CSE/STAT 416

Cross Validation; Ridge Regression

Pre-Class Videos (lecture slides below)

Tanmay Shah
University of Washington
April 3, 2024

Bias-Variance Tradeoff
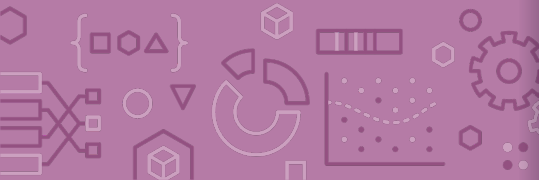
Tradeoff between bias and variance:

> Simple models: High bias + Low variance

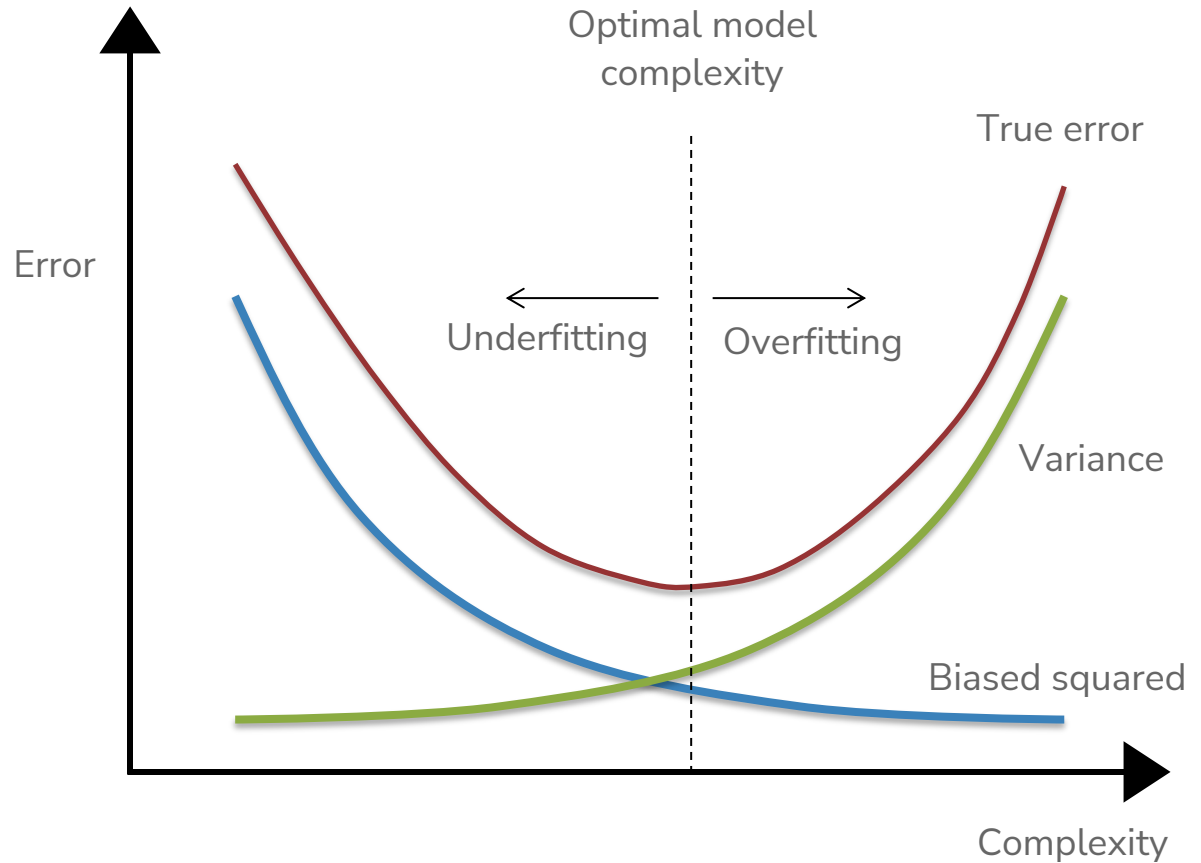> Complex models: Low bias + High variance

Source of errors for a particular model $\hat{f}$ using MSE loss function:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{bias}[\hat{f}(x)]^2 + \text{var}(\hat{f}(x)) + \sigma_\epsilon^2$$

Error = Biased squared + Variance + Irreducible Error

Bias – Variance Tradeoff

Optimal model complexity

True error

Error

← Underfitting | Overfitting →
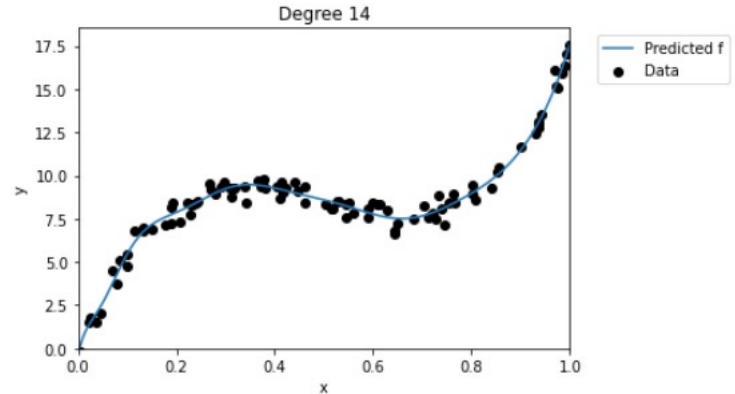
Variance

Biased squared

Complexity

Dataset Size

So far our entire discussion of error assumes a fixed amount of data. What happens to our error (true error and training error) as we get more data?

Error

Size of train set

Dataset Size

Model complexity doesn't depend on the size of the training set

The larger the training set, the lower the variance of the model, thus less overfitting
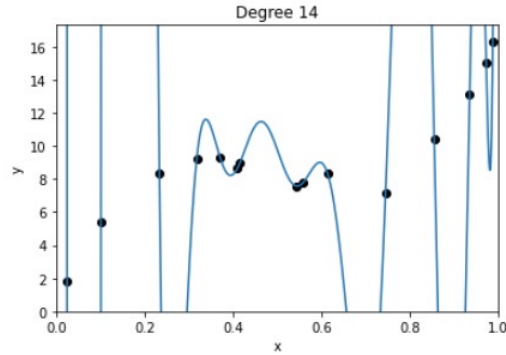
Demo

Bias-Variance Tradeoff

Training a linear regression model in Python

Observing the effect of the bias-variance tradeoff as compared to model complexity

Brain Break

Choosing Complexity

Choosing Complexity

So far we have talked about the affect of using different complexities on our error. Now, how do we choose the right one?

Think 👤

1 min

Suppose I wanted to figure out the right degree polynomial for my dataset (we'll try p from 1 to 20). What procedure should I use to do this? Pick the best option

For each possible degree polynomial p:

Train a model with degree p on the training set, pick p that has the lowest test error

Train a model with degree p on the training set, pick p that has the highest test error

Train a model with degree p on the test set, pick p that has the lowest test error

Train a model with degree p on the test set, pick p that has the highest test error

None of the above

Think 👤

2 min

Suppose I wanted to figure out the right degree polynomial for my dataset (we'll try p from 1 to 20). What procedure should I use to do this? Pick the best option

For each possible degree polynomial p:

Train a model with degree p on the training set, pick p that has the lowest test error

Train a model with degree p on the training set, pick p that has the highest test error

Train a model with degree p on the test set, pick p that has the lowest test error

Train a model with degree p on the test set, pick p that has the highest test error

None of the above

Choosing Complexity

We can't just choose the model that has the lowest **train** error because that will favor models that overfit!

It then seems like our only other choice is to choose the model that has the lowest **test** error (since that is our approximation of the true error)

- This is almost right. However, the test set has been **tampered**, thus is no longer is an unbiased estimate of the true error.

- We didn't technically train the model on the test set (that's good), but we chose **which model** to use based on the performance of the test set.

  - It's no longer a stand in for "the unknown" since we probed it many times to figure out which model would be best.

NEVER EVER EVER touch the test set until the end. You only use it ONCE to evaluate the performance of the best model you have selected during training.

Choosing Complexity

We will talk about two ways to pick the model complexity without ruining our test set.
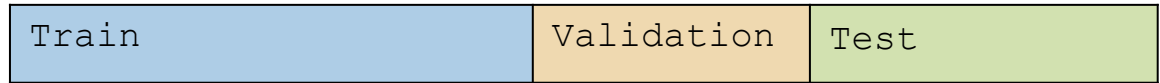
Using a validation set

Doing (k-fold) cross validation

Validation Set

So far we have divided our dataset into train and test

Train	Test

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

Train	Validation	Test

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)
```
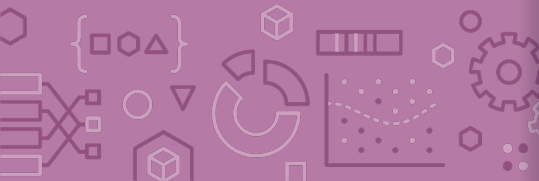
Validation Set

Pros

Easy to describe and implement

Pretty fast

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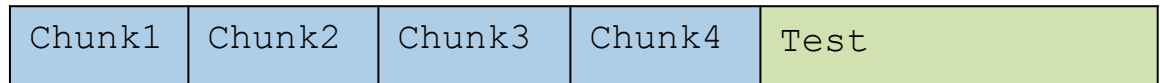
Cons

- Have to sacrifice even more training data

- Prone to overfitting*

Cross-Validation

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.

Train	Test

Chunk1	Chunk2	Chunk3	Chunk4	Test

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.
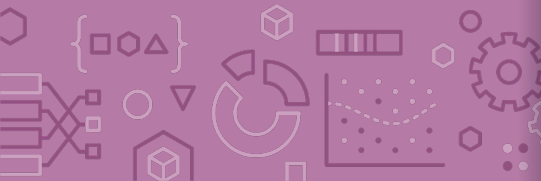
Cross Validation

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

Validation   **Training**

Error 1

Error 2

Error 3
.
.
Error k

Average all validation errors

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)
```
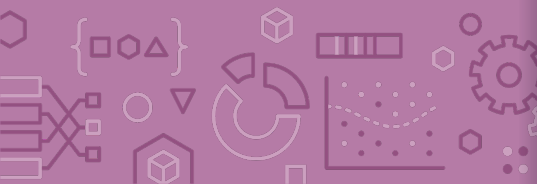
Cross-Validation

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

Cross-Validation

Generally, the more folds you use the better as you aren't relying on the specifics of a single validation fold.

Theoretical best estimator* is to use $k = n$

- Called "**Leave One Out Cross Validation**" (LOOCV)

In practice, people use $k = 5$ to 10 for computational simplicity

Recap

Theme: Assess the performance of our models

Ideas:

Model complexity

Train vs. Test vs. True error

Overfitting and Underfitting

Bias-Variance Tradeoff

Error as a function of train set size

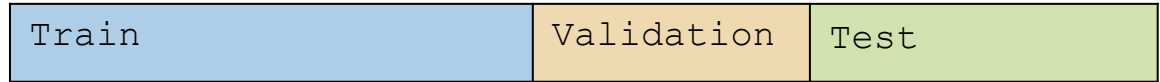Choosing best model complexity
- Validation set
- Cross Validation

Pre-Class Video 1:

Cross Validation

Validation Set

So far we have divided our dataset into train and test

Train	Test

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

Train	Validation	Test

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)
```
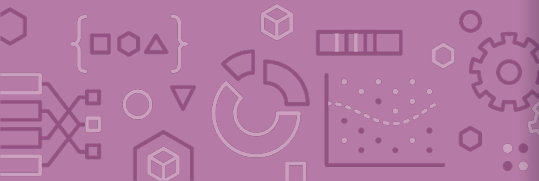
Validation Set

Pros

Easy to describe and implement

Pretty fast

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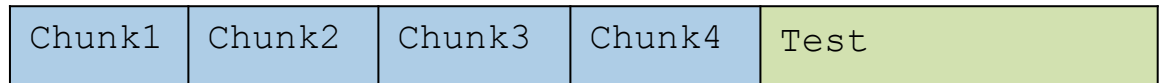
Cons

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

Cross-Validation

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.

Train	Test

Chunk1	Chunk2	Chunk3	Chunk4	Test

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

Validation   **Training**

Error 1

Error 2

Error 3
.
.
Error k

Average all validation errors

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)
```

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

Cross-Validation

What size of k?

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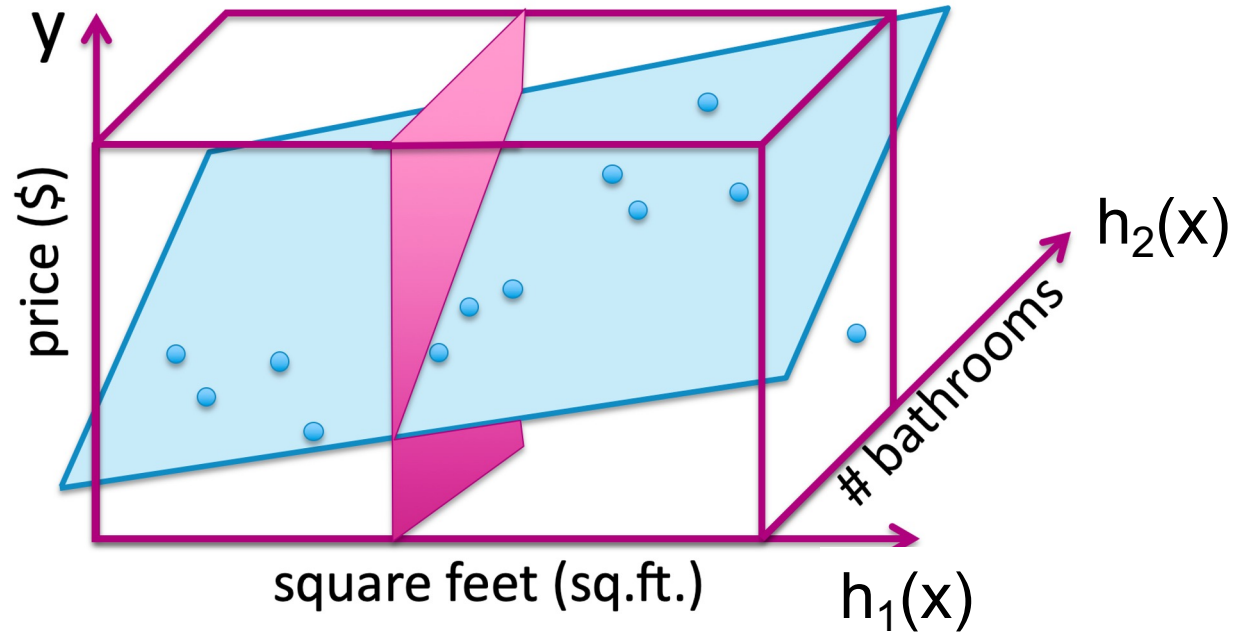
In practice, people use $k = 5$ to 10.

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

y

price ($)

square feet (sq.ft.)
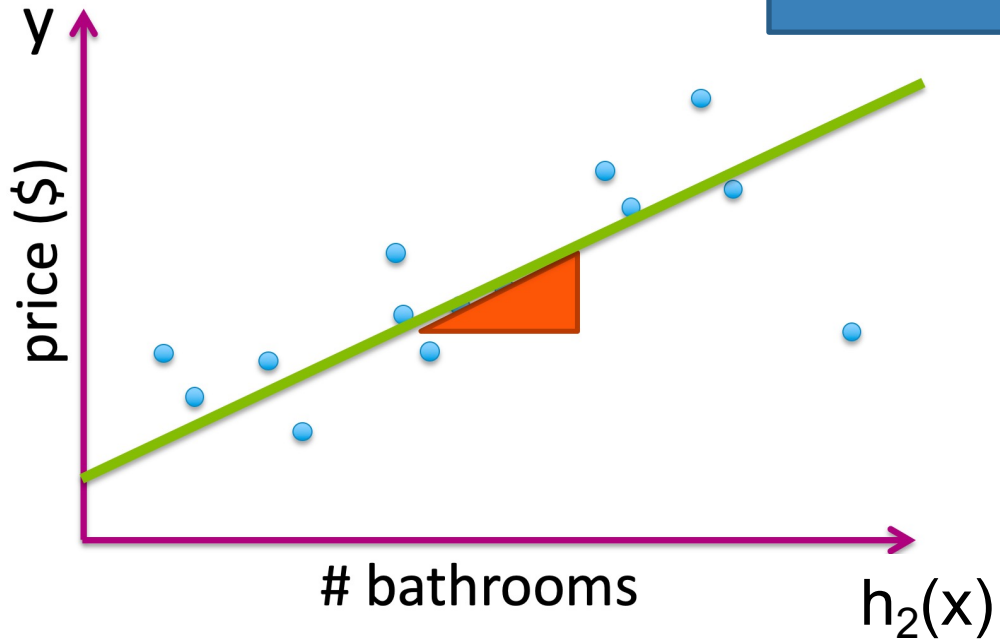
$h_1(x)$

$h_2(x)$

bathrooms

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!

y

price ($)

bathrooms

$h_2(x)$

Interpreting Coefficients

This also extends for multiple regression with many features!

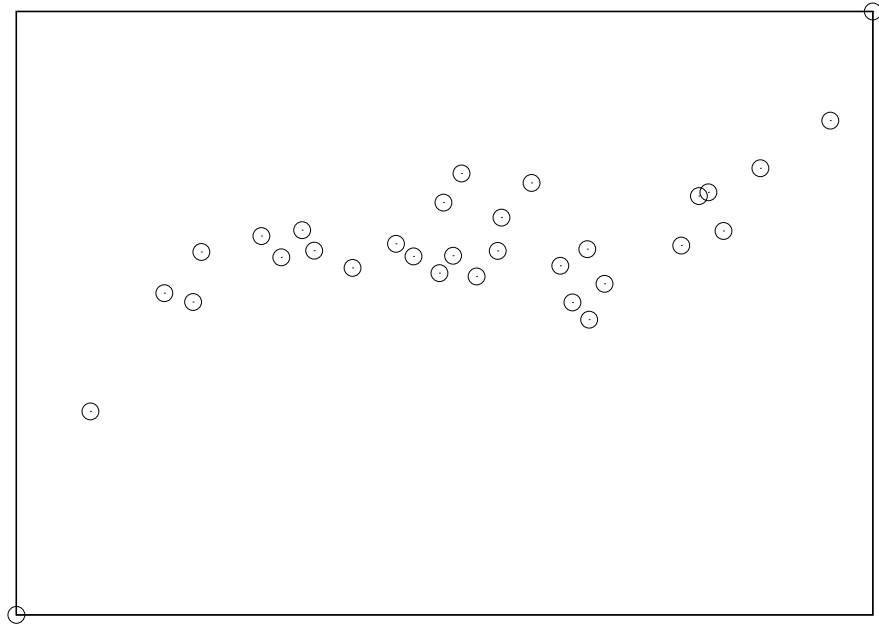$$\hat{y} = \widehat{w}_0 + \sum_{j=1}^{D} \widehat{w}_j h_j(x)$$

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

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.
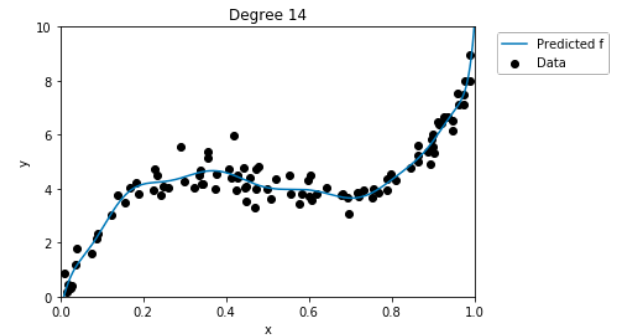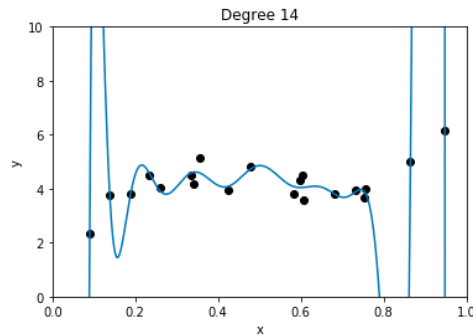
Overfitting

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

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.
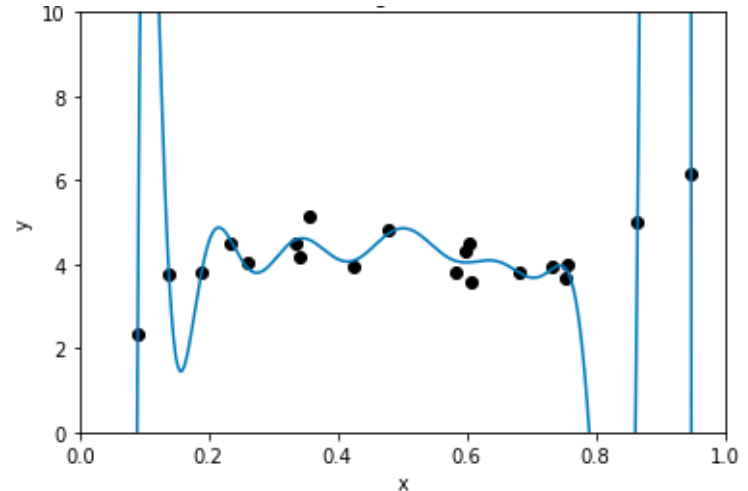
Number of Features

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
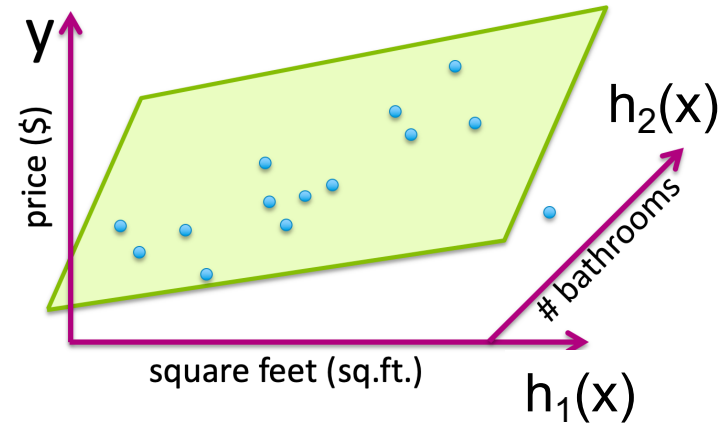
HARD

Number of Features

How do the number of features affect overfitting?

D features

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

MUCH HARDER!!

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
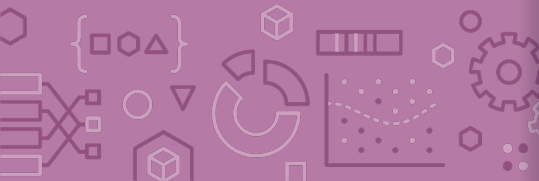
For selecting polynomial degree, we train p models.

For selecting which features to include, we'd have to train ____ models!

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

Tanmay Shah
University of Washington
April 3, 2024

❓ **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 $\widehat{w}$ but there exists another predictor w' from the same model class such that:

$$error_{true}(w') < error_{true}(\widehat{w})$$

$$error_{train}(w') > error_{train}(\widehat{w})$$

Bias – Variance Tradeoff

Error

Optimal model complexity

True error

Underfitting ← | → Overfitting

Variance

Biased squared

Complexity

45

Validation Set

So far we have divided our dataset into train and test

Train	Test

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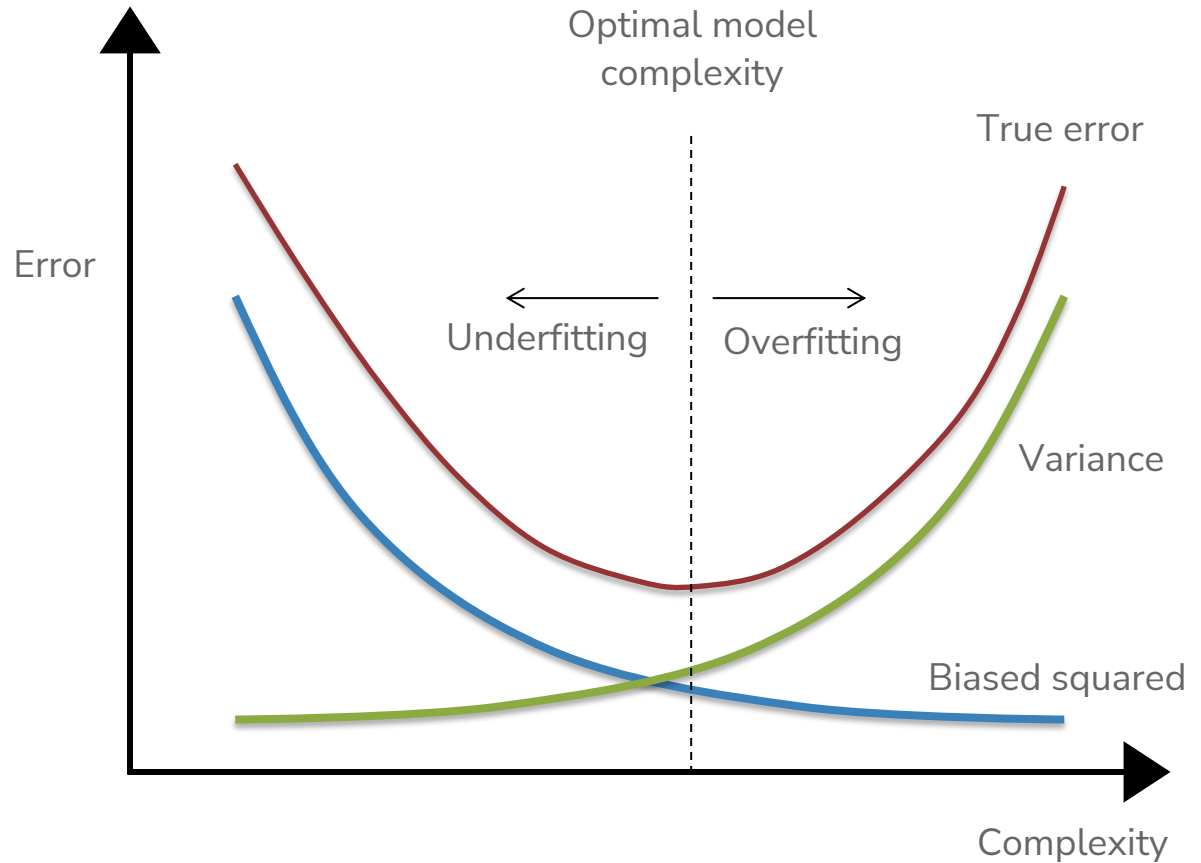
Train	Validation	Test

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)
```

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

Validation   **Training**

Error 1

Error 2

Error 3
.
.
Error k

Average all validation errors

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)
```

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

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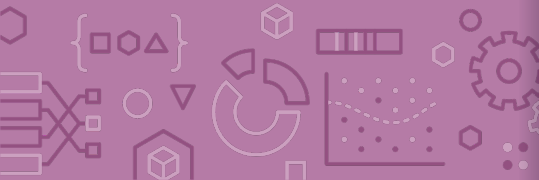
Say we are testing p different polynomial degrees, using the pseudocode for k-fold cross-validation.

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)
```
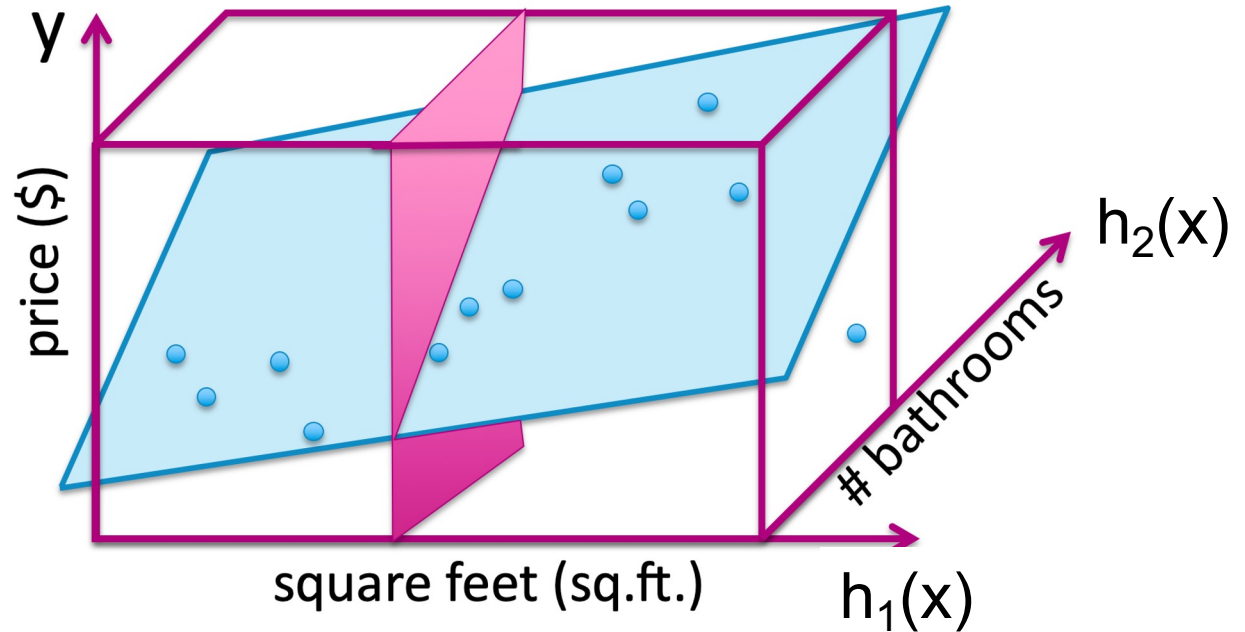
Coefficients and Overfitting

Interpreting Coefficients

Interpreting Coefficients – Multiple Linear Regression

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

Fix

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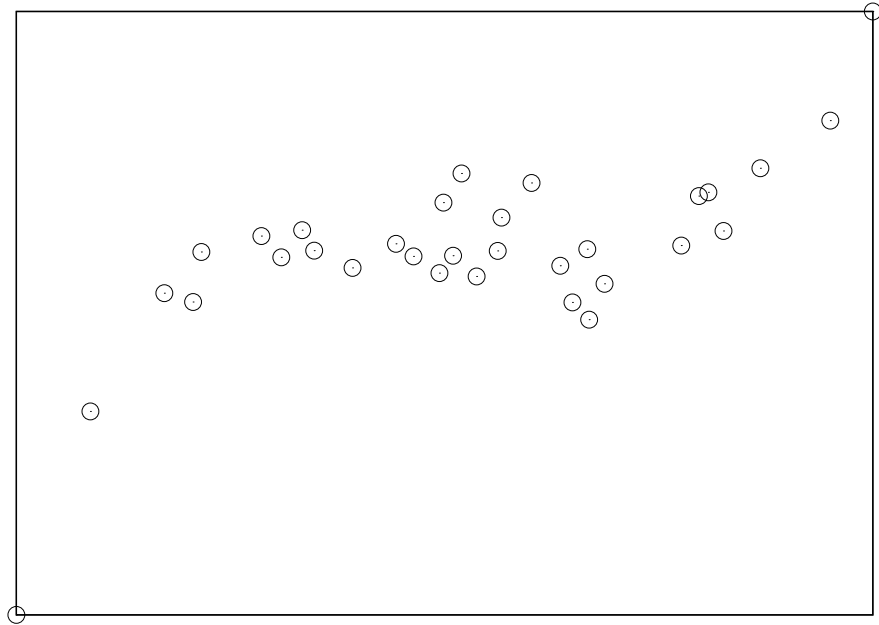
Often, overfitting is associated with very large estimated parameters $\hat{w}$!

slido

Group 👥

2 Minutes

sli.do #cs416

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
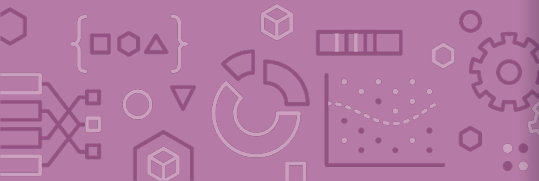
> For selecting polynomial degree, we train p models.

> For selecting which features to include, we'd have to train ____ models!

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 Thursday 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

- Historical Bias
- Representation Bias
- Measurement Bias

y

f̂

Optimization algorithm
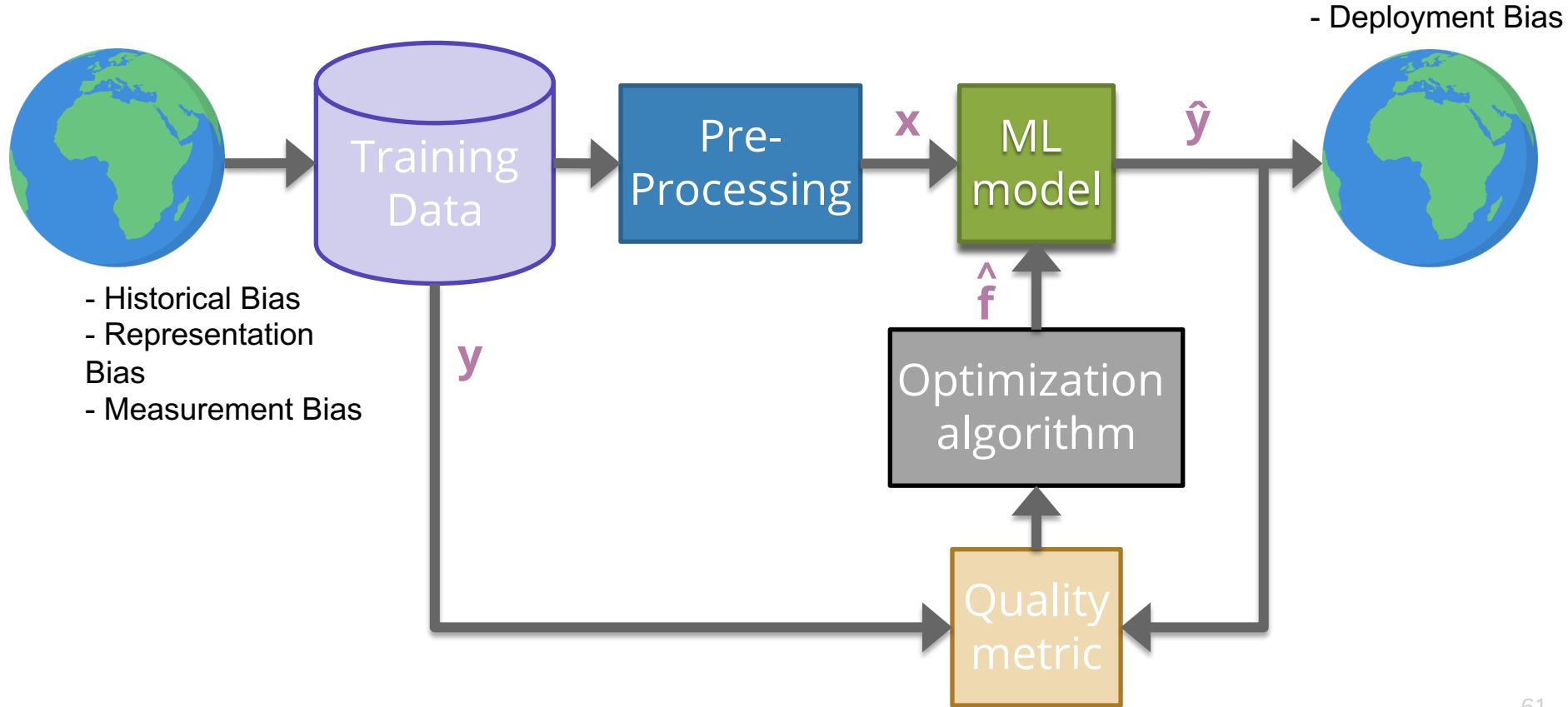
Quality metric

Regularization

Before, we used the quality metric that minimized loss

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

Change quality metric to balance loss with measure of overfitting

$L(w)$ is the measure of fit

$R(w)$ measures the magnitude of coefficients

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

λ: regularization parameter

How do we actually measure the magnitude of coefficients?

Magnitude

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

Sum?

Sum of absolute values?

Sum of squares?
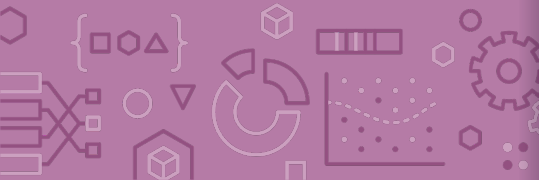
Ridge Regression

Change quality metric to minimize

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

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

What if $\lambda = 0$?

What if $\lambda = \infty$?

λ in between?

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.

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
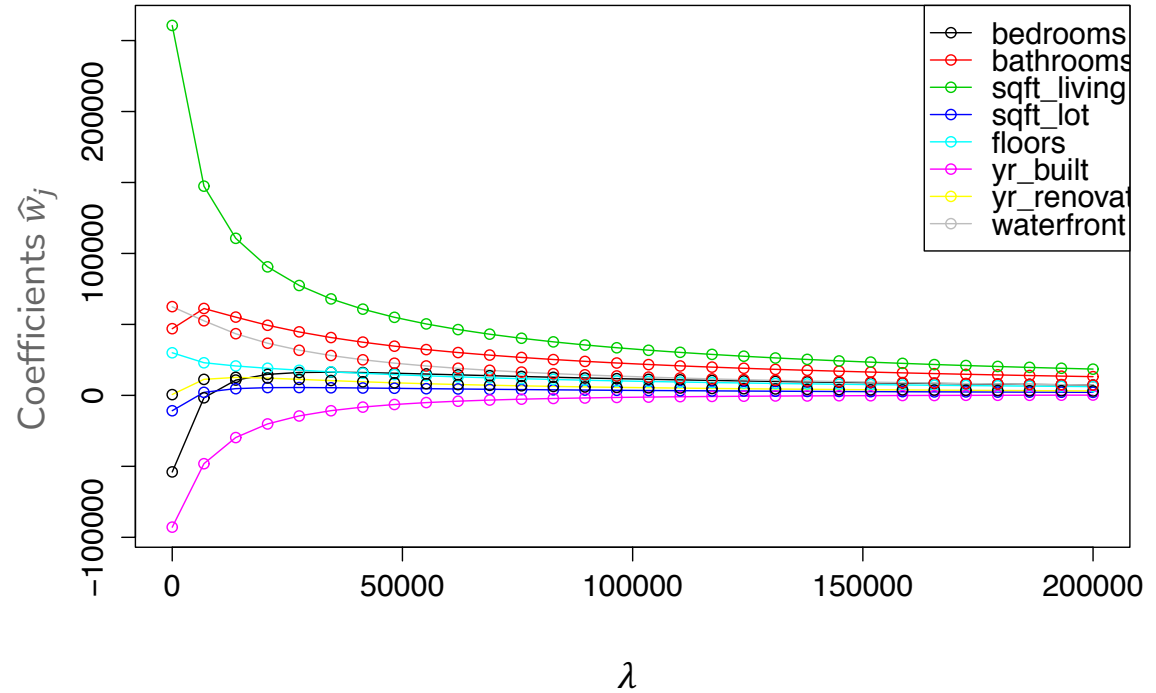
☕ Brain Break

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

How should we choose the best value of λ?

After we train each model with a certain λ_i and find

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

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

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

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

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

e)   None of the above

How should we choose the best value of λ?

After we train each model with a certain λ_i and find

$$\widehat{w}_i = \mathrm{argmin}_w \, MSE(w) + \lambda_i \left|\left|w\right|\right|_2^2:$$

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

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

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

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

e) None of the above

Choosing λ

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

$$\widehat{w}_{ridge} = \operatorname*{argmin}_{w} MSE(w) + \lambda ||w||_2^2$$

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

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

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:
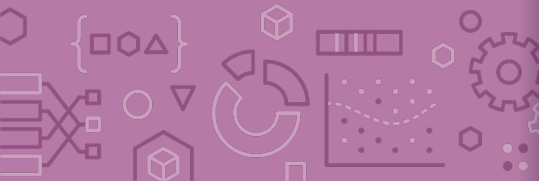
  Train a model using using Gradient Descent

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

  Compute validation error

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

  Track λ with smallest $validation\_error$

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

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?

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

$$\underset{w_0, w_{rest}}{\text{argmin}} \; MSE(w_0, w_{rest}) + \lambda ||w_{rest}||_2^2$$

Other Coefficients

The L2 penalty penalizes all (non-intercept) coefficients equally

Is that reasonable?

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 $\widehat{w}_1$ for that feature

What happens if we change the unit of x to square **miles?**
Would $\widehat{w}_1$ need to change?

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

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

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

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 $\widehat{w}_1$ for that feature

> What happens if we change the unit of x to square **miles?** Would $\widehat{w}_1$ need to change?

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

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

c) The $\widehat{w}_1$ 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 $\widehat{w}_1$ for that feature

What happens if we change the unit of x to square **miles?** Would $\widehat{w}_1$ 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!

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

Where

The mean of feature j:

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

The standard devation of feature j:

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

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