CSE/STAT 416

Cross Validation; Ridge Regression

Pre-Class Videos (lecture slides below)

Hunter Schafer
University of Washington
April 3, 2023

Pre-Class Video 1:

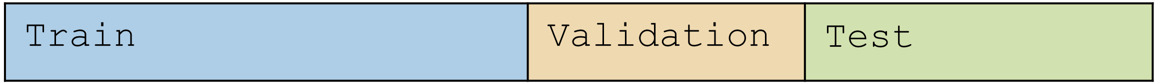
Cross Validation

Validation Set

Important: this should be randomized!!!

So far we have divided our dataset into train and test

We can't use Test to choose our model complexity, so instead, break up Train into ANOTHER dataset



e.g., 70% 15% 15%

We will pick the model that does best on validation. Note that this now makes the validation error of the "best" model a biased estimate of true error. The test error will be an unbiased estimate though since we never looked at it!

Validation Set

The process generally goes

```
train, validation, test = random_split(dataset)
for each model complexity p:   any hyper-parameter
    model = train_model(model_p, train)
    val_err = error(model, validation)
    keep track of p and model with smallest val_err
return best p & error(model, test)
```

Validation Set

Pros

Easy to describe and implement

Pretty fast

- Only requires training a model and predicting on the validation set for each complexity of interest

Cons

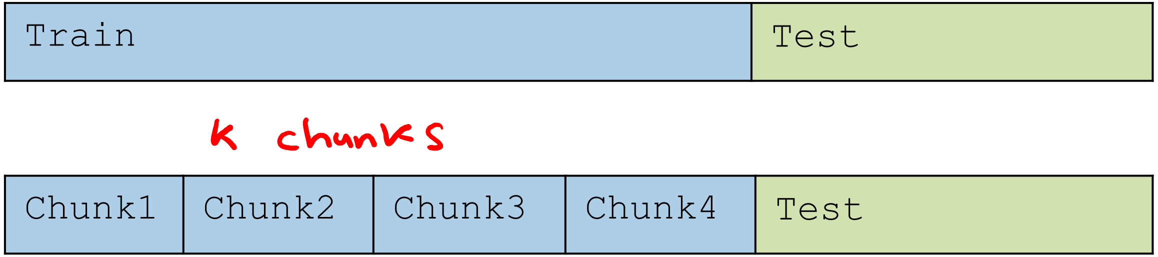
- Have to sacrifice even more training data
- Prone to overfitting*

Cross-Validation

This should be randomized!

Clever idea: Use many small validation sets without losing too much training data.

Still need to break off our test set like before. After doing so, break the training set into k chunks.



For a given model complexity, train it k times. Each time use all but one chunk and use that left out chunk to determine the validation error.

Cross Validation

For a set of hyperparameters, perform Cross Validation on k folds

Cross-Validation

The process generally goes

```
chunk_1, …, chunk_k, test = random_split(dataset)
for each model complexity p:
    for i in [1, k]:
        model = train_model(model_p, chunks - i)
        val_err = error(model, chunk_i)
    avg_val_err = average val_err over chunks
    keep track of p with smallest avg_val_err
return model trained on train (all chunks) with
best p & error(model, test)
```

iterate over hyperparameter settings

Interpretation: average validation error of models with complexity p.

Cross-Validation

Pros

- Prevent overfitting: By training the model on multiple folds instead of only 1 training set, this learns the model with the best generalization capabilities.
- Don't have to actually get rid of any training data!

Cons

- Slow. For each model selection, we have to train k times
- Very computationally expensive

} these go hand-in-hand

Cross-Validation

What size of k?

- Theoretical best estimator is to use $k = n$
  - Called "Leave One Out Cross Validation”
- In practice, people use $k = 5$ to 10.

Pre-Class Video 1:

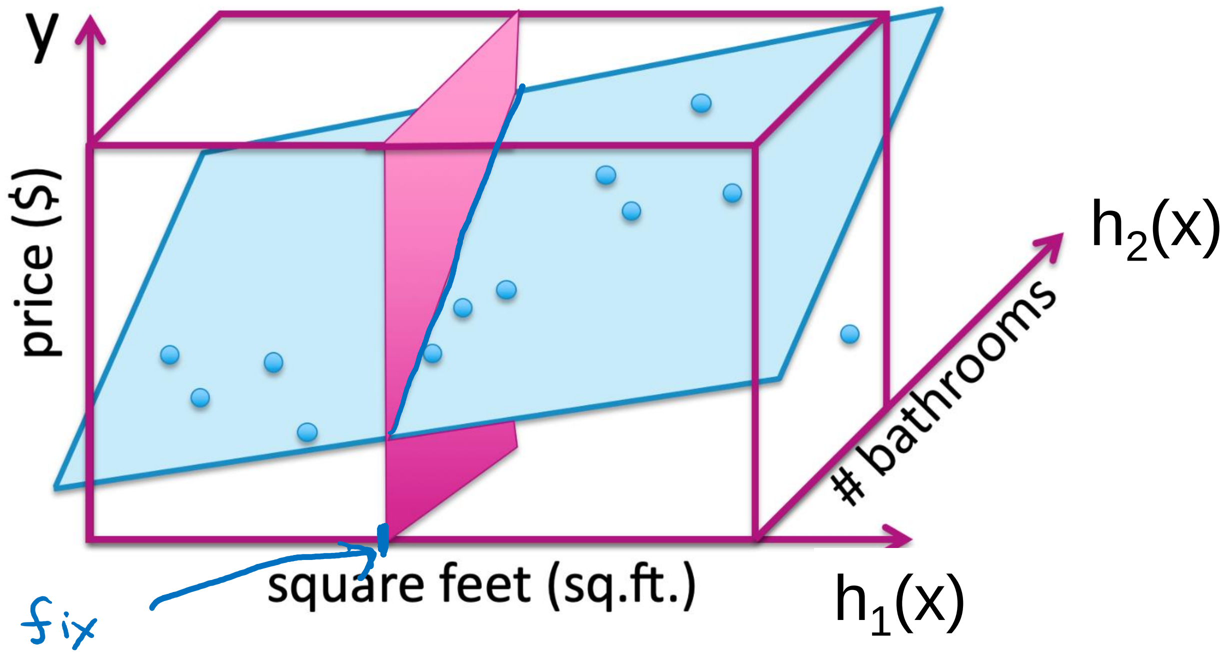
Cross Validation

Interpreting Coefficients

Interpreting Coefficients – Multiple Linear Regression

$$\hat{y} = \hat{w}_0 + \hat{w}_1 h_1(x) + \hat{w}_2 h_2(x)$$

Fix



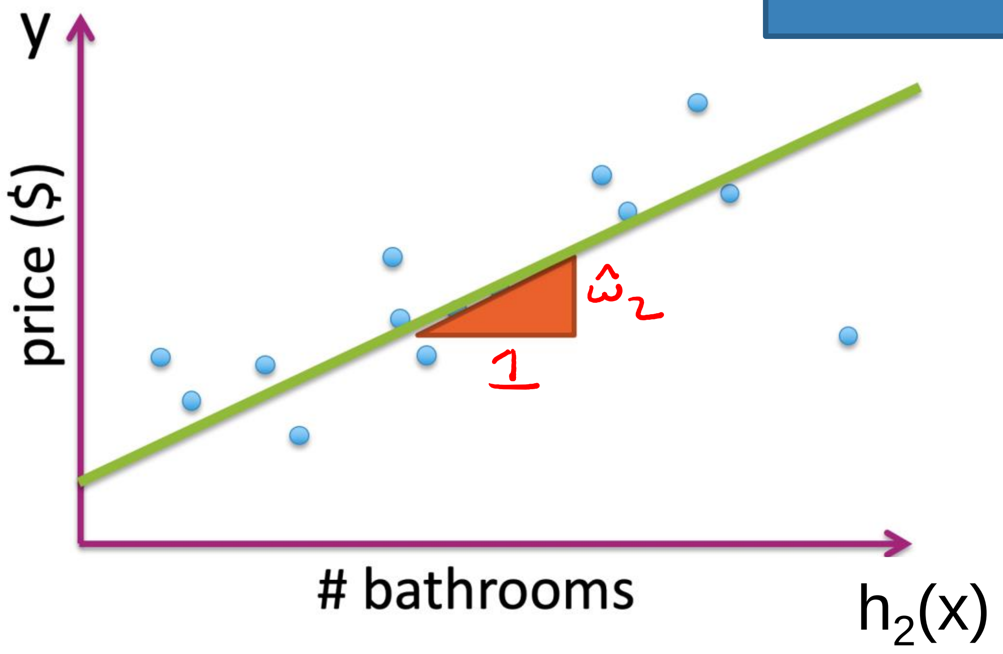
Interpreting Coefficients

Interpreting Coefficients – Multiple Linear Regression

$$\hat{y} = \hat{w}_0 + \hat{w}_1 h_1(x) + \hat{w}_2 h_2(x)$$

Fix

Holding $h_1(x)$ fixed!



Interpreting Coefficients

This also extends for multiple regression with many features!

$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{D} \hat{w}_j h_j(x)$$

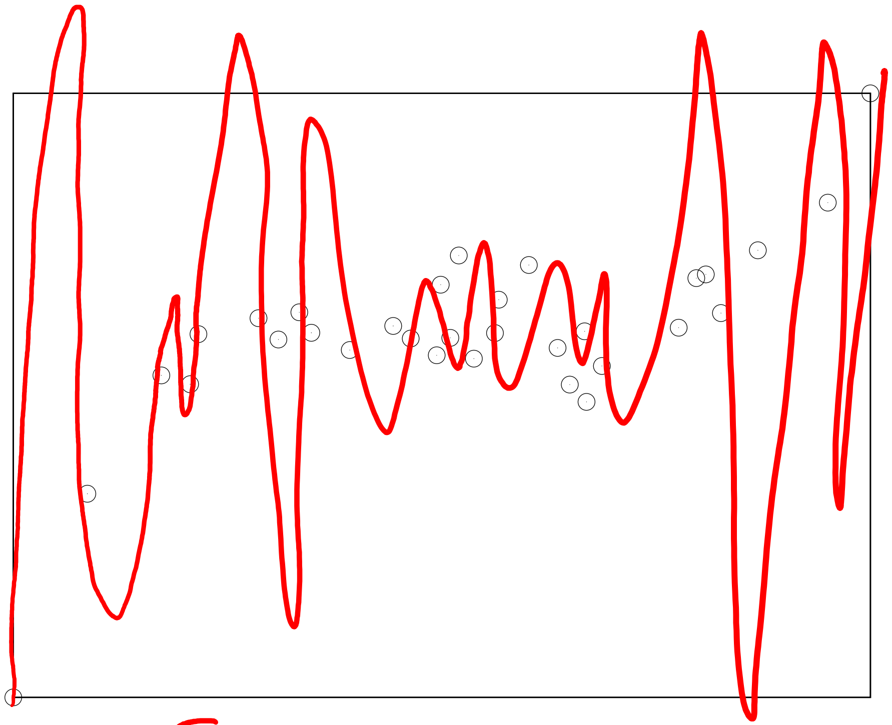
Interpret $\hat{w}_j$ as the change in y per unit change in $h_j(x)$ if all other features are held constant.

→ increase of 1

This is generally not possible for polynomial regression or if other features use same data input!

- Can't "fix" other features if they are derived from same input.

Overfitting



$\hat{w} = [\hat{w}_1, \hat{w}_2, \ldots \hat{w}_j, \ldots \hat{w}_D]$

Often, overfitting is associated with very large estimated parameters $\hat{w}$!

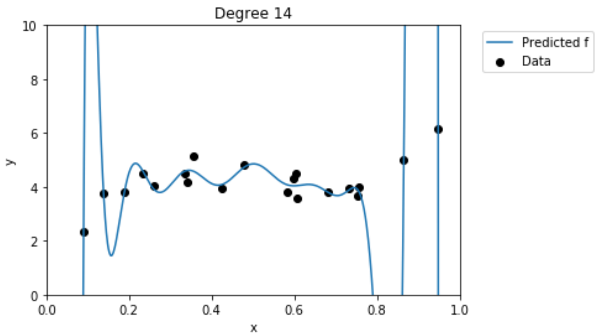
$|\hat{w}_j| \gg 0$

Number of Features

Overfitting is not limited to polynomial regression of large degree. It can also happen if you use a large number of features!

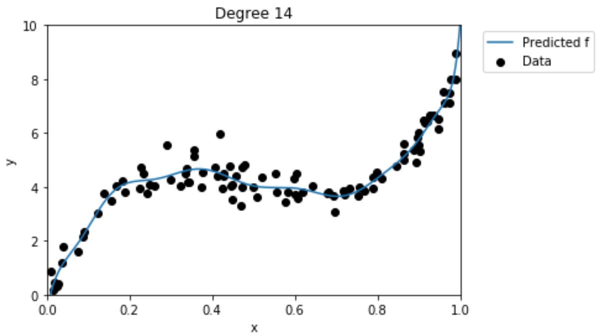
Why? Overfitting depends on whether the amount of data you have is large enough to represent the true function's complexity.

large $|\hat{w}_j|$



≈ 20 pts

moderate $|\hat{w}_j|$



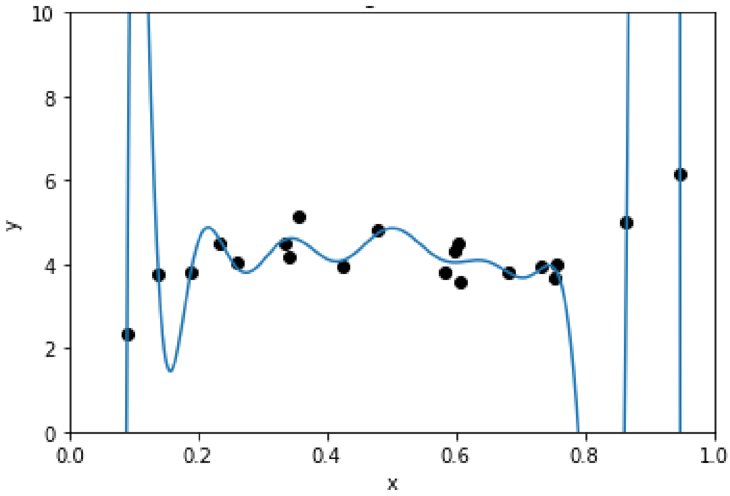
≈ 100 pts

Number of Features

How do the number of features affect overfitting?

1 feature

Data must include representative example of all $(h_1(x), y)$ pairs to avoid overfitting

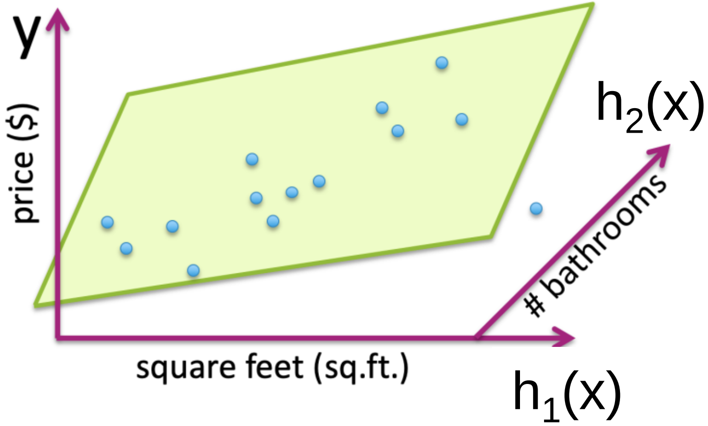


Number of Features

How do the number of features affect overfitting?

D features

Data must include representative example of all $((h_1(x), h_2(x), \ldots, h_D(x)), y)$ combos to avoid overfitting!



Introduction to the **Curse of Dimensionality**.
We will come back to this later in the quarter!

Prevent Overfitting

Last time, we **trained multiple models**, using cross validation / validation set, to find one that was less likely to overfit

- For selecting polynomial degree, we train p models.
- For selecting which features to include, we'd have to train ____ models!

next lecture ?

Can we **train one model** that isn't prone to overfitting in the first place?

- **Big Idea**: Have the model self-regulate to prevent overfitting by making sure its coefficients don't get "too large"

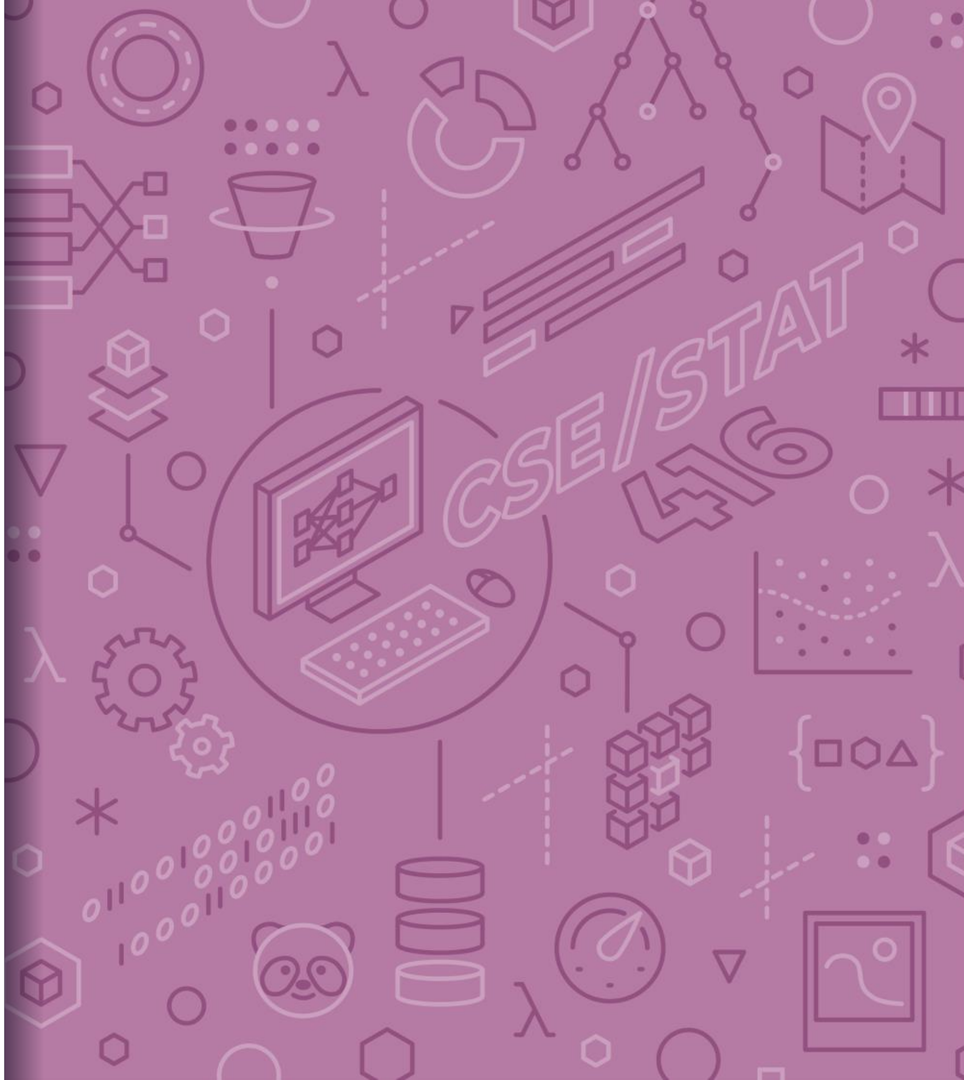
This idea is called **regularization**.

CSE/STAT 416

Cross Validation; Ridge Regression

Hunter Schafer
University of Washington
April 3, 2023

❓ **Questions?** Raise hand or **sli.do #cs416**
💬 **Before Class:** What is your favorite coffee/tea/boba/beverage shop near campus?
🎵 **Listening to:** [Sammy Rae & The Friends]



HW1 Walkthrough

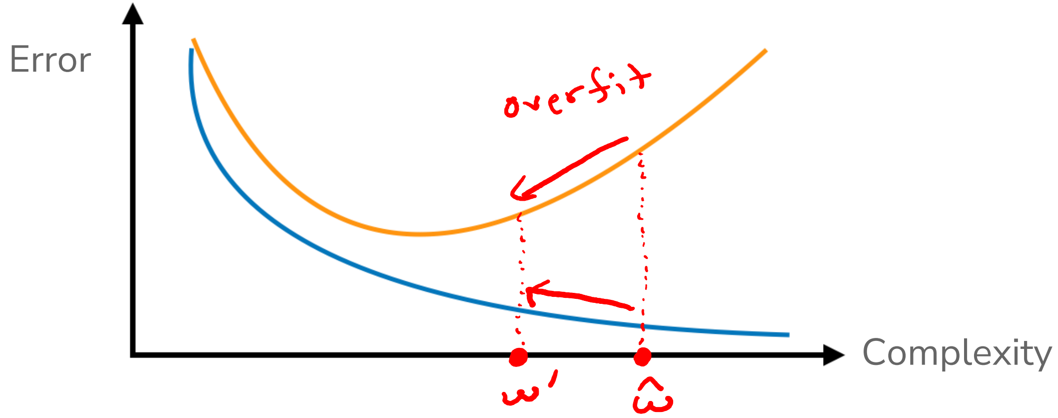
Recap

Overfitting

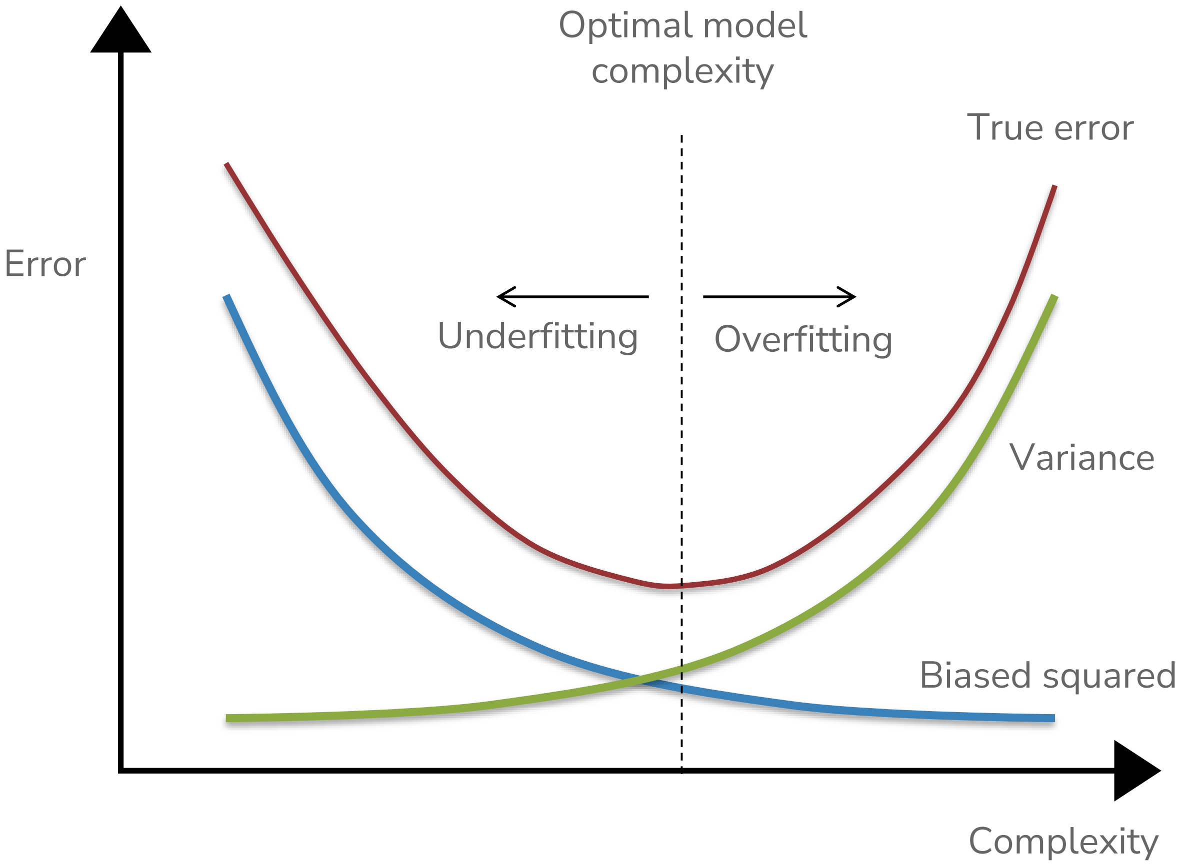
Overfitting happens when we too closely match the training data and fail to generalize.

Overfitting occurs when you train a predictor $\hat{w}$ but there exists another predictor w' from the same model class such that:

- $error_{true}(w') < error_{true}(\hat{w})$
- $error_{train}(w') > error_{train}(\hat{w})$



Bias – Variance Tradeoff

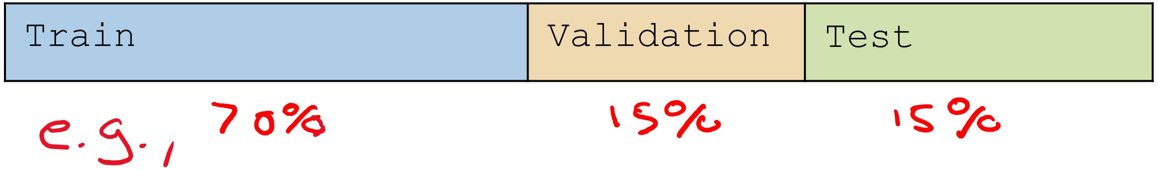


Validation Set

Important: this should be randomized!!!

So far we have divided our dataset into train and test

We can't use Test to choose our model complexity, so instead, break up Train into ANOTHER dataset



We will pick the model that does best on validation. Note that this now makes the validation error of the "best" model a biased estimate of true error. The test error will be an unbiased estimate though since we never looked at it!

Validation Set

The process generally goes

```
train, validation, test = random_split(dataset)
for each model complexity p:
    model = train_model(model_p, train)
    val_err = error(model, validation)
    keep track of p and model with smallest val_err
return best p & error(model, test)
```

any hyper-parameter

Validation Set

Pros

Easy to describe and implement

Pretty fast

- Only requires training a model and predicting on the validation set for each complexity of interest

Cons

- Have to sacrifice even more training data
- Prone to overfitting*

Cross Validation

For a set of hyperparameters, perform Cross Validation on k folds

Cross-Validation

The process generally goes

```
chunk_1, …, chunk_k, test = random_split(dataset)
for each model complexity p:
    for i in [1, k]:
        model = train_model(model_p, chunks - i)
        val_err = error(model, chunk_i)
    avg_val_err = average val_err over chunks
    keep track of p with smallest avg_val_err
return model trained on train (all chunks) with
best p & error(model, test)
```

iterate over hyperparameter settings

Interpretation: average validation error of models with complexity p.

Cross-Validation

Pros

- Prevent overfitting: By training the model on multiple folds instead of only 1 training set, this learns the model with the best generalization capabilities.
- Don't have to actually get rid of any training data!

Cons

- Slow. For each model selection, we have to train k times
- Very computationally expensive

} these go hand-in-hand

slido

Think

1 min

sli.do #cs416

Say we are testing p different polynomial degrees, using the pseudocode for k-fold cross-validation.

How many models would we train?

a) pk

b) $p(k-1)$

c) p^k

d) $pk+1$

```
chunk_1, …, chunk_k, test = random_split(dataset)
for each model complexity p:
    for i in [1, k]:
        model = train_model(model_p, chunks - i)
        val_err = error(model, chunk_i)
    avg_val_err = average val_err over chunks
    keep track of p with smallest avg_val_err
return model trained on train (all chunks) with
best p & error(model, test)
```

pk + 1

nested loop

final model

p times

k times

slido

Group

1 min

sli.do #cs416

Say we are testing p different polynomial degrees, using the pseudocode for k-fold cross-validation.

How many models would we train?

a) pk

b) $p(k-1)$

c) p^k

d) $pk+1$

```
chunk_1, …, chunk_k, test = random_split(dataset)
for each model complexity p:
    for i in [1, k]:
        model = train_model(model_p, chunks - i)
        val_err = error(model, chunk_i)
    avg_val_err = average val_err over chunks
    keep track of p with smallest avg_val_err
return model trained on train (all chunks) with
best p & error(model, test)
```

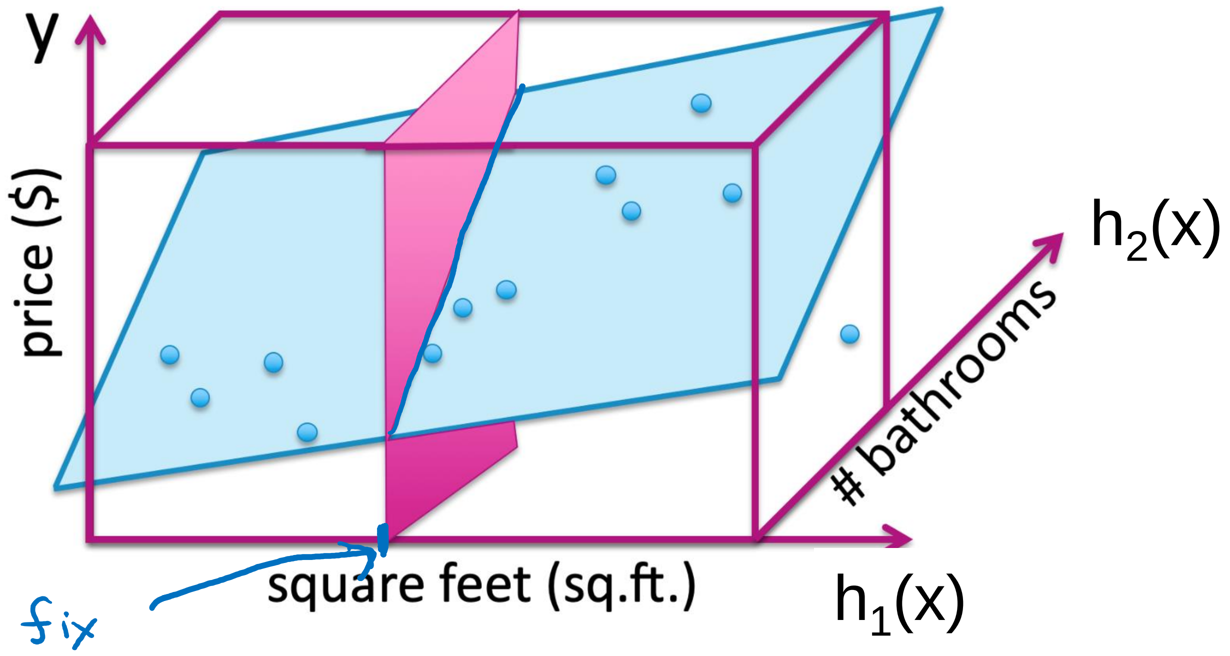
Coefficients and Overfitting

Interpreting Coefficients

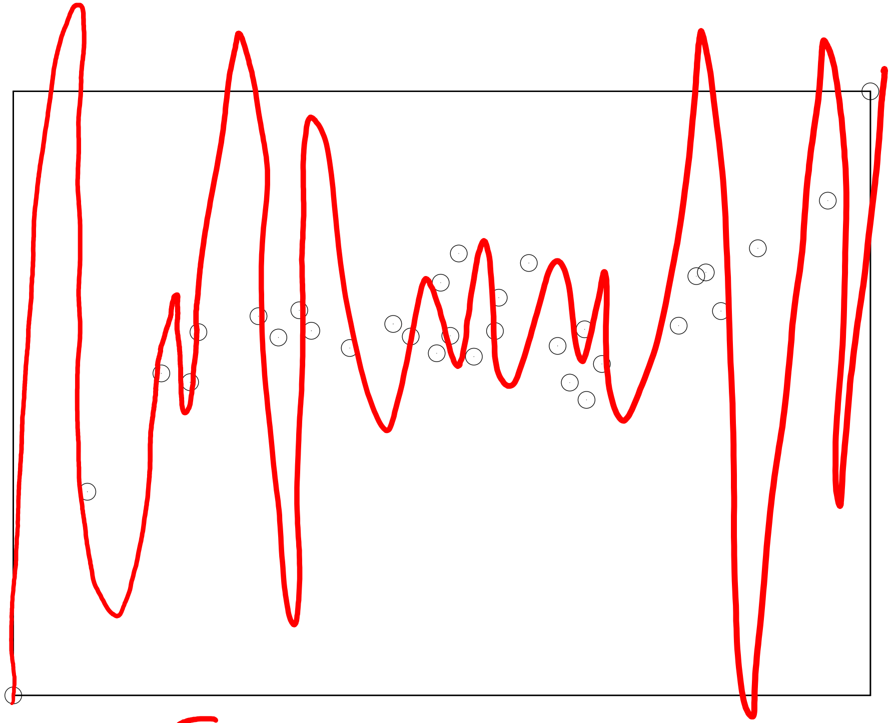
Interpreting Coefficients – Multiple Linear Regression

$$\hat{y} = \hat{w}_0 + \hat{w}_1 h_1(x) + \hat{w}_2 h_2(x)$$

Fix



Overfitting



$$\hat{w} = [\hat{w}_1, \hat{w}_2, \dots \hat{w}_j, \dots \hat{w}_D]$$

Often, overfitting is associated with very large estimated parameters $\hat{w}$!

$$|\hat{w}_j| \gg 0$$

slido

Group

2 Minutes

sli.do #cs416

Not discussed during class

What characterizes overfitting?

- **(Low / High)** Train Error, **(Low / High)** Test Error
- **(Low / High)** Bias, **(Low / High)** Variance

In which scenario is it more likely for a model to overfit?

- **(Few / Many)** Features
- **(Few / Many)** Parameters
- **(Small / Large)** Polynomial Degree
- **(Small / Large)** Dataset

Prevent Overfitting

Last time, we **trained multiple models**, using cross validation / validation set, to find one that was less likely to overfit

- For selecting polynomial degree, we train p models.
- For selecting which features to include, we'd have to train ____ models!

next lecture ?

Can we **train one model** that isn't prone to overfitting in the first place?

- **Big Idea**: Have the model self-regulate to prevent overfitting by making sure its coefficients don't get "too large"

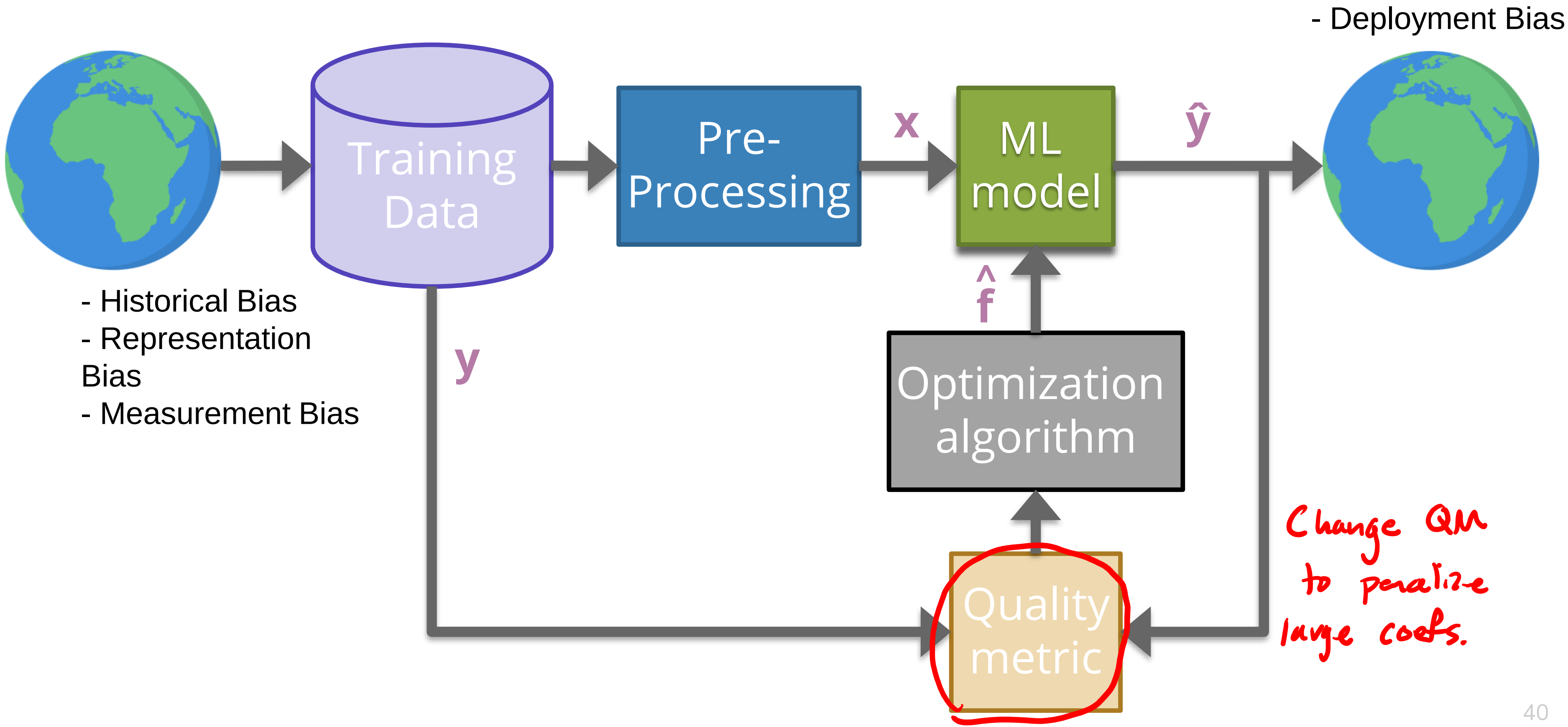
This idea is called **regularization**.

Regularization

Administrivia

- HW0 due Tuesday night at 11:59 pm on Ed and Gradescope
  - See late day policy if you need more time
- Office Hours (see schedule on website) started last week
  - So far, pretty sparsely attended so do make use of those
  - Longer wait times closer to assignment due dates
- Ed Discussion Board
  - Respond to other student's questions and in Megathread
  - Post privately if you're question is very detailed to your answer
  - Do not post solutions to assignments publicly
- Learning Reflection specifications are constant week-to-week, so you can start building LR1 now even if the turn in isn't open yet!

ML Pipeline


- Deployment Bias
Training Data
Pre-Processing
x
ML model
ŷ
f̂
y
Optimization algorithm
Quality metric
- Historical Bias
- Representation Bias
- Measurement Bias
Change QM to penalize large coefs.

$L(w) = MSE(w) = \frac{1}{n} \sum_{j=1}^{D} (y_i - \hat{f}(x_i))^2$

Regularization

Before, we used the quality metric that minimized loss

$$\hat{w} = \underset{w}{\operatorname{argmin}}\, L(w)$$

Change quality metric to balance loss with measure of overfitting

- $L(w)$ is the measure of fit
- $R(w)$ measures the magnitude of coefficients

$$\hat{w} = \underset{w}{\operatorname{argmin}}\, L(w) + \lambda\, R(w)$$

controls how much we care about magnitudes

λ: regularization parameter

How do we actually measure the magnitude of coefficients?

Magnitude

$w = [w_1, \dots, w_D]$ $R(w) =$ measure of overfitting

Come up with some number that summarizes the magnitude of the coefficients in w.

✗ **Sum?**

$R(w) = \sum_{j=1}^{D} w_j$

$w = [-1000, 1000]$

$R(w) = 0$

✓ **Sum of absolute values?**

$R(w) = \sum_{j=1}^{D} |w_j| = ||w||_1$

L_1 norm (LASSO)
↳ discussed Wed.

✓ **Sum of squares?**

$R(w) = \sum_{j=1}^{D} w_j^2 = ||w||_2^2$

L_2 norm (Ridge)
↳ today

p-norm: $||w||_p^p = \sum_{j=1}^{D} |w_j|^p$

Ridge Regression

Change quality metric to minimize

$$\hat{w} = \underset{w}{\operatorname{argmin}}\ MSE(w) + \lambda \|w\|_2^2$$

λ is a tuning **hyperparameter** that changes how much the model cares about the regularization term.

What if $\lambda = 0$?

$\hat{w} = \underset{w}{\operatorname{argmin}}\ MSE(w)$

$= \hat{w}_{OLS}$

Ordinary Least Squares

(OLS)

What if $\lambda = \infty$?

Essentially $\hat{w} = \underset{w}{\operatorname{argmin}}\ \lambda \|w\|_2^2 \rightarrow \hat{w} = \vec{0}$

any $w_j \neq 0$ would incur $R(w) = \infty$

λ in between? $0 \leq \|\hat{w}_{ridge}\|_2^2 \leq \|\hat{w}_{OLS}\|_2^2$

Poll Everywhere

Think

1 Minutes

How does λ affect the bias and variance of the model? For each underlined section, select "Low" or "High" appropriately.

When $\lambda = 0$

The model has **(Low / High)** Bias and **(Low / High)** Variance.

When $\lambda = \infty$

The model has **(Low / High)** Bias and **(Low / High)** Variance.

Poll Everywhere

Group

2 Minutes

How does λ affect the bias and variance of the model? For each underlined section, select "Low" or "High" appropriately.

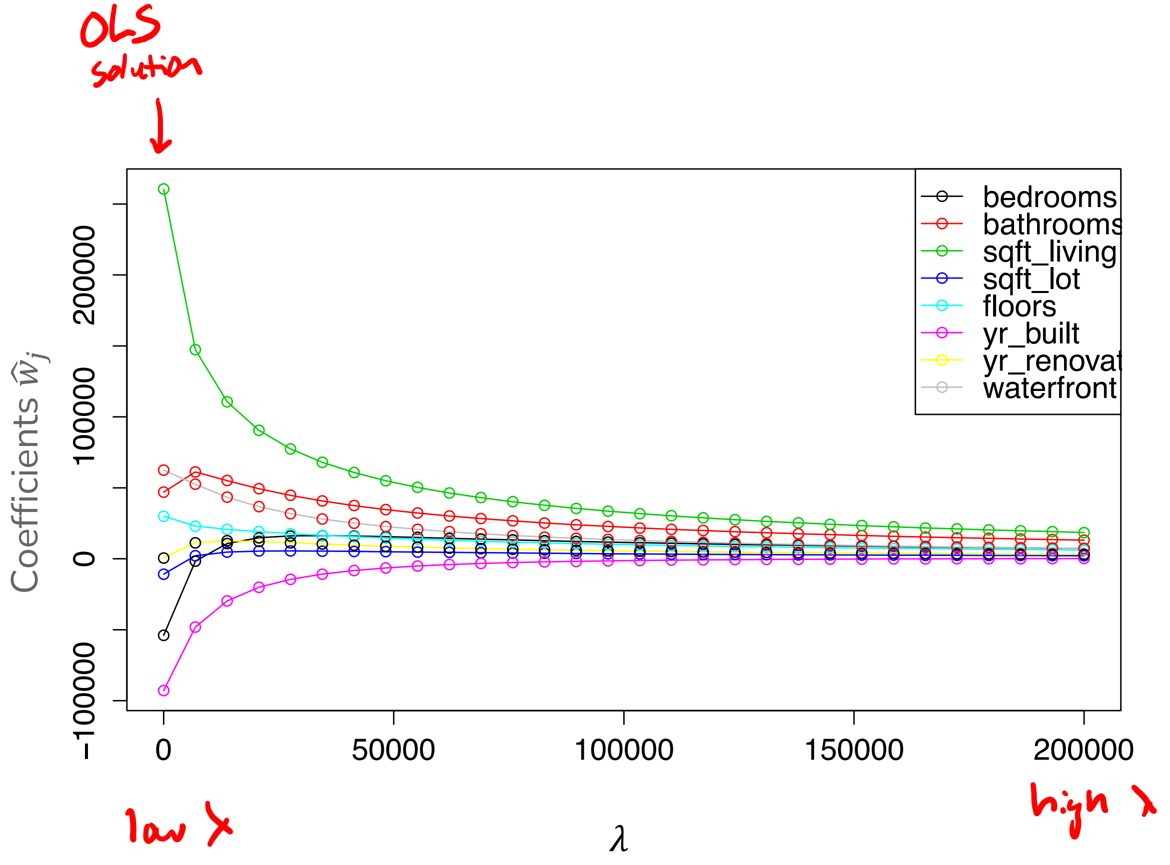
When $\lambda = 0$

The model has **(Low / High)** Bias and **(Low / High)** Variance.

When $\lambda = \infty$

The model has **(Low / High)** Bias and **(Low / High)** Variance.

Coefficient Paths



Brain Break

1:26

Demo: Ridge Regression

See Jupyter Notebook for interactive visualization.

Shows relationship between

- Regression line
- Mean Square Error
  - Also called Ordinary Least Squares
- Ridge Regression Quality Metric
- Coefficient Paths

Choosing λ

slido

Think

1 min

sli.do #cs416

How should we choose the best value of λ?

After we train each model with a certain λ_i and find

$\hat{w}_i = \text{argmin}_w \, MSE(w) + \lambda_i ||w||_2^2$:

overfit

a) Pick the λ_i that has the smallest $MSE(\hat{w}_i)$ on the **train set**

b) Pick the λ_i that has the smallest $MSE(\hat{w}_i)$ on the **validation set**

c) Pick the λ_i that has the smallest $MSE(\hat{w}_i) + \lambda_i ||\hat{w}_i||_2^2$ on the **train set**

see next

d) Pick the λ_i that has the smallest $MSE(\hat{w}_i) + \lambda_i ||\hat{w}_i||_2^2$ on the **validation set**

e) None of the above

slido

Group

2 min

sli.do #cs416

How should we choose the best value of λ?

After we train each model with a certain λ_i and find

$\hat{w}_i = \text{argmin}_w \, MSE(w) + \lambda_i ||w||_2^2$:

a) Pick the λ_i that has the smallest $MSE(\hat{w}_i)$ on the **train set**

b) Pick the λ_i that has the smallest $MSE(\hat{w}_i)$ on the **validation set**

c) Pick the λ_i that has the smallest $MSE(\hat{w}_i) + \lambda_i ||\hat{w}_i||_2^2$ on the **train set**

d) Pick the λ_i that has the smallest $MSE(\hat{w}_i) + \lambda_i ||\hat{w}_i||_2^2$ on the **validation set**

e) None of the above

Choosing λ

For any particular setting of λ, use Ridge Regression objective to train.

$$\hat{w}_{ridge} = \underset{w}{\operatorname{argmin}}\, MSE(w) + \lambda ||w||_2^2$$

If λ is too small, will overfit to **training set**. Too large, $\hat{w}_{ridge} = 0$.

How do we choose the right value of λ? We want the one that will do best on **future data.** Hence, we use the validation set.

For future data, what matters is that the model gets accurate predictions.

- $MSE(w)$ measures error of predictions ☆
- $MSE(w) + \lambda ||w||_2^2$ measures error of predictions & coefficient size
  used for training QM

☆ Regularization is a tool **used during training** to get a model that is likely to generalize. Regularization is **not used during prediction**. ☆

Choosing λ

The process for selecting λ is exactly the same as we saw with using a validation set or using cross validation.

for λ in λs:

Train a model using using Gradient Descent

$$\hat{w}_{ridge(\lambda)} = \underset{w}{\operatorname{argmin}}\, MSE_{train}(w) + \lambda||w||_2^2$$

Compute validation error

$$validation\_error = MSE_{val}(\hat{w}_{ridge(\lambda)})$$

Track λ with smallest $validation\_error$

Return λ^* & estimated future error $MSE_{test}(\hat{w}_{ridge(\lambda^*)})$

slido

Group

2 minutes

Not discussed in class

A model **parameter** is learnt during training (e.g., $\hat{w}$)

A **hyperparameter** is a parameter that is external to the model, whose value is used to influence the learning process.

What hyperparameters have we learned so far?

Some examples
- Which features/transformations to use
  - E.g., degree of polynomial
- Learning rate in Gradient Descent
- λ for regularization
- # folds for cross validation

Scaling

Regularization

At this point, I've hopefully convinced you that regularizing coefficient magnitudes is a good thing to avoid overfitting!

You:

We might have gotten a bit carried away, it doesn't ALWAYS make sense...

The Intercept

For most of the features, looking for large coefficients makes sense to spot overfitting. The one it does not make sense for is the **intercept**.

We shouldn't penalize the model for having a higher intercept since that just means the y value units might be really high! Also, the intercept doesn't affect the curvature of a loss function (it's just a linear scale).

- My demo before does this wrong and penalizes w_0 as well!

Two ways of dealing with this

- Center the y values so they have mean 0
  - This means forcing w_0 to be small isn't a problem
- Change the measure of overfitting to not include the intercept

$$\underset{w_0, w_{rest}}{\operatorname{argmin}} \underbrace{MSE(w_0, w_{rest})}_{\text{all params}} + \lambda \underbrace{\left|\left|w_{rest}\right|\right|_2^2}_{\text{all but intercept}}$$

Other Coefficients

- The L2 penalty penalizes all (non-intercept) coefficients equally
- Is that reasonable?

slido

Think

1 Minute

sli.do #cs416

How would the coefficient change if we change the scale of our feature?

Consider our housing example with $(sq.ft., price)$ of houses

- Say we learned a coefficient $\hat{w}_1$ for that feature
- What happens if we change the unit of x to square **miles?** Would $\hat{w}_1$ need to change?

a) The $\hat{w}_1$ in the new model with sq. miles would be larger

b) The $\hat{w}_1$ in the new model with sq. miles would be smaller

c) The $\hat{w}_1$ in the new model with sq. miles would stay the same

coeff = slope = rise/run

If features are on a smaller scale, coeff needs to go up to get the same "rise" for smaller "run"

slido

Group

1 Minute

sli.do #cs416

How would the coefficient change if we change the scale of our feature?

Consider our housing example with $(sq.ft., price)$ of houses

- Say we learned a coefficient $\hat{w}_1$ for that feature
- What happens if we change the unit of x to square **miles?** Would $\hat{w}_1$ need to change?

a) The $\hat{w}_1$ in the new model with sq. miles would be larger

b) The $\hat{w}_1$ in the new model with sq. miles would be smaller

c) The $\hat{w}_1$ in the new model with sq. miles would stay the same

Scaling Features

The other problem we overlooked is the "scale" of the coefficients.

Remember, the coefficient for a feature increase per unit change in that feature (holding all others fixed in multiple regression)

Consider our housing example with $(sq.ft., price)$ of houses

- Say we learned a coefficient $\hat{w}_1$ for that feature
- What happens if we change the unit of x to square **miles?** Would $\hat{w}_1$ need to change?
  - It would need to get bigger since the prices are the same but its inputs are smaller

This means we accidentally penalize features for having large coefficients due to having small value inputs!

Scaling Features

Fix this by **normalizing** the features so all are on the same scale!

$$\tilde{h}_j(x_i) = \frac{h_j(x_i) - \mu_j(x_1, \dots, x_N)}{\sigma_j(x_1, \dots, x_N)}$$

Where

The mean of feature j:

$$\mu_j(x_1, \dots, x_N) = \frac{1}{N}\sum_{i=1}^{N} h_j(x_i)$$

The standard devation of feature j:

$$\sigma_j(x_1, \dots, x_N) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(h_j(x_i) - \mu_j(x_1, \dots, x_N)\right)^2}$$

Important: Must scale the test data and all future data using the means and standard deviations **of the training set!**

- Otherwise the units of the model and the units of the data are not comparable!

Recap

Theme: Use regularization to prevent overfitting

Ideas:

- How to interpret coefficients
- How overfitting is affected by number of data points
- Overfitting affecting coefficients
- Use regularization to prevent overfitting
- How L2 penalty affects learned coefficients
- Visualizing what regression is doing
- Practicalities: Dealing with intercepts and feature scaling