac{1}{n} \sum_{j=1}^n (y_j - f_{D \setminus j}(x_j))^2$$

LOO cross validation is (almost) unbiased estimate!

- > When computing LOOCV error, we only use $N-1$ data points
 - So it's not estimate of true error of learning with N data points
 - Usually pessimistic, though – learning with less data typically gives worse answer

- > LOO is almost unbiased! Use LOO error for model selection!!!
 - E.g., picking λ

Computational cost of LOO

- > **Suppose you have 100,000 data points**
- > **You implemented a great version of your learning algorithm**
 - **Learns in only 1 second**
- > **Computing LOO will take about 1 day!!!**
 -

Use k -fold cross validation

> Randomly divide training data into k equal parts

– D_1, \dots, D_k

> For each i

– Learn classifier $f_{D \setminus D_i}$ using data point not in D_i

– Estimate error of $f_{D \setminus D_i}$ on validation set D_i :

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

1	2	3	4	5
Train	Train	Validation	Train	Train

Use k -fold cross validation

> Randomly divide training data into k equal parts

- D_1, \dots, D_k

1	2	3	4	5
Train	Train	Validation	Train	Train

> For each i

- Learn classifier $f_{D \setminus D_i}$ using data point not in D_i
- Estimate error of $f_{D \setminus D_i}$ on validation set D_i :

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

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

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

> k -fold cross validation properties:

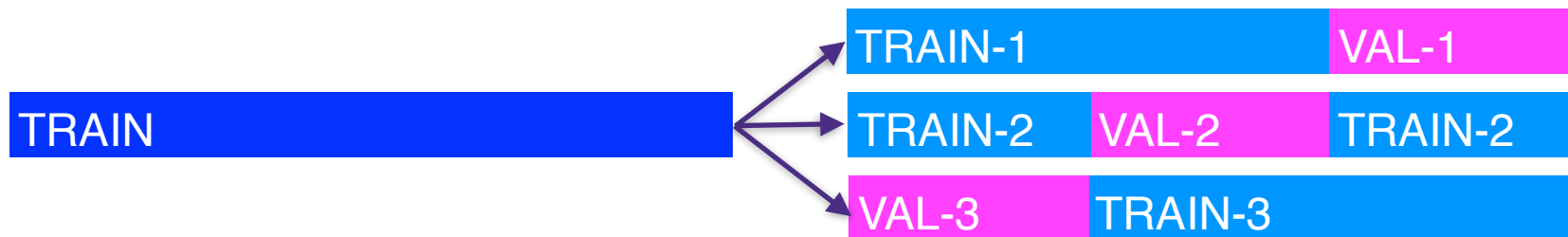
- Much faster to compute than LOO
- More (pessimistically) biased – using much less data, only $n(k-1)/k$
- Usually, $k = 10$

Recap

- > Given a dataset, begin by splitting into



- > Model selection: Use k-fold cross-validation on **TRAIN** to train predictor and choose magic 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**

Example 1

- > You wish to predict the stock price of zoom.us given historical stock price data
- > 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?

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 $\frac{|\sum_{i=1}^n x_{i,j} y_i|}{\sqrt{\sum_{i=1}^n x_{i,j}^2}}$

- > 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: Sparse Regression

Sparsity

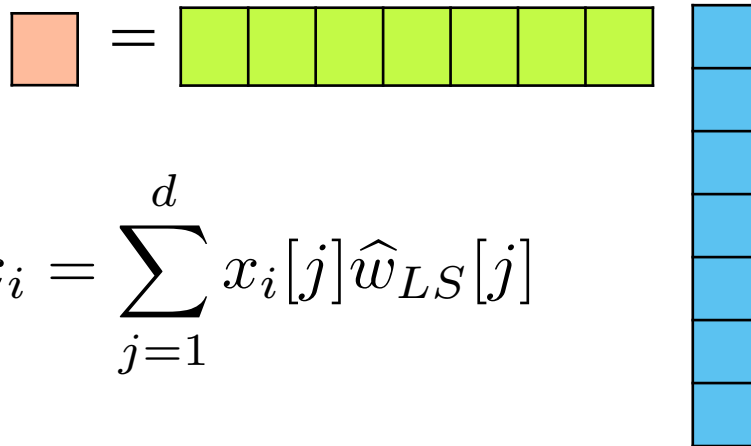
$$\hat{w}_{LS} = \arg \min_w \sum_{i=1}^n (y_i - x_i^T w)^2$$

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

Sparsity

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

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



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

Sparsity

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

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



Lot size	Dishwasher
Single Family	Garbage disposal
Year built	Microwave
Last sold price	Range / Oven
Last sale price/sqft	Refrigerator
Finished sqft	Washer
Unfinished sqft	Dryer
Finished basement sqft	Laundry location
# floors	Heating type
Flooring types	Jetted Tub
Parking type	Deck
Parking amount	Fenced Yard
Cooling	Lawn
Heating	Garden
Exterior materials	Sprinkler System
Roof type	
Structure style	

- How do we find “best” subset among all possible?

Finding best subset: Exhaustive

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

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:

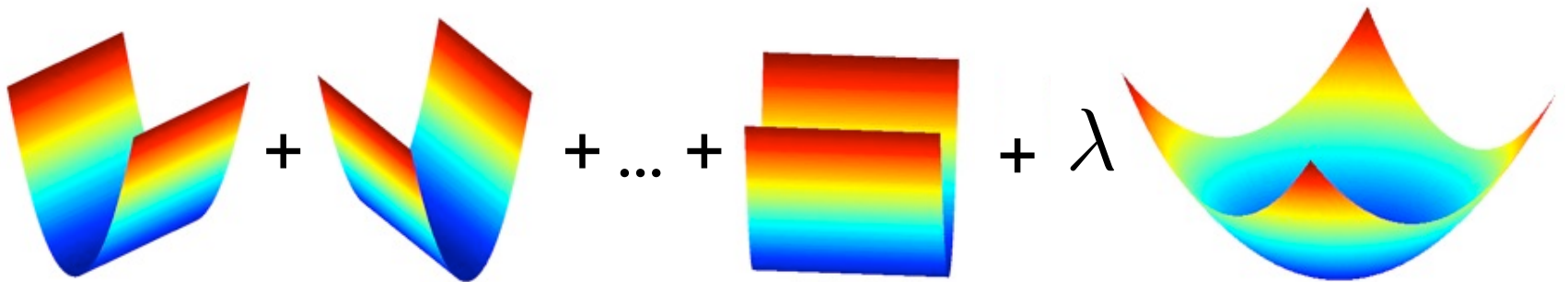
In forward algorithm, insert steps to remove features no longer as important

Lots of other variants, too.

Finding best subset: Regularize

Ridge regression makes coefficients small

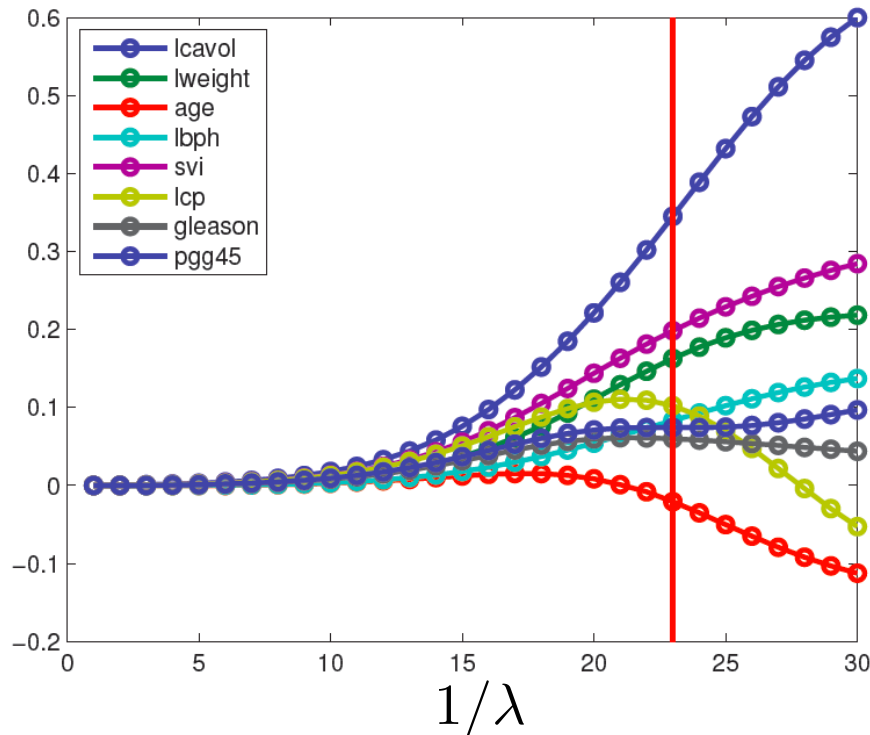
$$\hat{w}_{ridge} = \arg \min_w \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda \|w\|_2^2$$



Finding best subset: Regularize

Ridge regression makes coefficients small

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

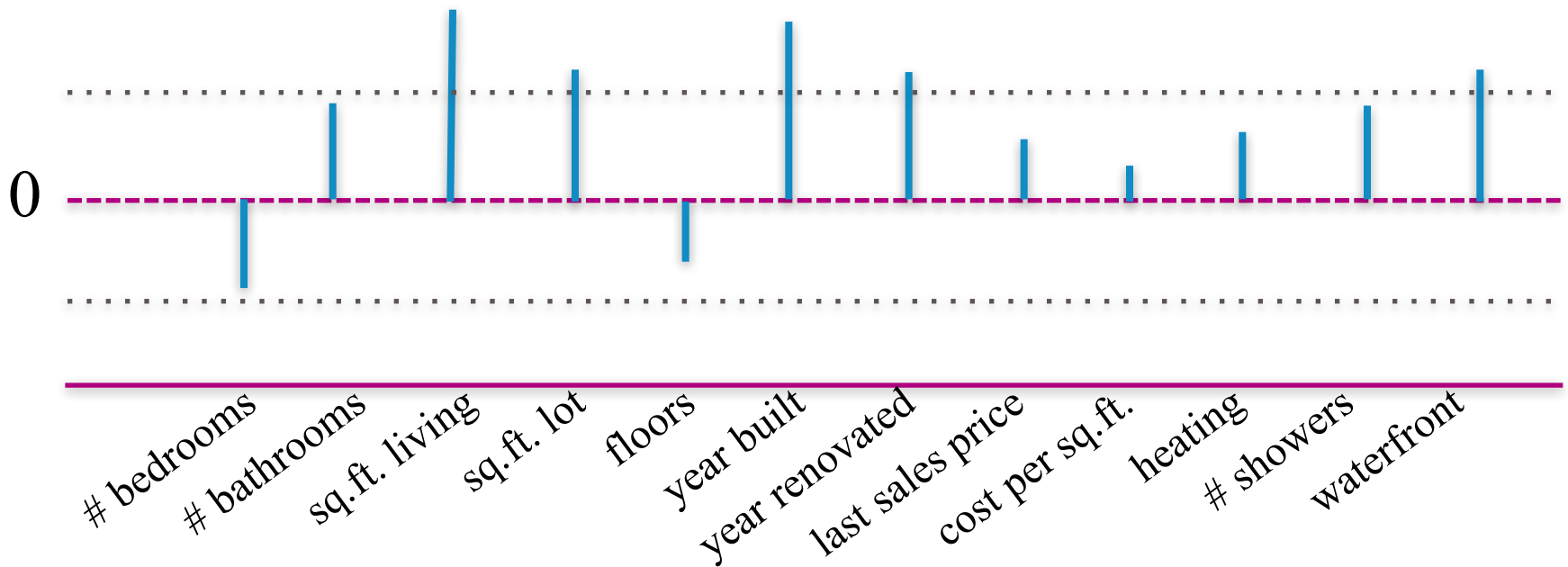


From
Kevin Murphy
textbook

Thresholded Ridge Regression

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

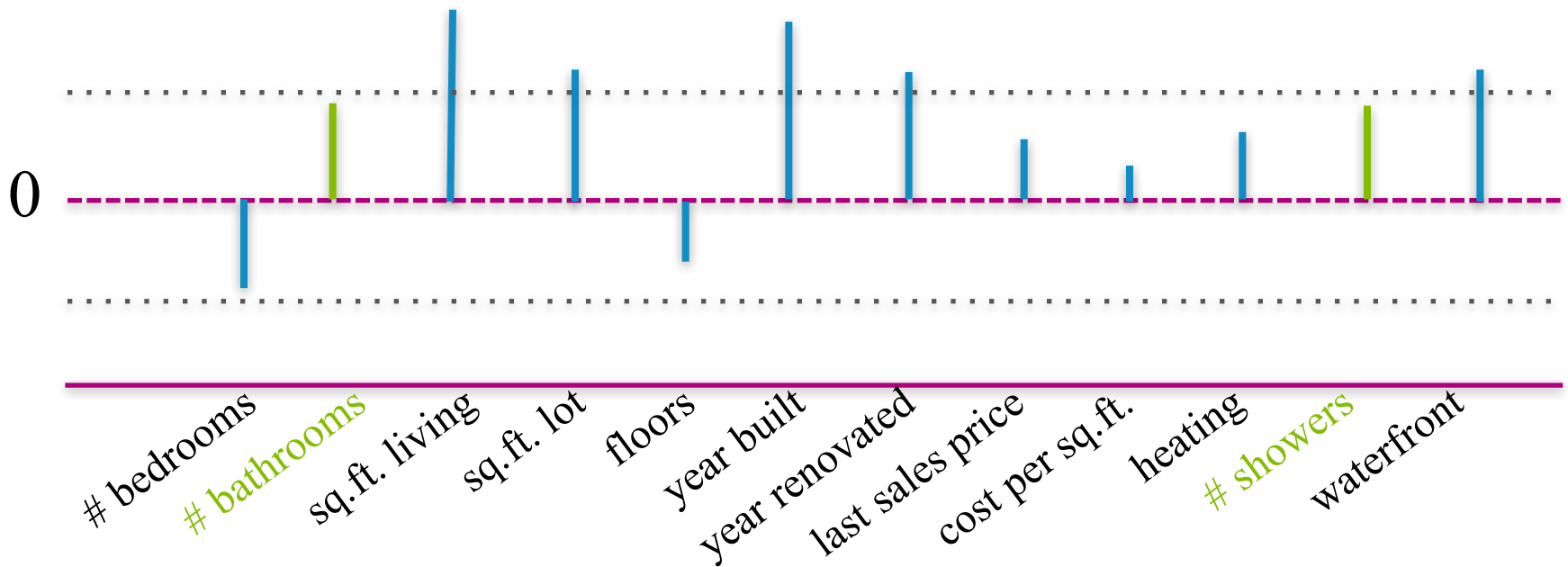
Why don't we just set **small** ridge coefficients to 0?



Thresholded Ridge Regression

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

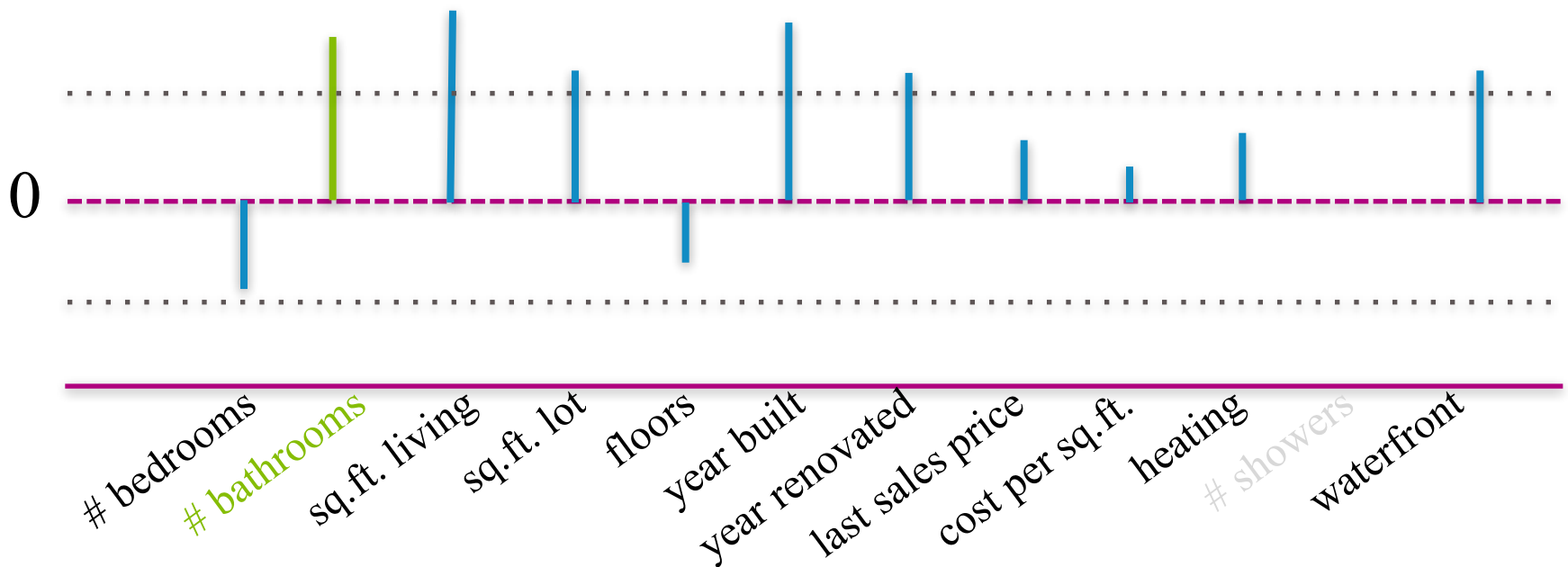
Consider two **related** features (bathrooms, showers)



Thresholded Ridge Regression

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

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

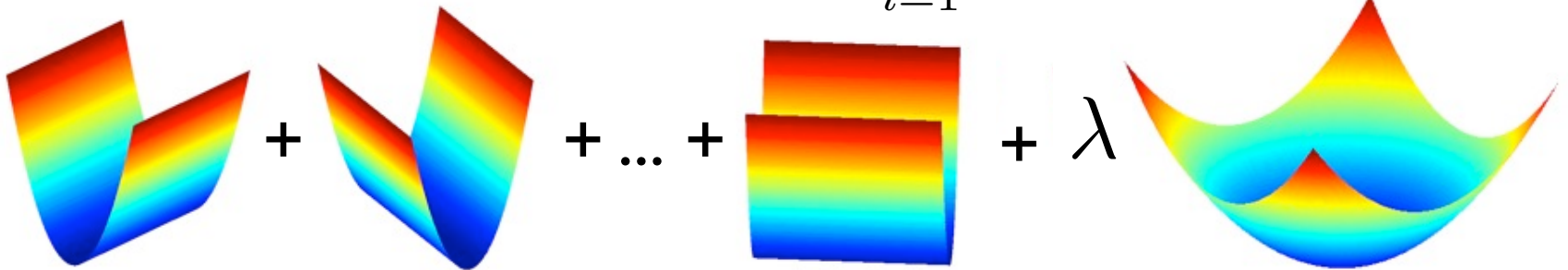


Can another regularizer perform selection automatically?

Recall Ridge Regression

- Ridge Regression objective:

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

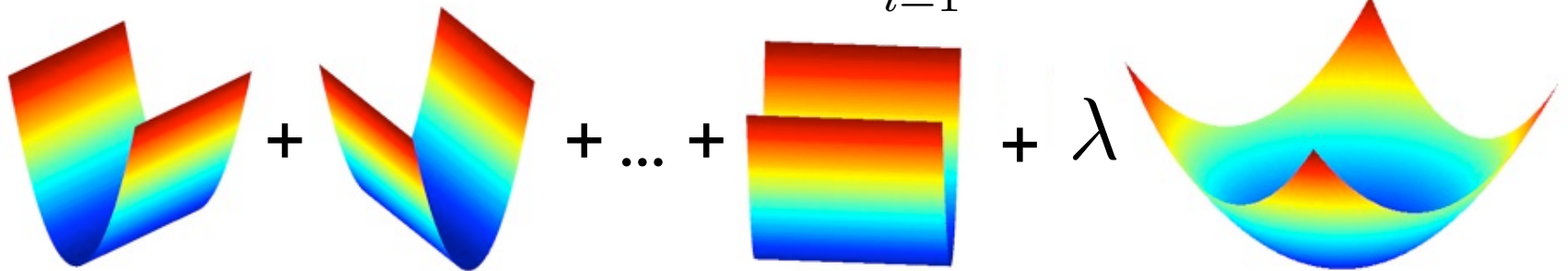


$$\|w\|_p = \left(\sum_{i=1}^d |w|^p \right)^{1/p}$$

Ridge vs. Lasso Regression

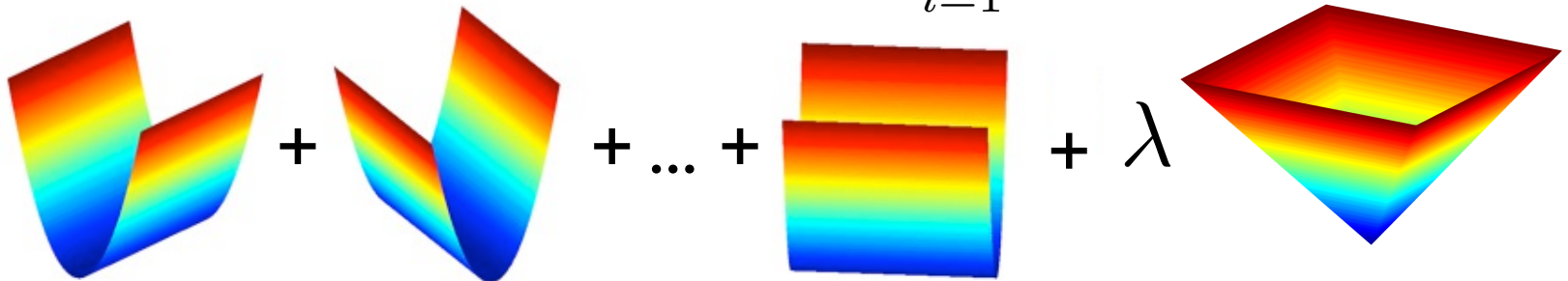
- Ridge Regression objective:

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



- Lasso objective:

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



Penalized Least Squares

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

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

Penalized Least Squares

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

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

For any $\lambda \geq 0$ for which \hat{w}_r achieves the minimum, there exists a $\nu \geq 0$ such that

$$\hat{w}_r = \arg \min_w \sum_{i=1}^n (y_i - x_i^T w)^2 \quad \text{subject to } r(w) \leq \nu$$

Penalized Least Squares

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

$$\text{Lasso : } r(w) = \|w\|_1$$

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

For any $\lambda \geq 0$ for which \hat{w}_r achieves the minimum, there exists a $\nu \geq 0$ such that

$$\hat{w}_r = \arg \min_w \sum_{i=1}^n (y_i - x_i^T w)^2 \quad \text{subject to } r(w) \leq \nu$$

