

Lecture 15

Cross Validation, Regularization

Different methods for ensuring the generalizability of our models to unseen data.

Today's Roadmap

Cross Validation

- **The Holdout Method**
- K-Fold Cross Validation
- Test Sets

Regularization

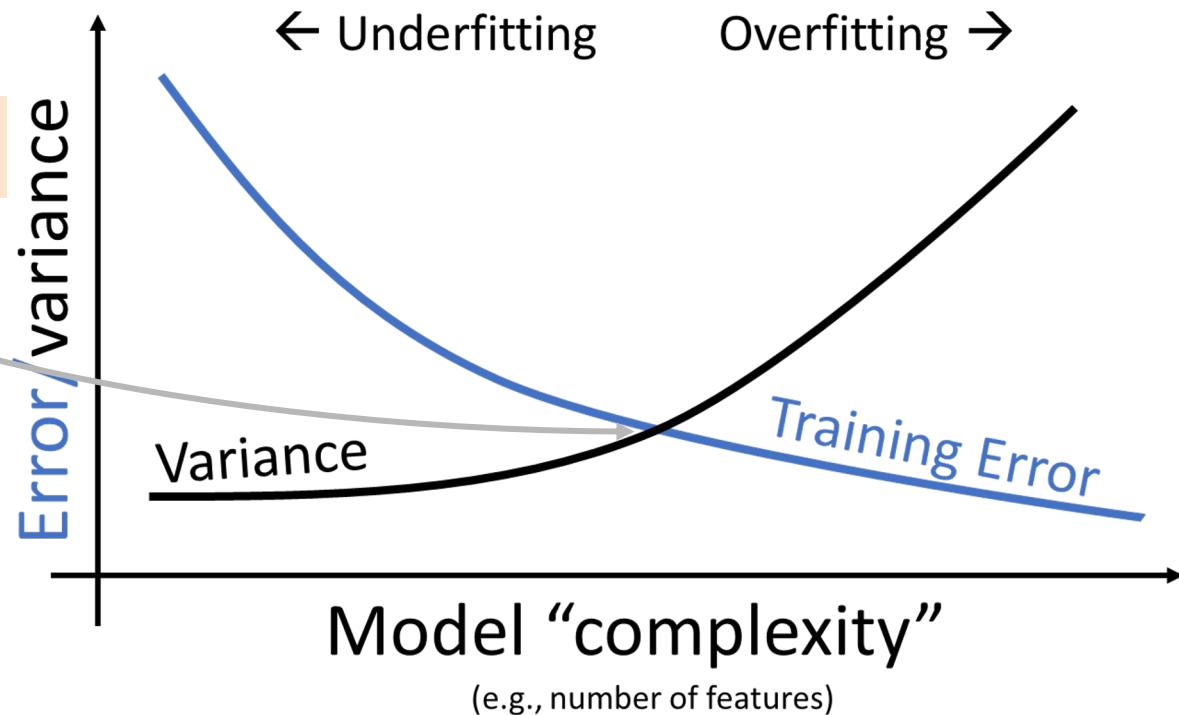
- L2 Regularization (Ridge)
- Scaling Data for Regularization
- L1 Regularization (LASSO)

Review: Error vs. Complexity

As we increase the complexity of our model:

- Training error decreases.
- Variance increases.

Our goal: find this “sweet spot”



In today's lecture, we will address both topics in detail.

First half of lecture: **Cross-Validation**

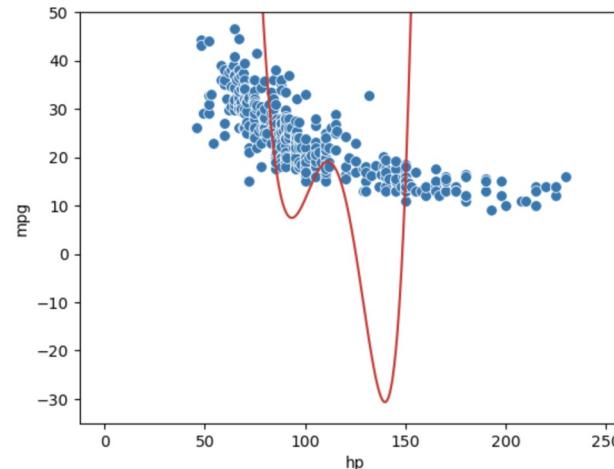
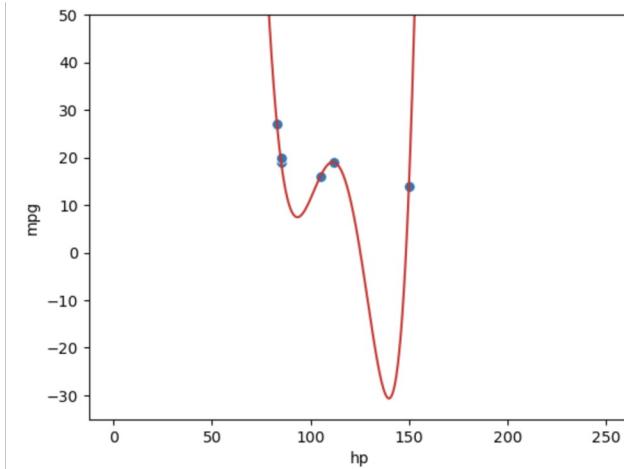
- Formalize the idea of training and test data.
- Introduce a method to "preview" how a model will perform on unseen data.

Second half of lecture: **Regularization**

- Introduce a method to finetune our model's complexity.

Where We Left Things

A complex model may not perform well on data it did not encounter during training.



How to quantify performance on this "unseen" data? Introduce a **test set**.

Test Sets

A **test set** is a portion of our dataset that we set aside for testing purposes.

- We do *not* consider the test set when fitting/training the model.
- The test set is only ever touched once: to compute the performance (MSE, RMSE, etc) of the model *after* all fine-tuning has been completed.

Our new workflow for modeling: First, perform a **train-test split** (see [documentation](#)). Consider only the training set when designing the model. Then, evaluate on the test set.



Validation Sets

What if we were dissatisfied with our test set performance?

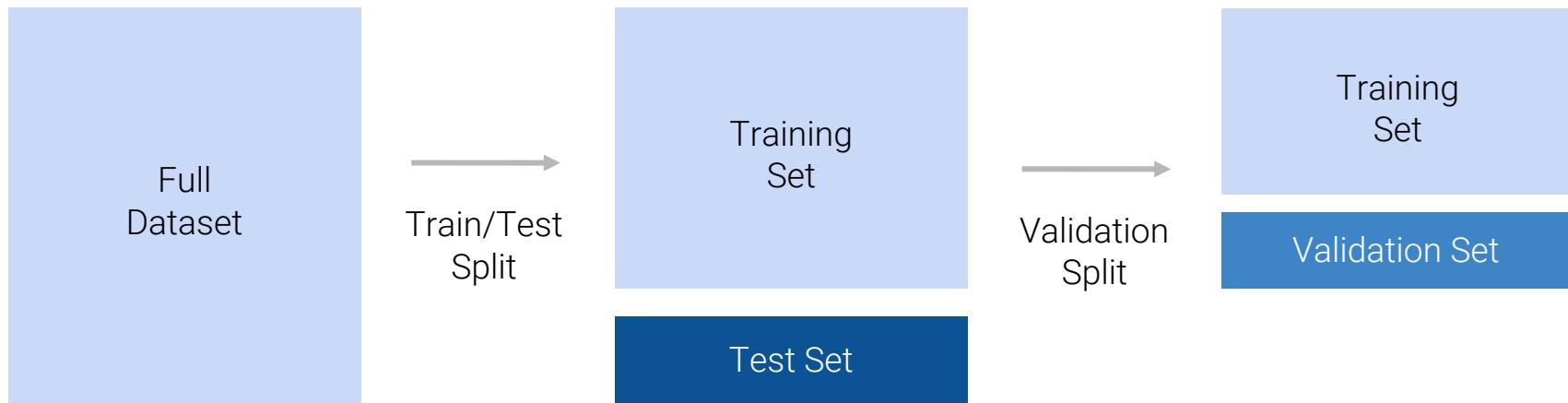
In our current framework, we'd be stuck – we can't then go back and adjust our model, because we'd be *factoring in information from the test set* to design our model. The test set would no longer represent performance on unseen data.

Solution: introduce a **validation set**.

Validation Sets

A **validation set** is a portion of our *training* set that we set aside for assessing model performance while it is *still being developed*.

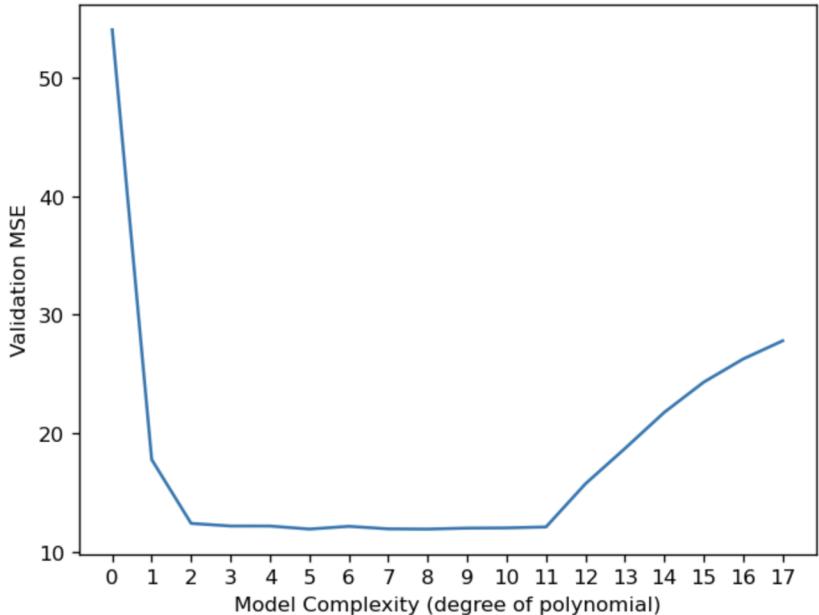
- Train model on the training set. Assess performance on the validation set. Adjust the model, then repeat.
- After *all* model development is complete, assess final performance on the test set.



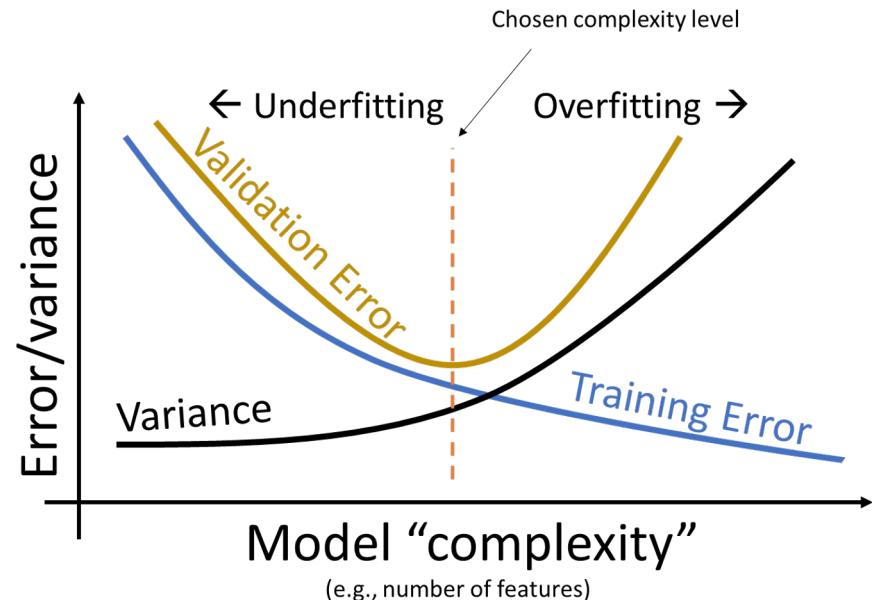
Updating Our Understanding of Model Complexity

Computing the validation error allows us to visualize under- and overfitting.

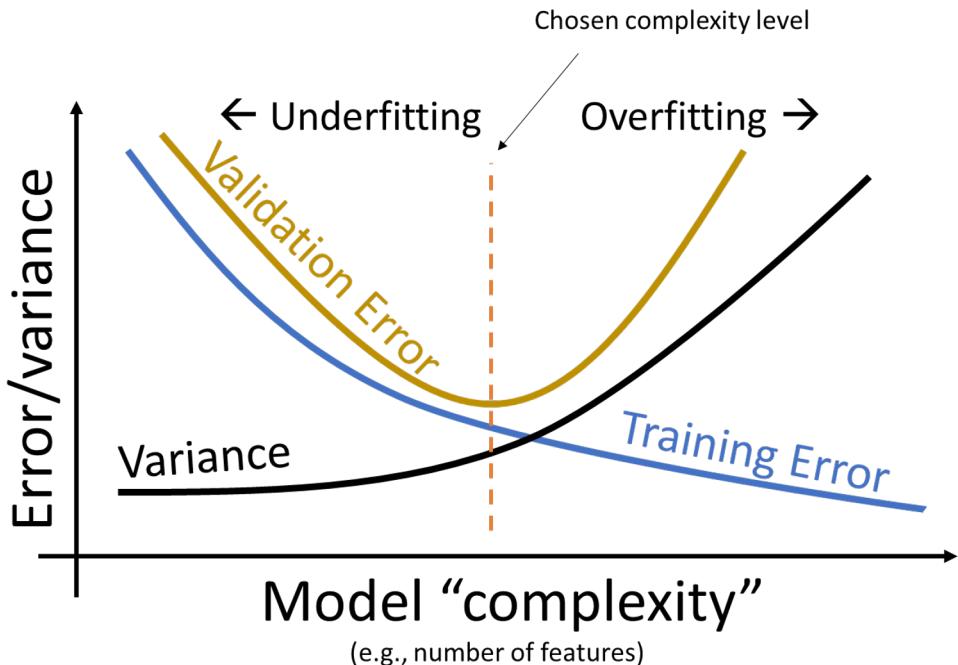
Our experiment using **vehicles**



General trend for an arbitrary dataset



Updating Our Understanding of Model Complexity



Typically, as model complexity increases:

- Training error decreases
- Variance increases
- Error on validation set decreases, then increases

Our goal: Choose the model complexity that minimizes validation error.

We will discuss how in the second half of lecture.

K-Fold Cross Validation

Cross Validation

- The Holdout Method
- **K-Fold Cross Validation**
- Test Sets

Regularization

- L2 Regularization (Ridge)
- Scaling Data for Regularization
- L1 Regularization (LASSO)

Another View of Validation

Introducing a validation set gave us one "extra" chance to assess model performance.

Specifically, now we understand how the model performs on *one* particular set of unseen data.

- It's possible that we may have, by random chance, selected a set of validation points that was *not* representative of other unseen data that the model might encounter.

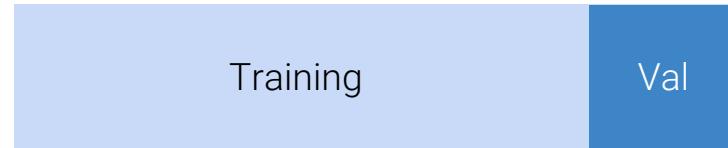
```
Val error from train/validation split #1: 14.6104005581132  
Val error from train/validation split #2: 24.755706579814404  
Val error from train/validation split #3: 22.23208329959848
```

Ideally: Assess model performance on *several* different validation sets before touching the test set.

Validation Folds

In our original validation split, we set aside x% of the training data to use for validation.

- For example, 20% of the training data is used for validation



We could have selected *any* 20% portion of the training data for validation.

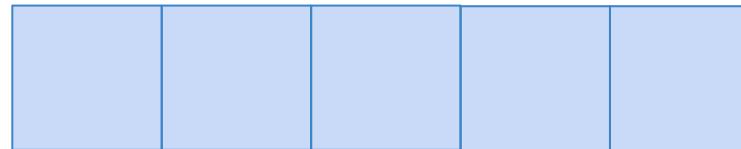


In total, there are 5 non-overlapping “chunks” of datapoints we could set aside for validation.

Validation Folds

The common term for one of these chunks is a "fold".

- Our training data has 5 folds, each containing 20% of the datapoints.



Another perspective: we actually have 5 validation sets "hidden" in our training set.

In **cross-validation**, we perform validation splits for each of these folds.

K-Fold Cross-Validation

For a dataset with K folds:

- Pick one fold to be the validation fold.
- Train model on data from every fold *other* than the validation fold.
- Compute the model's error on the validation fold and record it.
- Repeat for all K folds.

The **cross-validation error** is the average error across all K validation folds.



Train model on
all other folds Compute
MSE on V_1

Validation error #1

Validation error #2

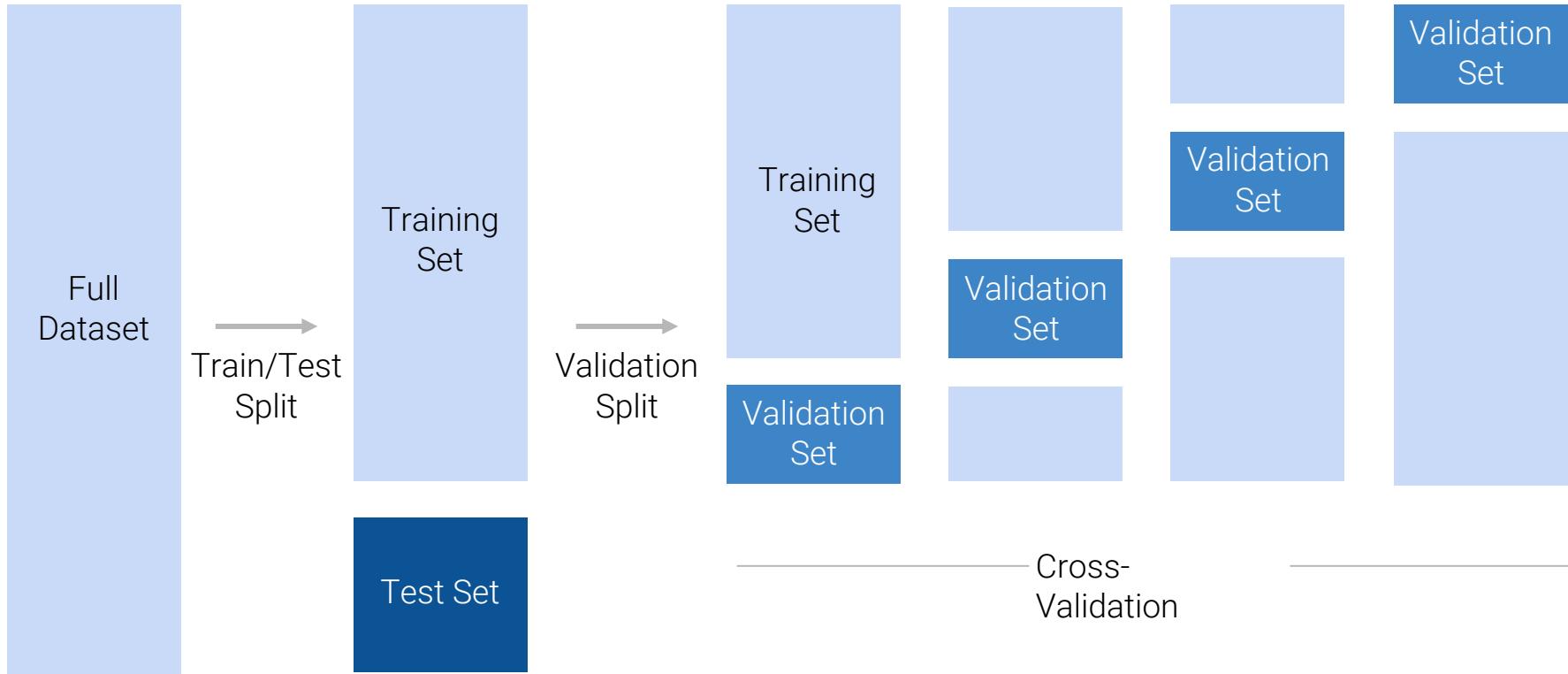
Validation error #3

Validation error #4

Validation error #5

Cross-validation error = mean of validation errors #1 to #5

Model Selection Workflow



Cross-validation is often used for **hyperparameter** selection.

Hyperparameter: Value in a model chosen *before* the model is fit to data.

- Cannot solve for hyperparameters via calculus, OLS, gradient descent, etc – we must choose it ourselves.
- Examples
 - Degree of polynomial model
 - Gradient descent learning rate, α
 - Regularization penalty, λ (to be introduced later this lecture)

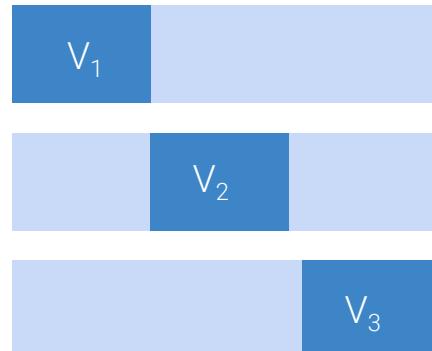
Hyperparameter Tuning

To select a hyperparameter value via cross-validation:

- List out several different “guesses” for the best hyperparameter.
- For each guess, run cross-validation to compute the CV error for that choice of hyperparameter value.
- Select the hyperparameter value with lowest CV error.

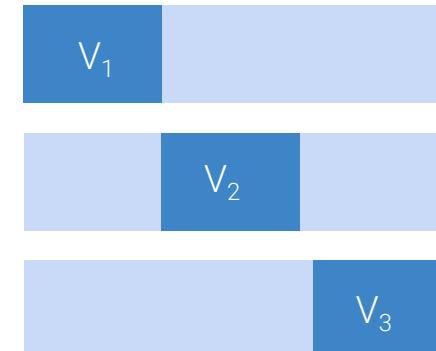
Example: Guesses for learning rate are 0.1, 1, and 10. We decide to apply 3-fold cross-validation.

$$\alpha = 0.1$$



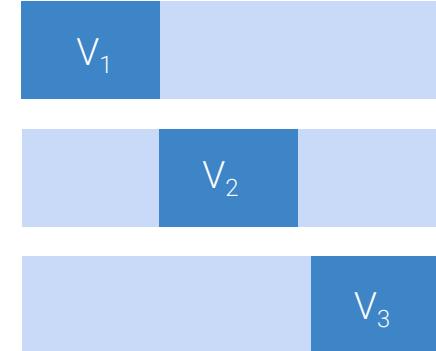
CV error: 4.67

$$\alpha = 1$$



CV error: 7.01

$$\alpha = 10$$



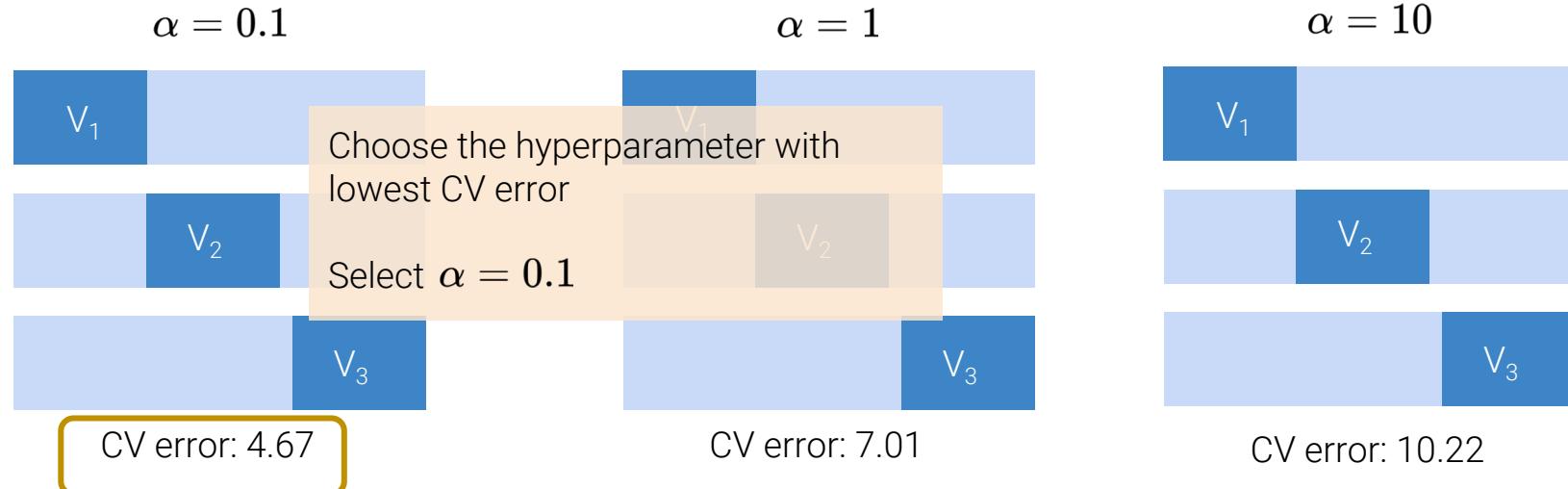
CV error: 10.22

Hyperparameter Tuning

To select a hyperparameter value via cross-validation:

- List out several different “guesses” for the best hyperparameter
- For each guess, run cross-validation to compute the CV error for that choice of hyperparameter value
- Select the hyperparameter value with lowest CV error

Example: Guesses for learning rate are 0.1, 1, and 10. We decide to apply 3-fold cross-validation.



Test Your Understanding

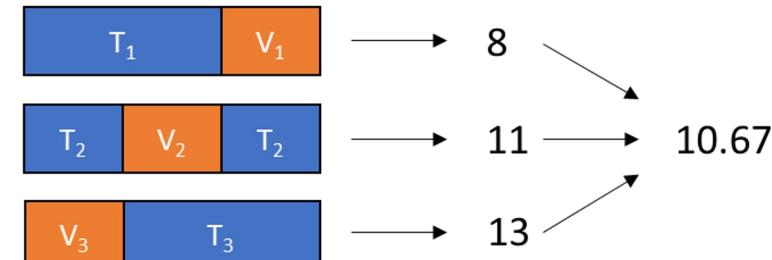
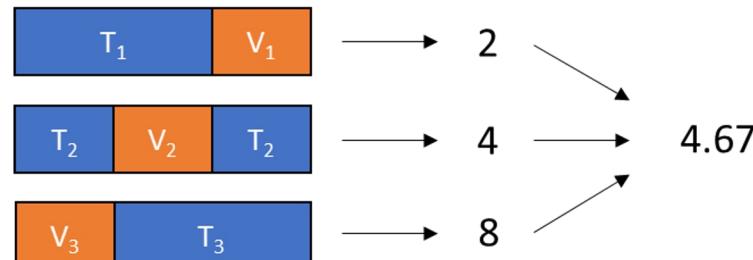
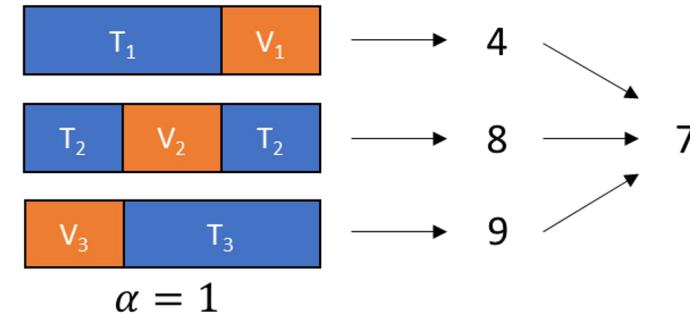
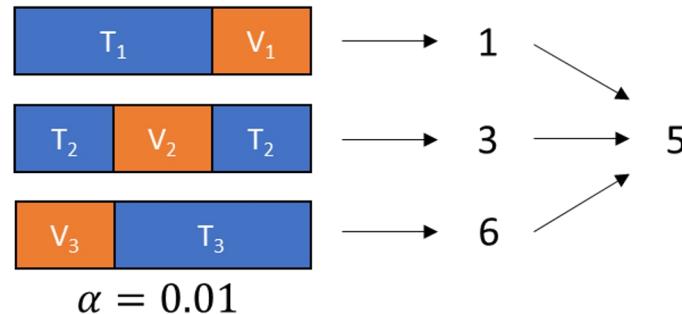
Suppose we pick $k = 3$ and we have 4 possible hyperparameter values $\alpha = [0.01, 0.1, 1, 10]$.

- How many total MSE values will we compute to get the quality of $\alpha=10$?
- How many total MSE values will we compute to find the best α ?

Test Your Understanding: How Many MSEs?

Suppose we pick $k = 3$ and we have 4 possible hyperparameter values $\alpha = [0.01, 0.1, 1, 10]$.

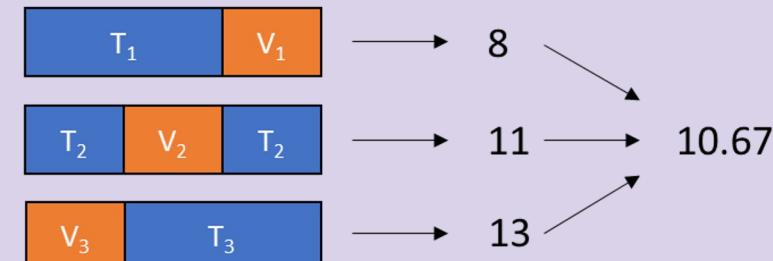
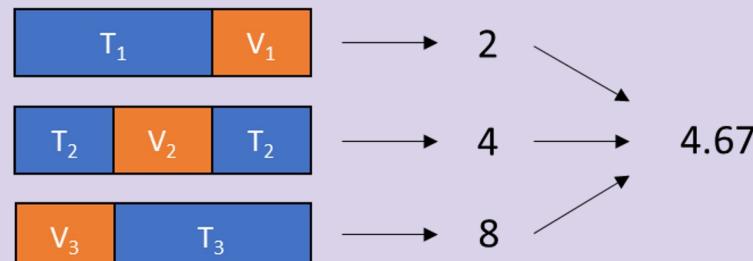
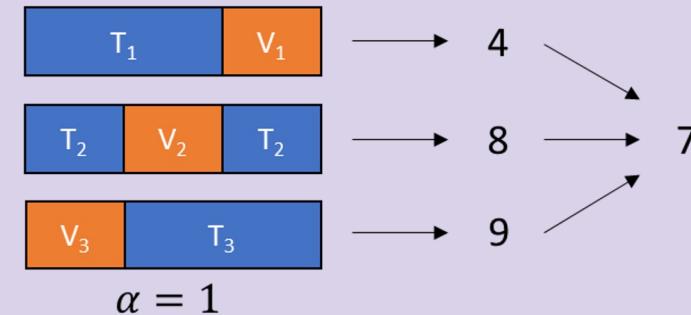
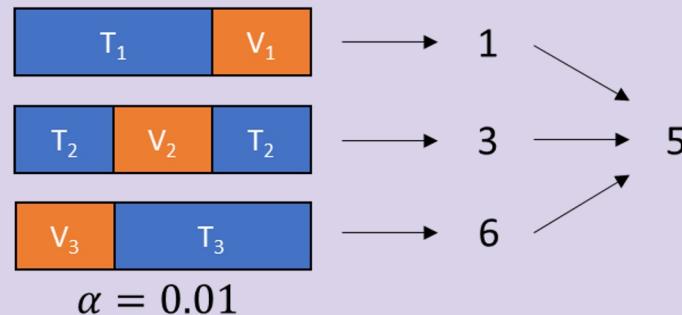
- How many total MSE values will we compute to get the quality of $\alpha=10$? 3
- How many total MSE values will we compute to find the best α ? 12



Test Your Understanding

Which α should we pick?

What fold (or folds) should we use as our training set for computing our final model parameters θ ?

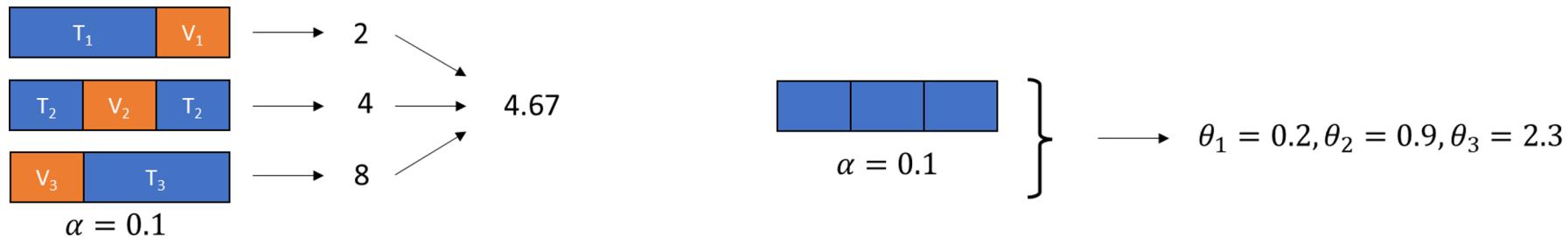


Test Your Understanding

Which α should we pick? 0.1

What fold (or folds) should we use as our training set for computing our final model parameters θ ?

- There's no reason to prefer any fold over any other. In practice, best to train all model on all of the data, i.e. use all 3 folds.



Picking K

Typical choices of k are 5, 10, and N, where N is the amount of data.

- K=N is also known as “leave one out cross validation”, and will typically give you the best results.
 - In this approach, each validation set is only one point.
 - Every point gets a chance to get used as the validation set.
- k=N is also very expensive, require you to fit a huge number of models.

Ultimately, the tradeoff is between k and computation time.

K-Fold Cross Validation and Hold Out Method in sklearn

As an example, the code below performs a GridSearchCV uses 5 fold cross validation to find the optimal parameters for a model called “scaled_ridge_model”.

- The hyperparameters to try are stored in a dictionary called “parameters_to_try”.
- The loss function is given by the “scoring” parameter.
- The number of folds is given by “cv = 5”.

```
model_finder = GridSearchCV(estimator = scaled_ridge_model,  
                            param_grid = parameters_to_try,  
                            scoring = "neg_mean_squared_error",  
                            cv = 5)
```

Can also do the Hold Out method in sklearn:

```
model_finder = GridSearchCV(estimator = scaled_ridge_model,  
                            param_grid = parameters_to_try,  
                            scoring = "neg_mean_squared_error",  
                            cv=[[training_indices, dev_indices]])
```

Cross Validation Summary

When selecting between models, we want to pick the one that we believe would generalize best on unseen data. Generalization is estimated with a “**cross validation score**”*.

- When selecting between models, keep the model with the best **score**.

Two techniques to compute a “**cross validation score**”:

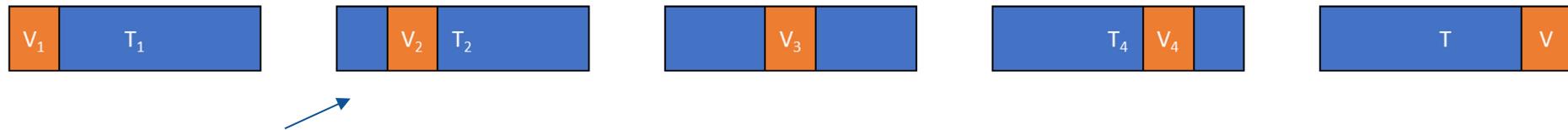
- The Holdout Method: Break data into a separate **training set** and **validation set**.
 - Use **training set** to **fit** parameters (thetas) for the model.
 - Use **validation set** to **score** the model.
 - Also called “Simple Cross Validation” in some sources.
- K-Fold Cross Validation: Break data into K contiguous non-overlapping “folds”.
 - Perform K rounds of Simple Cross Validation, except:
 - Each fold gets to be the **validation set** exactly once.
 - The final **score** of a model is the **average validation score** across the K trials.

Extra: Exhaustive Cross Validation

We don't have to use non-overlapping contiguous chunks! Could use

- Every 5th data point as validation set.
- Data in positions 0-2%, 20-22%, 40-42%, 60-62%, 80-82% as validation set.
- Etc.

Iterating over ALL possible such permutations is known as "exhaustive cross validation."



Use first 20% and last 60%
to train, remaining 20% as
validation set.

Test Sets

Cross Validation

- The Holdout Method
- K-Fold Cross Validation
- **Test Sets**

Regularization

- L2 Regularization (Ridge)
- Scaling Data for Regularization
- L1 Regularization (LASSO)

Providing an Estimate of Model Generalization to the World

Suppose we're researchers building a state of the art regression model.

- After months of work and after comparing billions of candidate models, we find the model with the best **validation set loss**.

Now we want to report this model out to the world so it can be compared to other models.

- Our **validation set loss** is not an unbiased estimator of its performance!
- Instead, we'll run our model just one more time on a special **test set**, that we've never seen or used for any purpose whatsoever.

Why Are Validation Set Errors Biased?

Analogy:

- Imagine we have a golf ball hitting competition. Whoever can hit the ball the farthest wins.
- Suppose we have the best 10000000 golfers in the world **play a tournament**. There are probably many roughly equal players near the top.
- When we're done, we want to provide an unbiased estimate of our best golfer's distance in yards.
- Using the **tournament results** may be biased, as the the winner maybe got just a bit lucky (maybe they had favorable wind during their rounds).
- Better unbiased estimate: Have the winner **play one more trial and report their score**.

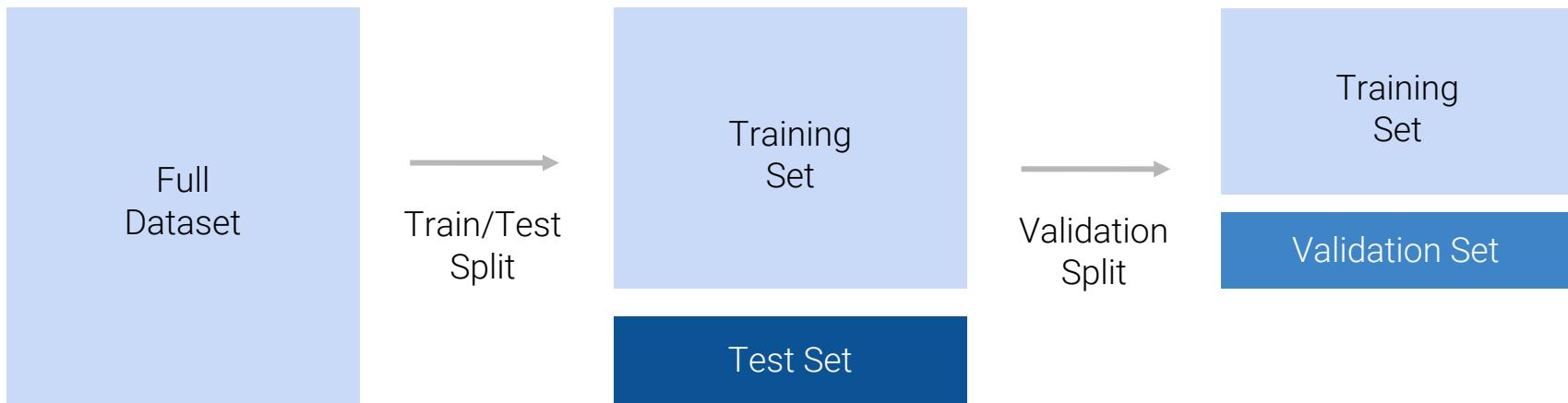


Test Sets

Test sets can be something that we generate ourselves. Or they can be a common data set whose solution is unknown.

In real world machine learning competitions, competing teams share a **common test set**.

- To avoid accidental or intentional overfitting, the **correct predictions for the test set are never seen by the competitors**.



Creating a Test Set Ourselves

We can do this easily in code. As before, we shuffle using scikit-learn then split using numpy.

```
diamond_data = shuffle(diamond_data)  
diamond_data.head()
```

```
#split our 2000 rows into 1500 for training, 300 for validation, 200 for test  
diamond_training_data, diamond_validation_data, diamond_test_data = np.split(diamond_data, [1500, 1800])
```

Then we use np.split, now providing two numbers instead of one. For example, the code above splits the data into a **Training**, **Validation**, and **Test** set.

- Recall that a **validation set** is just another name for a **development set**.
- **Training set** used to pick parameters.
- **Validation set** used to pick hyperparameters (or **pick between different models**).
- **Test set** used to provide an **unbiased MSE at the end**.

Validation Sets and Test Sets in Real World Practice

Standard **validation sets** and **test sets** are used as standard benchmarks to compare ML algorithms.

- Example: [ImageNet](#) is a dataset / competition used to compare to different image classification algorithms (e.g. this image is of a “Dog”).
 - 1,281,167 **training images**. Images and correct label provided.
 - 50,000 **validation images**. Images and correct label provided.
 - 100,000 **test images**. Images provided, but no correct label.
- When writing papers, researchers report their performance on the **validation images**.
 - This set is a “**validation set**” with respect to the entire global research community.
 - Research groups cannot report the **test error** because they cannot compute it!
- When ImageNet was a competition, the **test set** was used to rank different algorithms.
 - Researchers provide their predictions for the **test set** to a central server.
 - Server (which knows the labels) reports back a **test set score**.
 - Best **test set score** wins.

Regularization

Cross Validation

- The Holdout Method
- K-Fold Cross Validation
- Test Sets

Regularization

- L2 Regularization (Ridge)
- Scaling Data for Regularization
- L1 Regularization (LASSO)

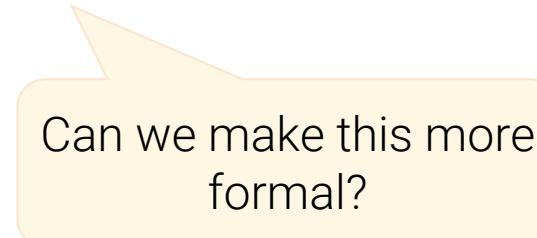
Controlling the *Model Complexity*

Basic Idea

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_{\theta}(x_i))$$

Such that:

f_{θ} does not “overfit”



Can we make this more formal?

Controlling the *Model Complexity*

Basic Idea

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_{\theta}(x_i))$$

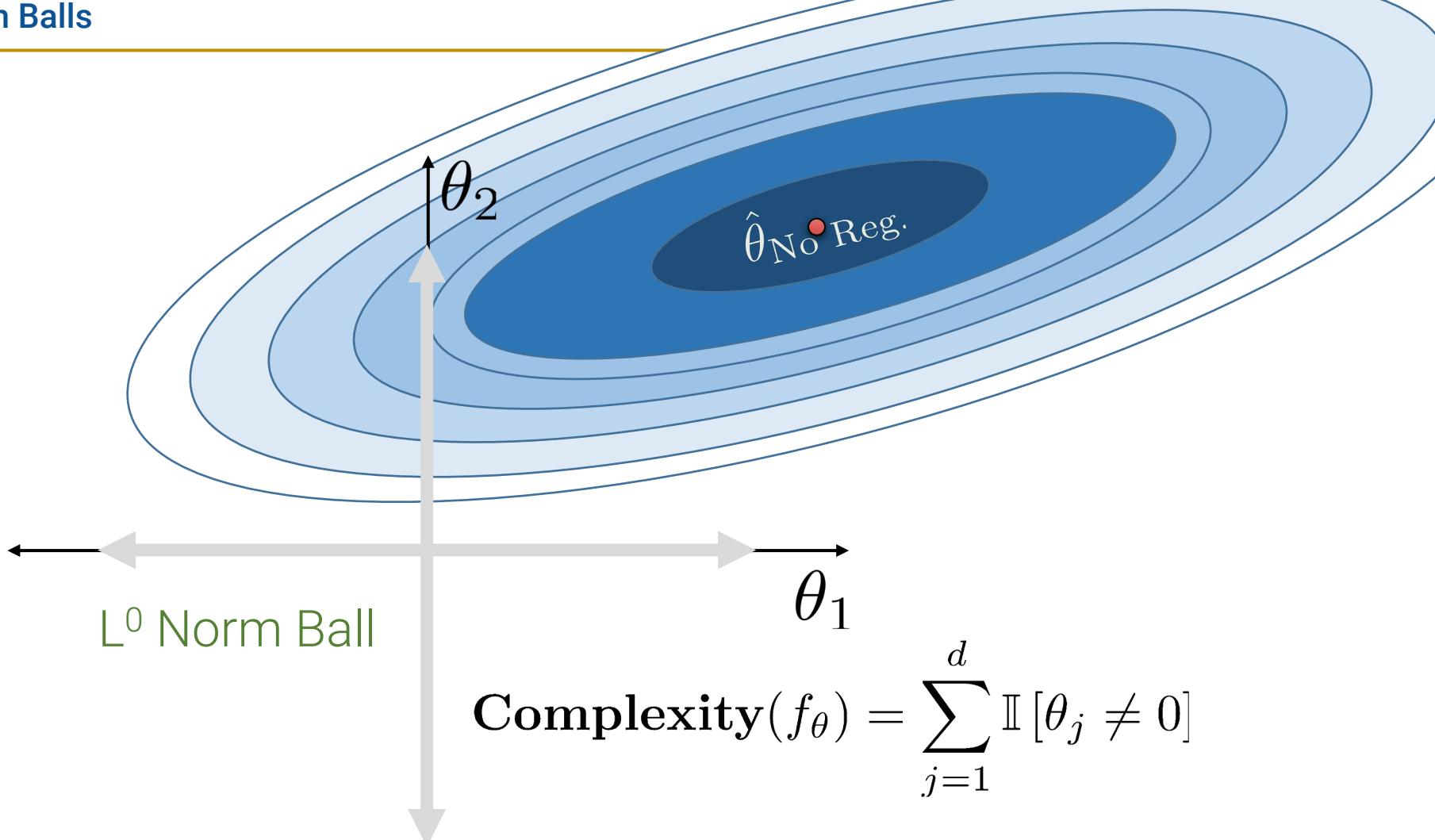
Such that:

Regularization
Hyperparameter

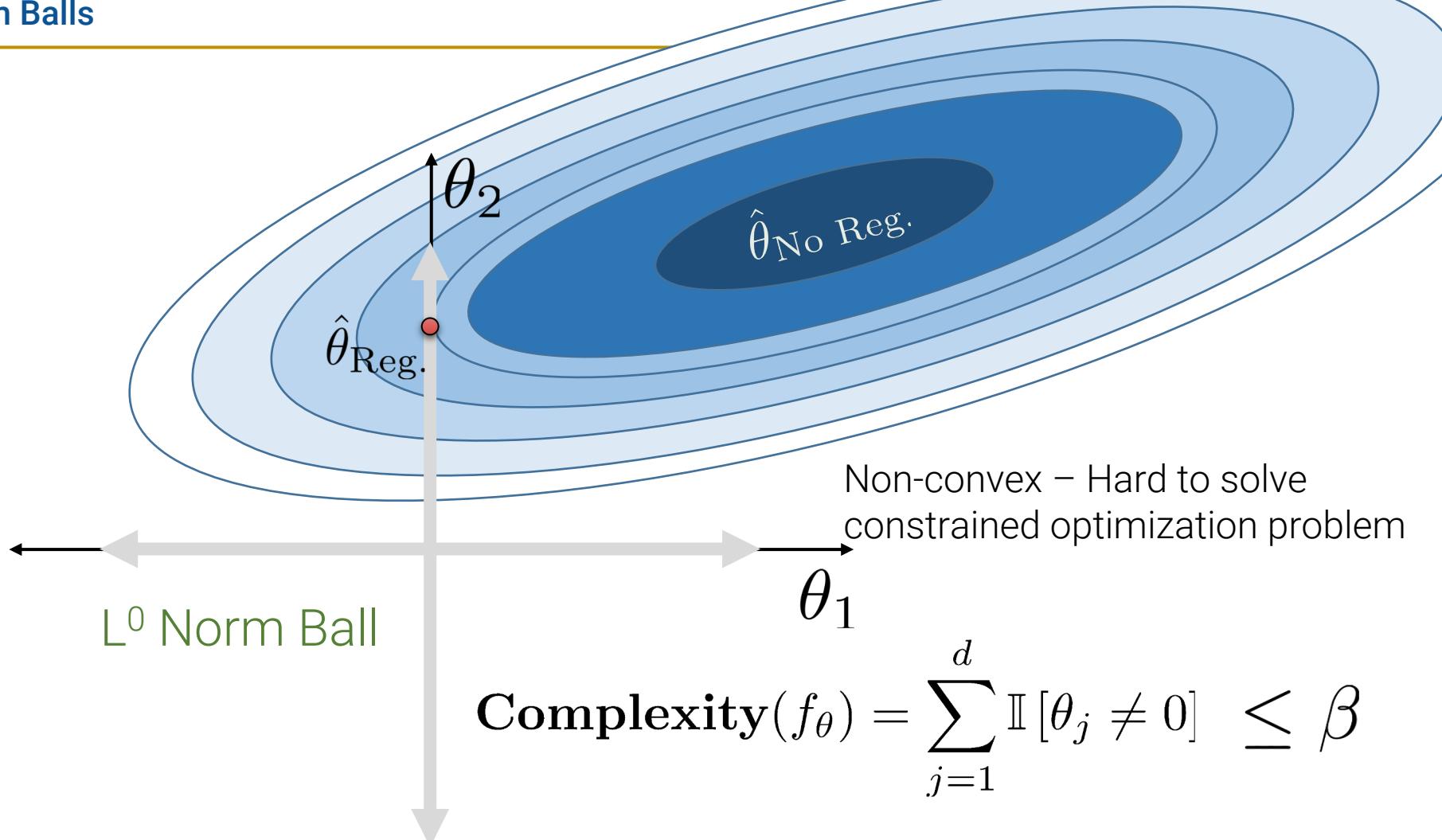
$$\text{Complexity}(f_{\theta}) \leq \beta$$

How do we define
this?

Norm Balls



Norm Balls



Find the best value of θ which uses fewer than β features.

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_{\theta}(x_i))$$

Such that:

Need an approximation!

$$\text{Complexity}(f_{\theta}) = \sum_{j=1}^d \mathbb{I}[\theta_j \neq 0] \leq \beta$$

Combinatorial search problem – NP-hard to solve in general.

Restricting Model Complexity

Idea: Only use each feature "a little" in the model.

$$\hat{Y} = \theta_0 + \theta_1\phi_1 + \theta_2\phi_2 \dots + \theta_p\phi_p$$

- If we restrict how large each parameter θ_i can be, we restrict how much each feature contributes to the model.
- When θ_i is close to or equal to 0, the model decreases in complexity because feature ϕ_i barely impacts the prediction.

In **regularization**, we restrict complexity by *putting a limit* on the magnitudes of the model parameters θ_i .

L2 Regularization (Ridge)

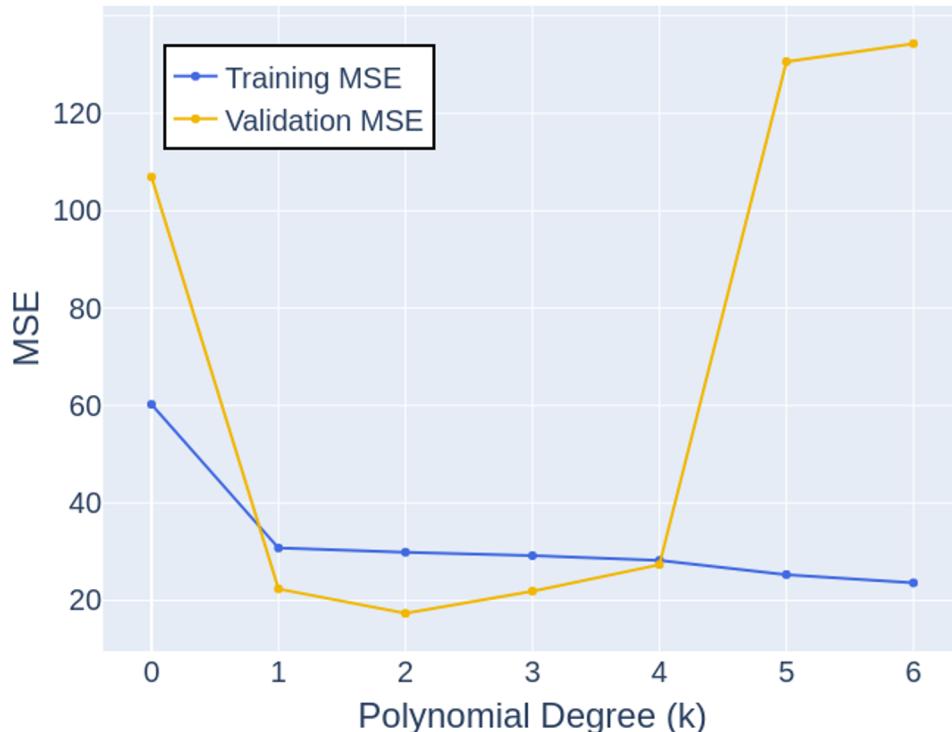
Cross Validation

- The Holdout Method
- K-Fold Cross Validation
- Test Sets

Regularization

- **L2 Regularization (Ridge)**
- Scaling Data for Regularization
- L1 Regularization (LASSO)

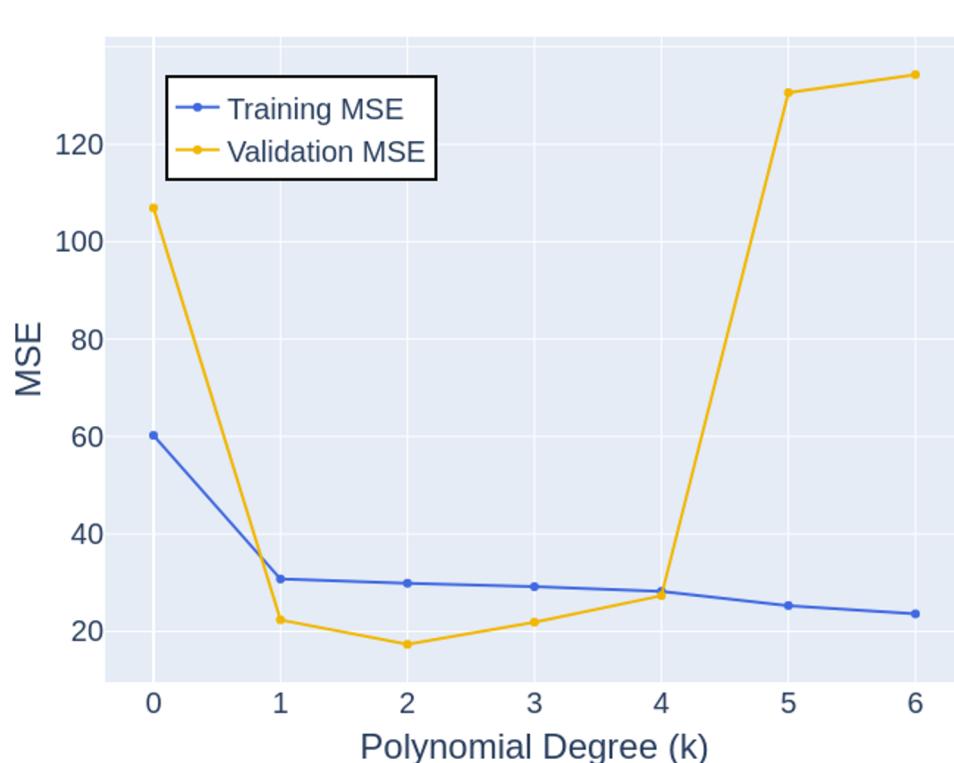
We saw how we can select model complexity by choosing the hyperparameter that minimizes **validation error**. This **validation error** can be computed using the **Holdout Method** or **K-Fold Cross Validation**.



k	Training MSE	Validation MSE
0	60.235744	106.925296
1	30.756678	22.363676
2	29.875269	17.331880
3	29.180868	21.889257
4	28.214850	27.340989
5	25.290990	130.597678
6	23.679651	135.787493

For the example below, our hyperparameter was the polynomial degree.

- Tweaking the “complexity” is simple, just increase or decrease the degree.



k	Training MSE	Validation MSE
0	60.235744	106.925296
1	30.756678	22.363676
2	29.875269	17.331880
3	29.180868	21.889257
4	28.214850	27.340989
5	25.290990	130.597678
6	23.679651	135.787493

A More Complex Example

Suppose we have a dataset with 9 features.

- We want to decide which of the 9 features to include in our linear regression.

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

Tweaking Complexity via Feature Selection

With 9 features, there are 2^9 different models. One approach:

- For each of the 2^9 linear regression models, compute the **validation MSE**.
- Pick the model that has the lowest **validation MSE**.

Runtime is exponential in the number of parameters!

	hp	w	dis	hp ²	hp w	hp dis	w ²	w dis	dis ²	MSE
Least complex model	no	no	no	no	no	no	no	no	no	172.2
	no	no	no	no	no	no	no	no	yes	77.3
	no	no	no	no	no	no	no	yes	no	85.3
	no	no	no	no	no	no	no	yes	yes	77.2
	no	no	no	no	no	no	yes	no	no	81.1
	no	no	no	no	no	no	yes	no	yes	74.6
...										
Most complex model	yes	yes	yes	yes	yes	yes	yes	yes	yes	195.3

Tweaking Complexity via Feature Selection

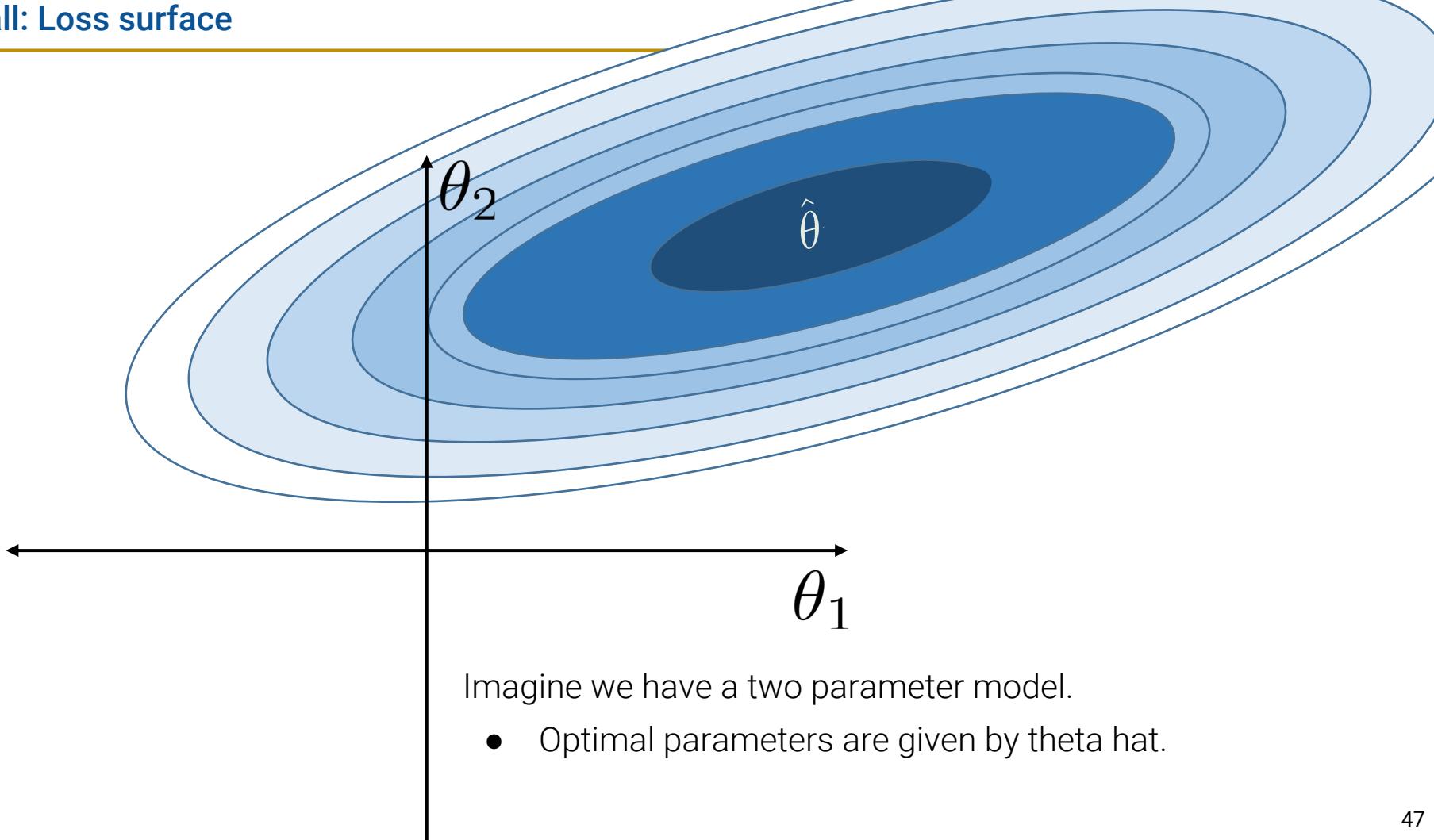
Alternate Idea: What if we use all of the features, but only a little bit?

- Let's see a simple example for a 2 feature model.
- Will return to this 9 feature model later.



	hp	w	dis	hp^2	hp w	hp dis	w^2	w dis	dis^2	MSE
Least complex model	no	no	no	no	no	no	no	no	no	172.2
	no	no	no	no	no	no	no	no	yes	77.3
	no	no	no	no	no	no	no	yes	no	85.3
	no	no	no	no	no	no	no	yes	yes	77.2
	no	no	no	no	no	no	yes	no	no	81.1
	no	no	no	no	no	no	yes	no	yes	74.6
...										
Most complex model	yes	yes	yes	yes	yes	yes	yes	yes	yes	195.3

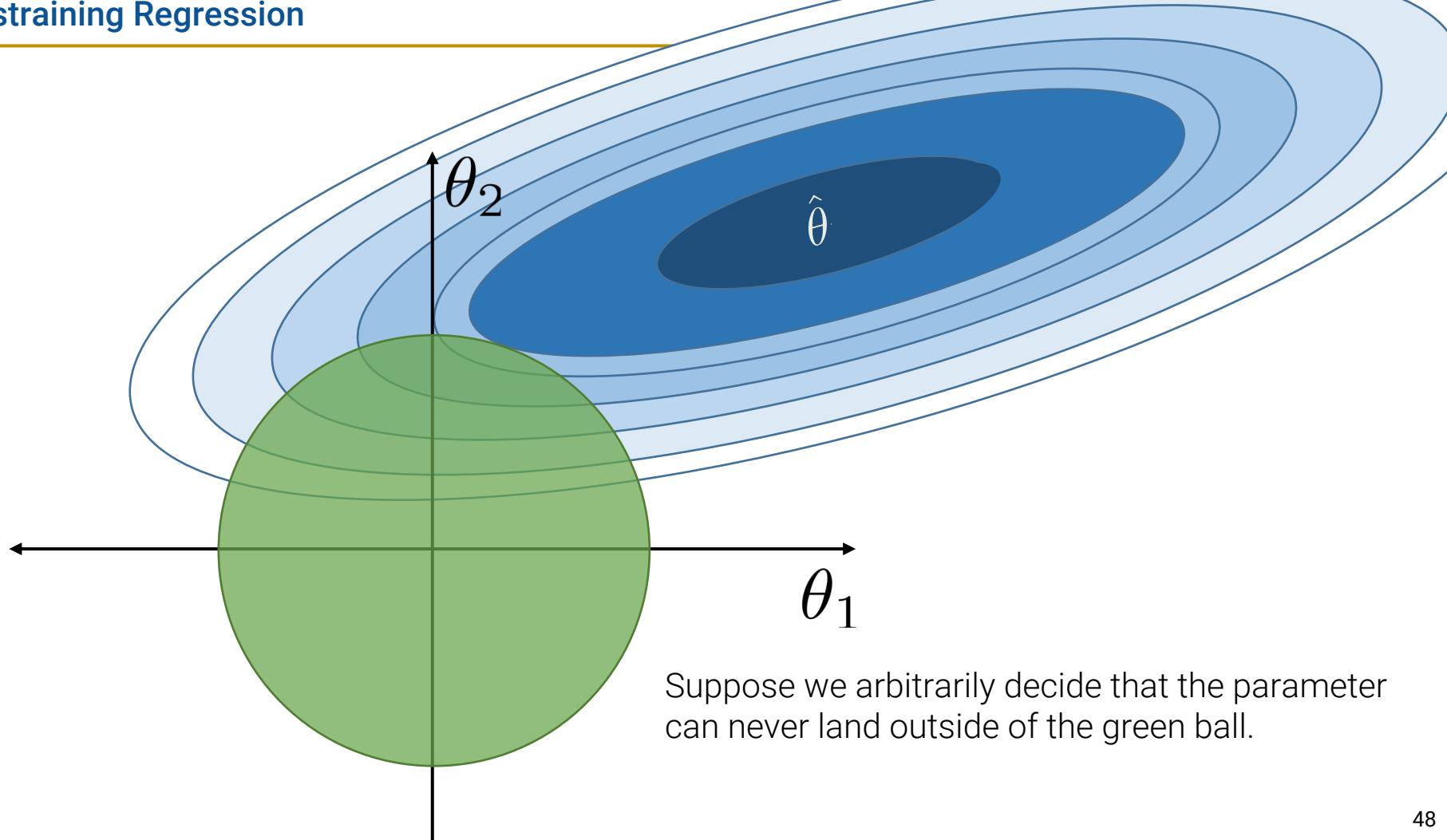
Recall: Loss surface



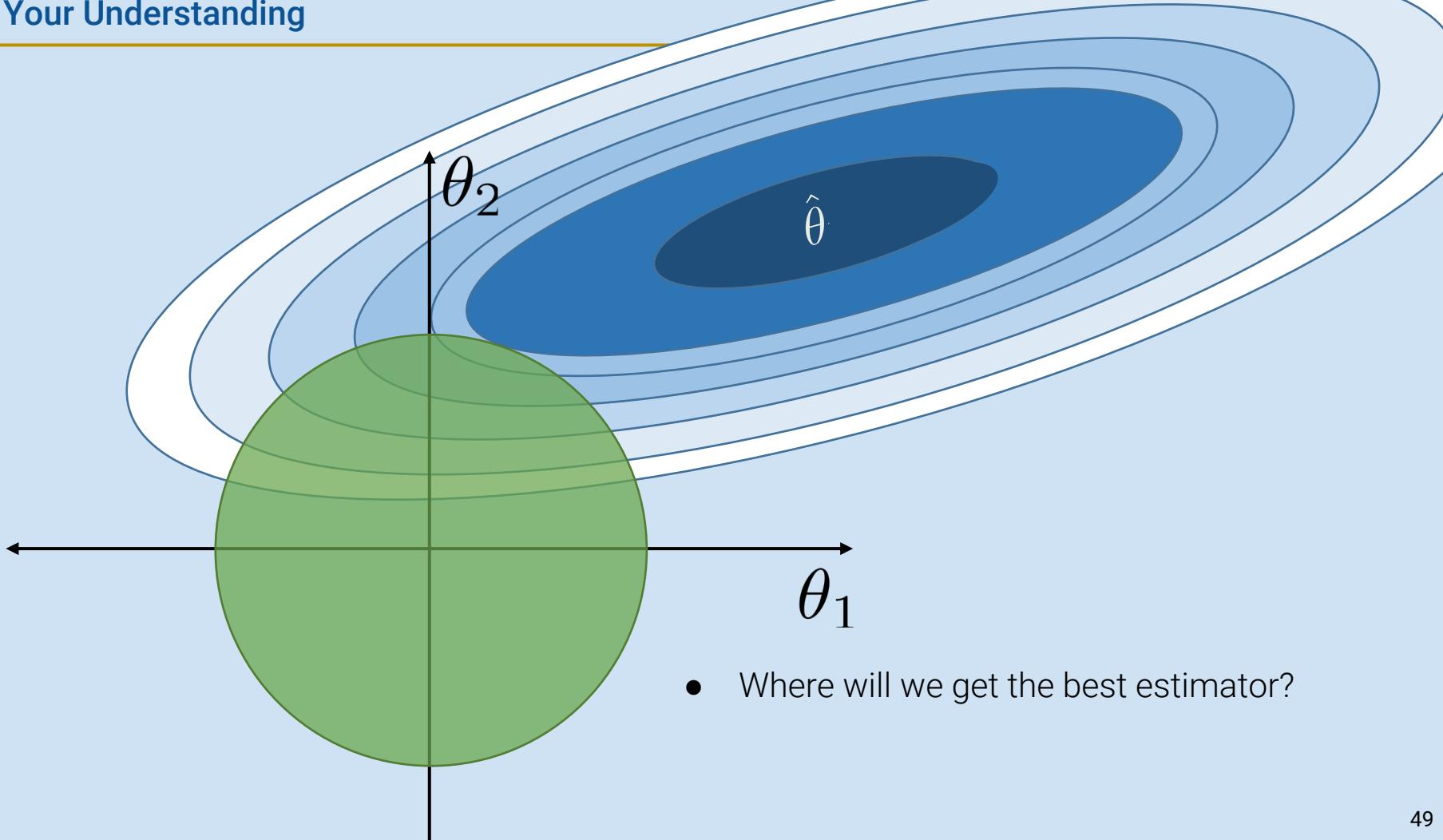
Imagine we have a two parameter model.

- Optimal parameters are given by theta hat.

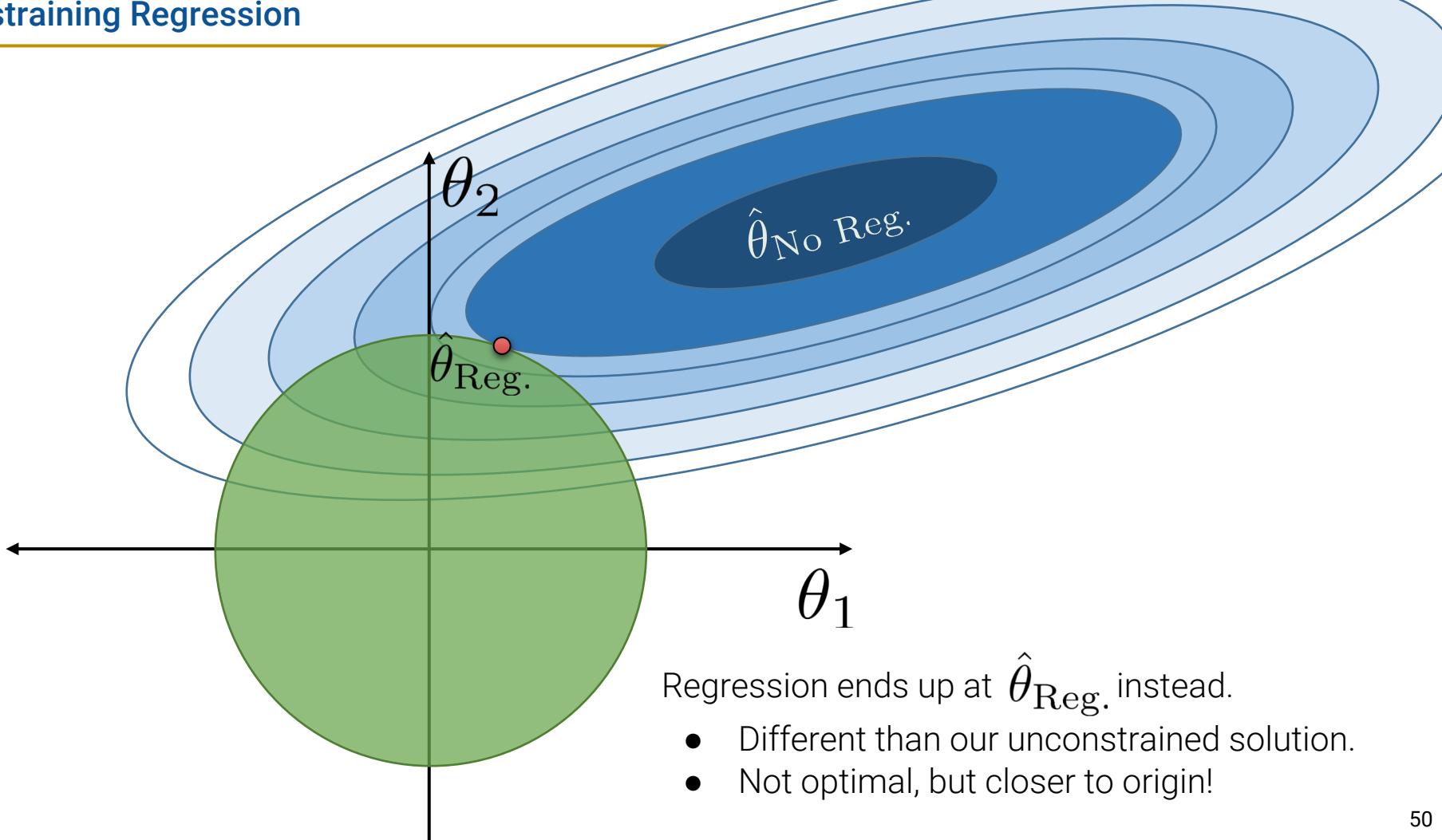
Constraining Regression



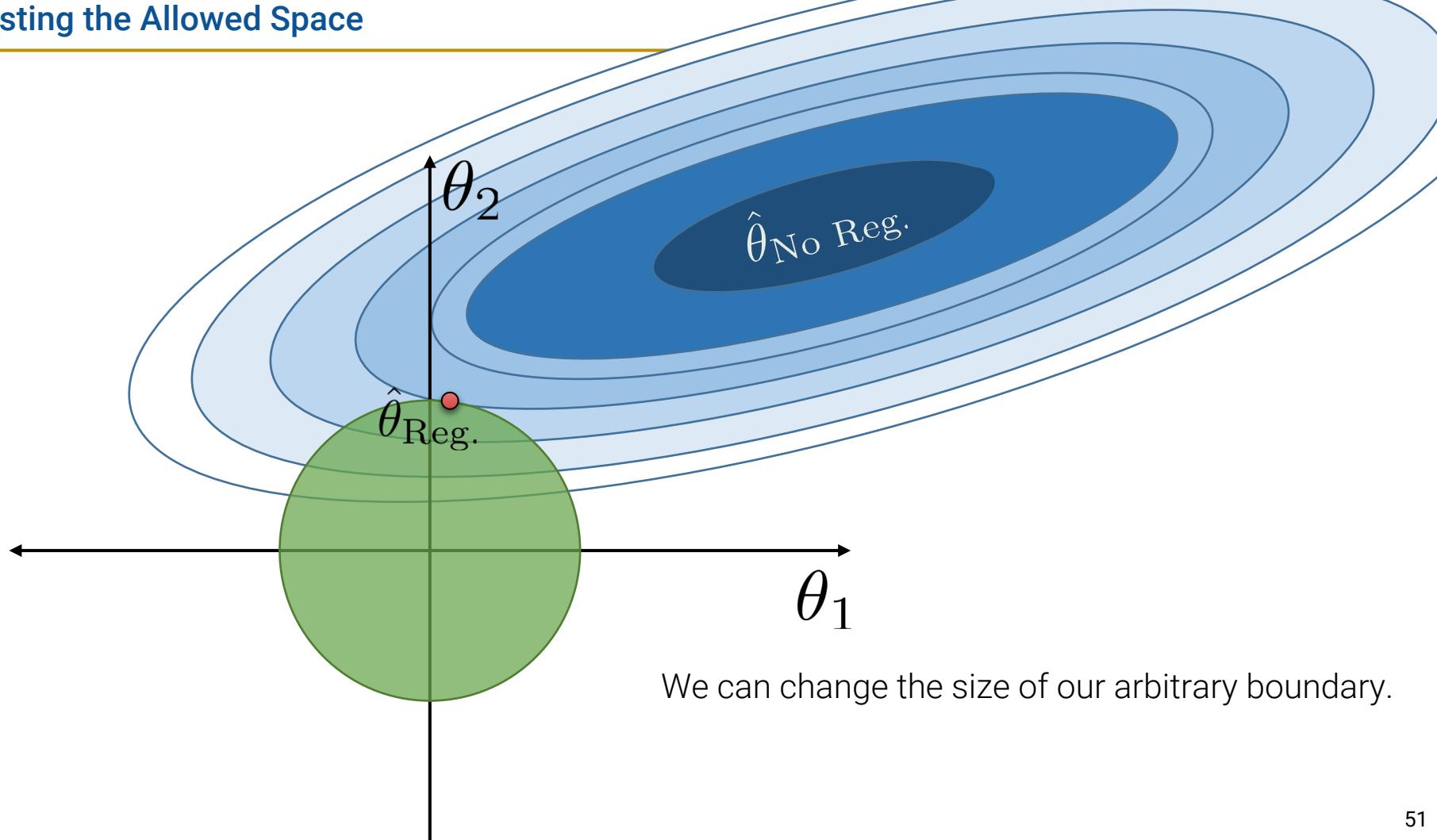
Test Your Understanding



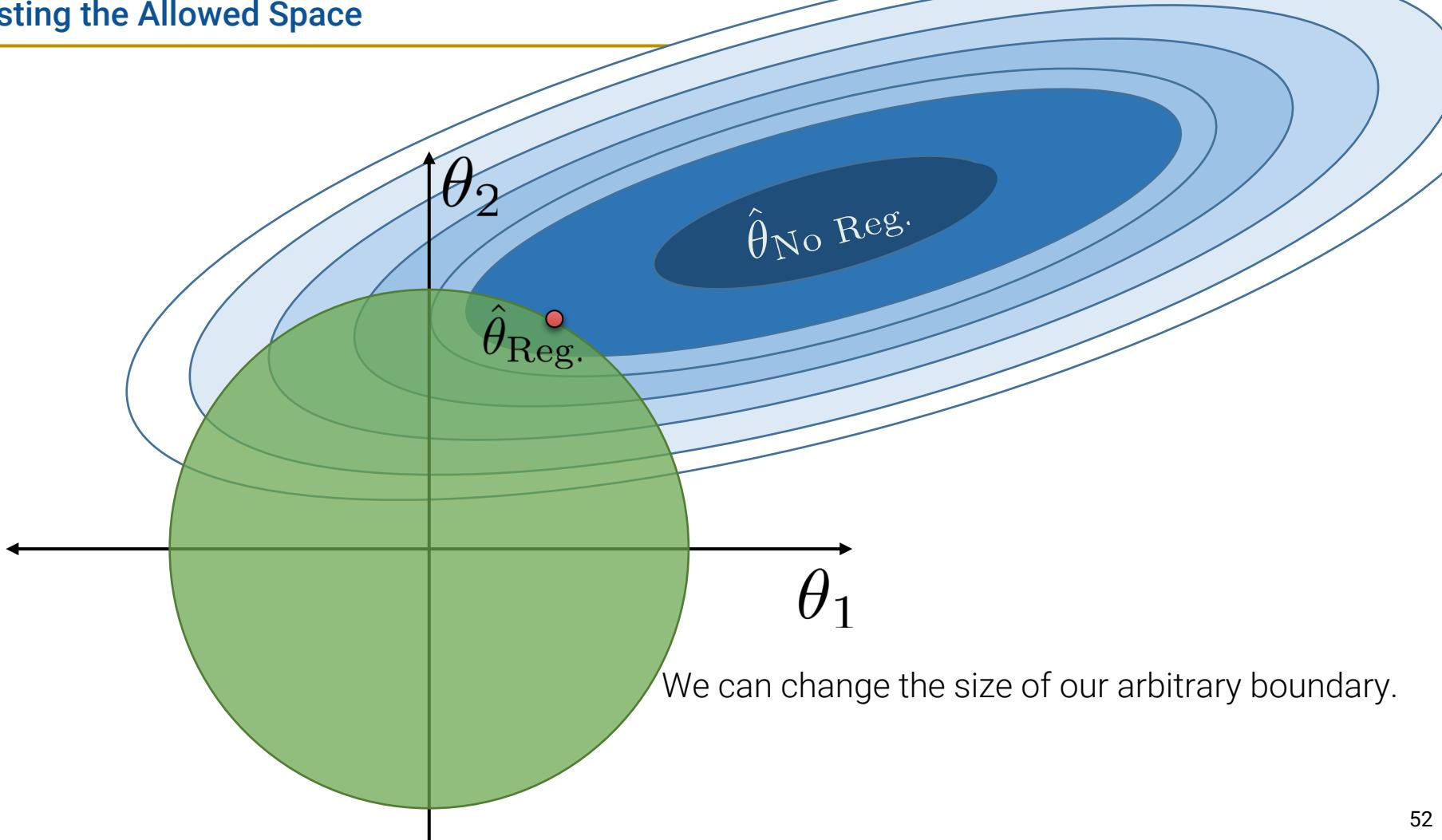
Constraining Regression



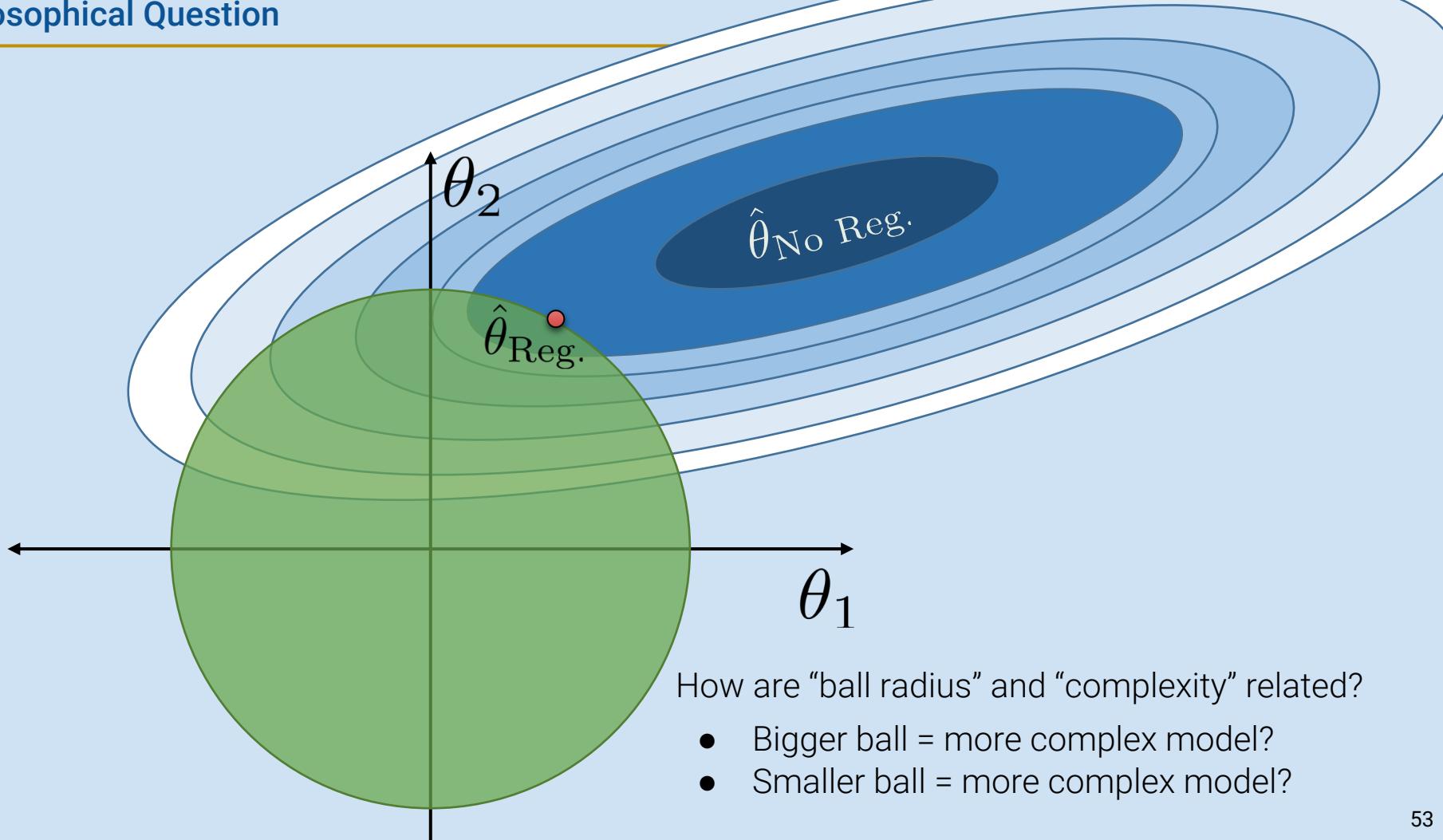
Adjusting the Allowed Space



Adjusting the Allowed Space



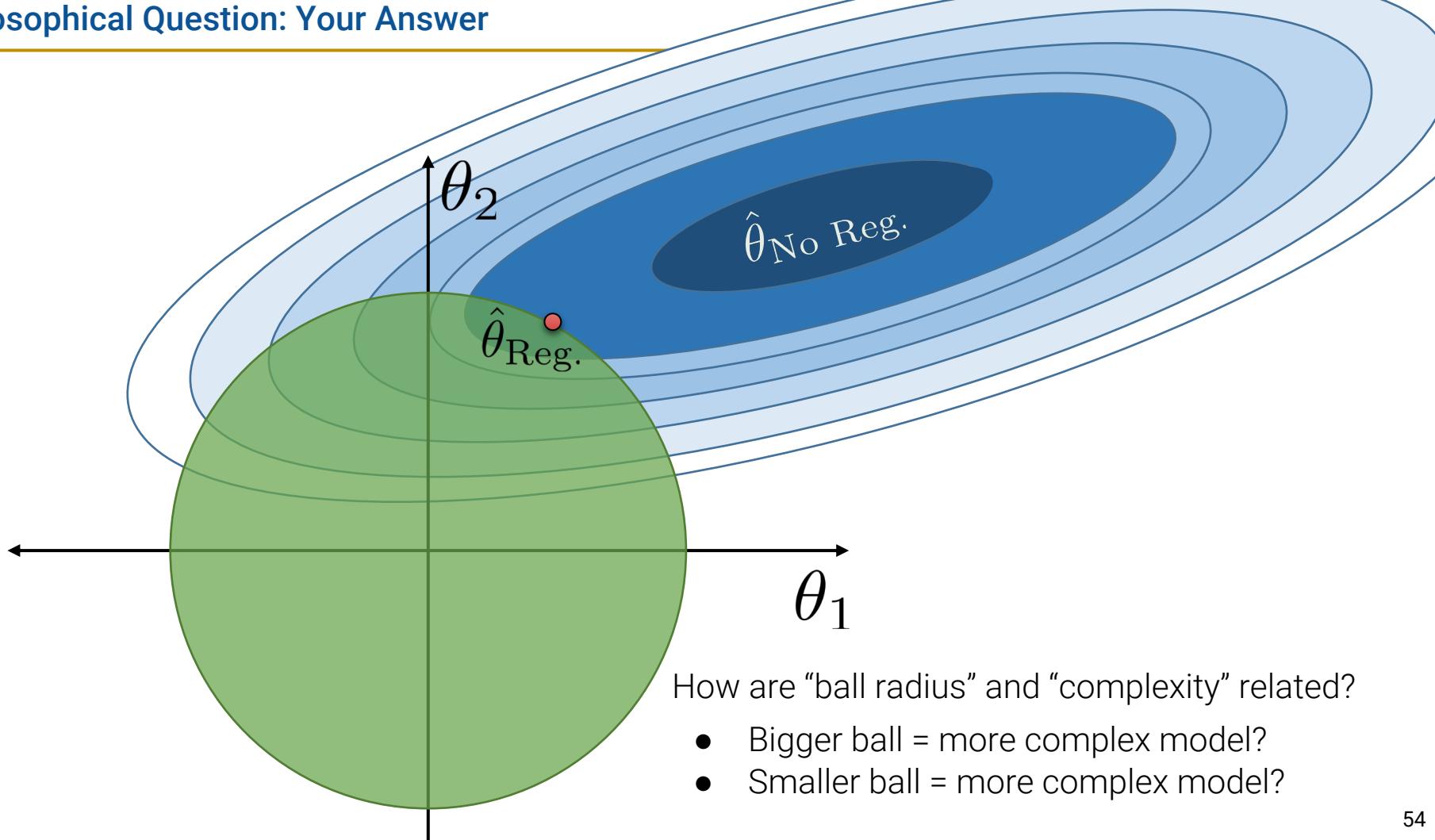
Philosophical Question



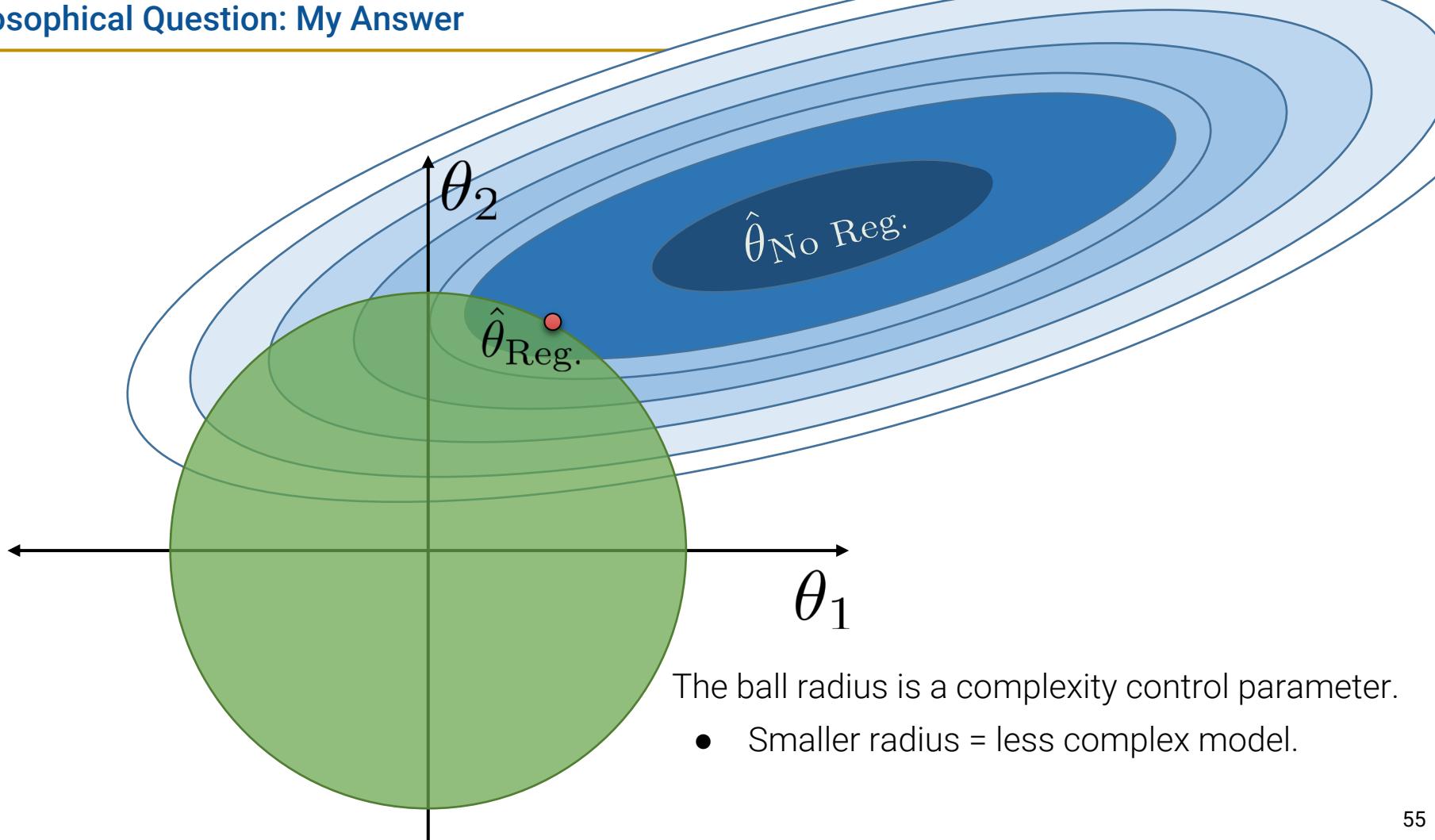
How are “ball radius” and “complexity” related?

- Bigger ball = more complex model?
- Smaller ball = more complex model?

Philosophical Question: Your Answer



Philosophical Question: My Answer



Test Your Understanding

Let's return to our 9 feature model from before ($d = 9$).

- If we pick a very small ball radius, what kind of model will we have? (Choose all that apply)
 - A model that only returns zero.
 - A constant model.
 - Ordinary least squares.
 - Something else.

$$\hat{y} = \theta_0 + \theta_1\phi_1 + \dots + \theta_d\phi_d$$

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

Test Your Understanding Answer

Let's return to our 9 feature model from before ($d = 9$).

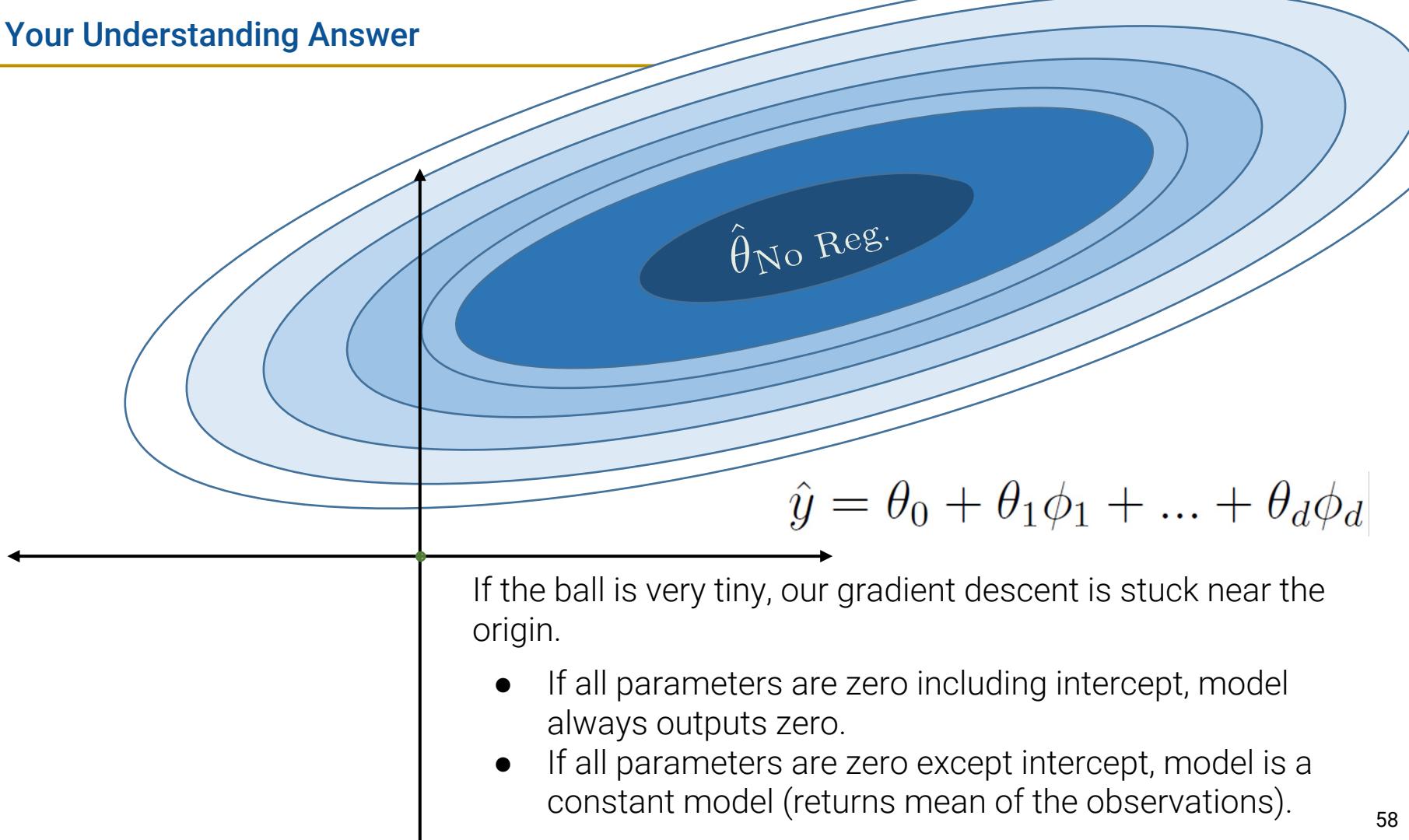
- If we pick a very small ball radius, what kind of model will we have?

- a. A model that only returns zero.
- b. A constant model.
- c. Ordinary least squares.
- d. Something else.

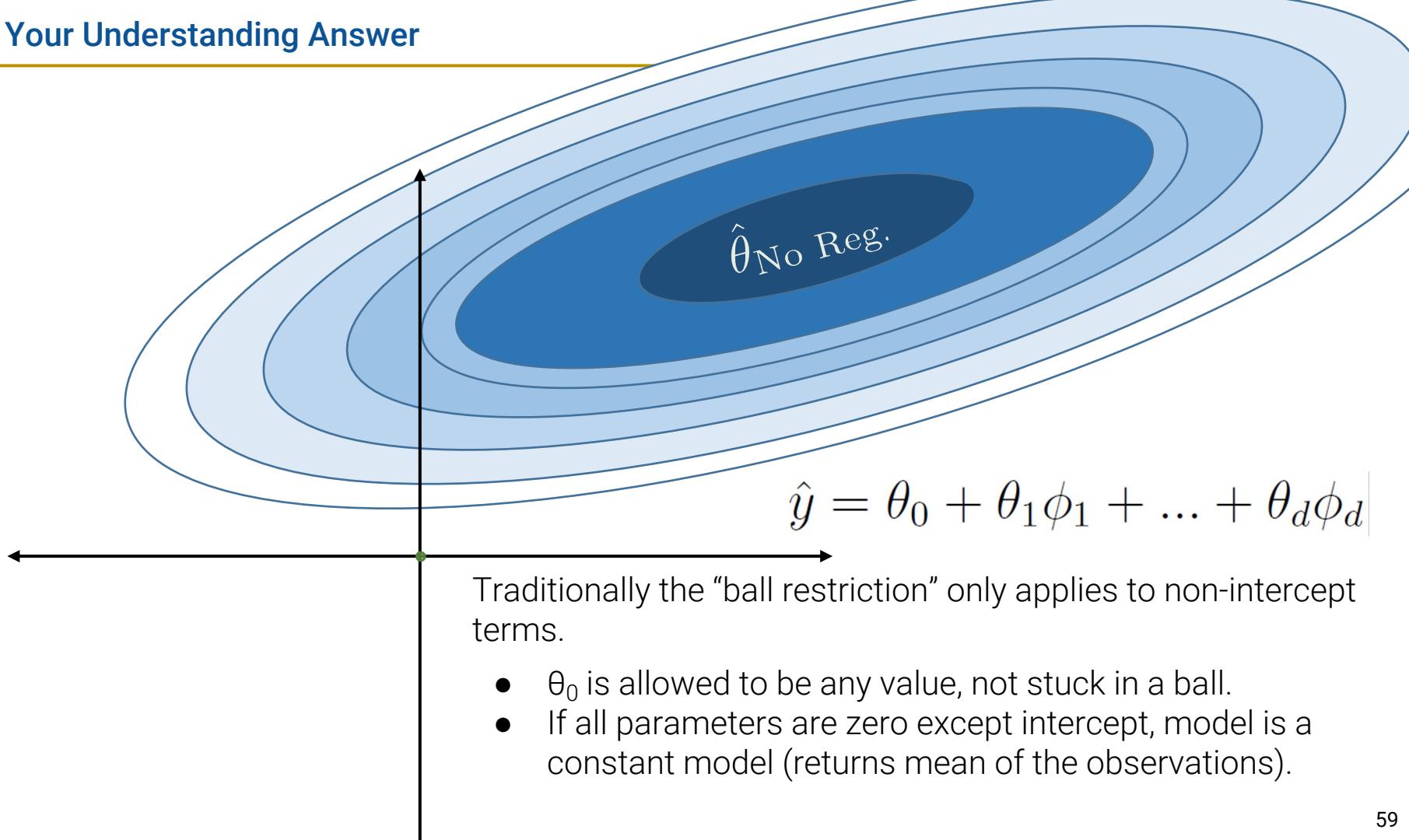
$$\hat{y} = \theta_0 + \theta_1\phi_1 + \dots + \theta_d\phi_d$$

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

Test Your Understanding Answer



Test Your Understanding Answer



Test Your Understanding

Back to our 9 feature model from before ($d = 9$).

- If we pick a very large ball radius, what kind of model will we have?

- a. A model that only returns zero.
- b. A constant model.
- c. Ordinary least squares.
- d. Something else.

$$\hat{y} = \theta_0 + \theta_1\phi_1 + \dots + \theta_d\phi_d$$

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

Test Your Understanding Answer

Back to our 9 feature model from before ($d = 9$).

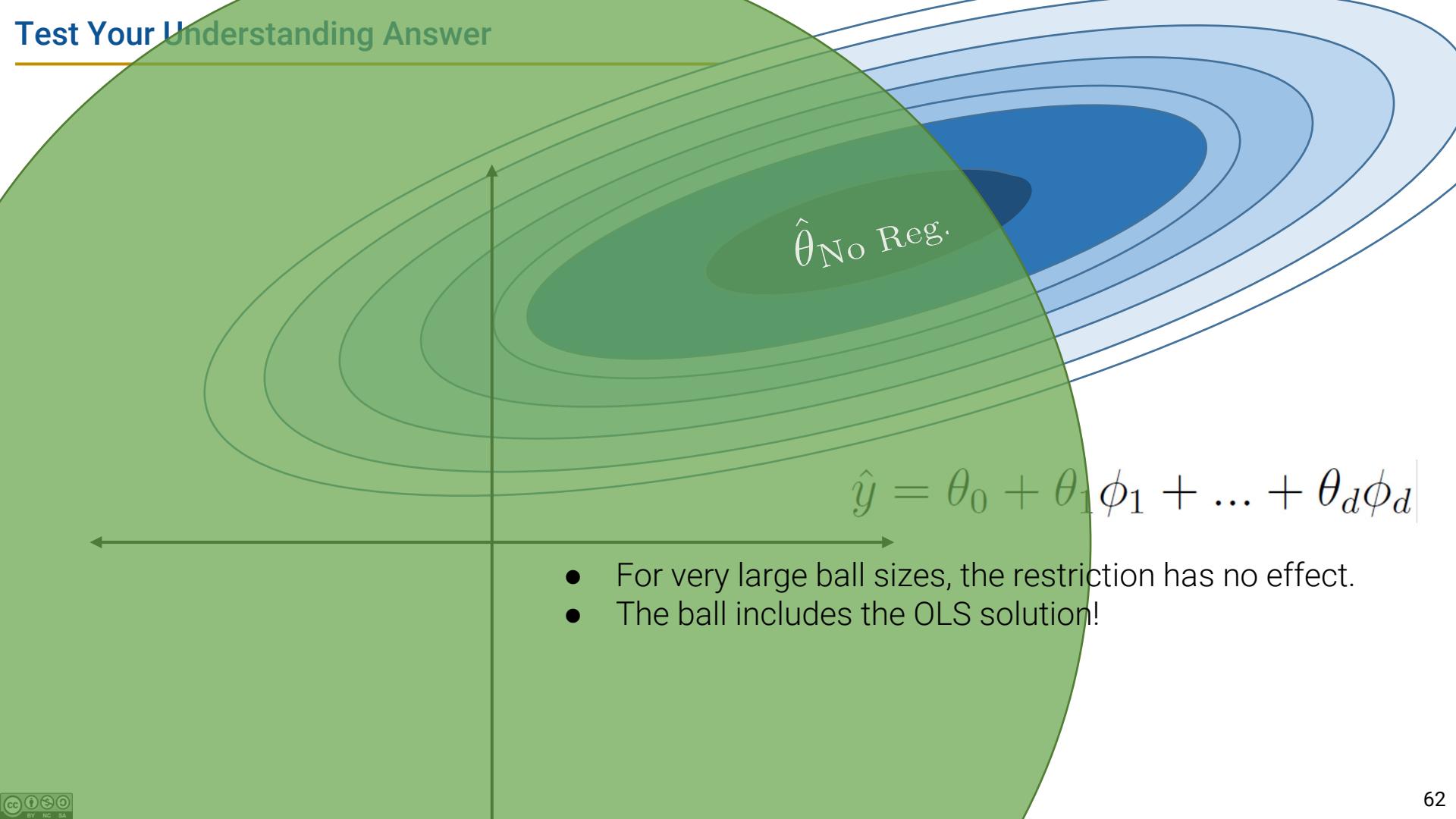
- If we pick a very large ball radius, what kind of model will we have?

- a. A model that only returns zero.
- b. A constant model.
- c. **Ordinary least squares.**
- d. Something else.

$$\hat{y} = \theta_0 + \theta_1\phi_1 + \dots + \theta_d\phi_d$$

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

Test Your Understanding Answer



Training and Validation Errors vs. Ball Size for Our 9D Model

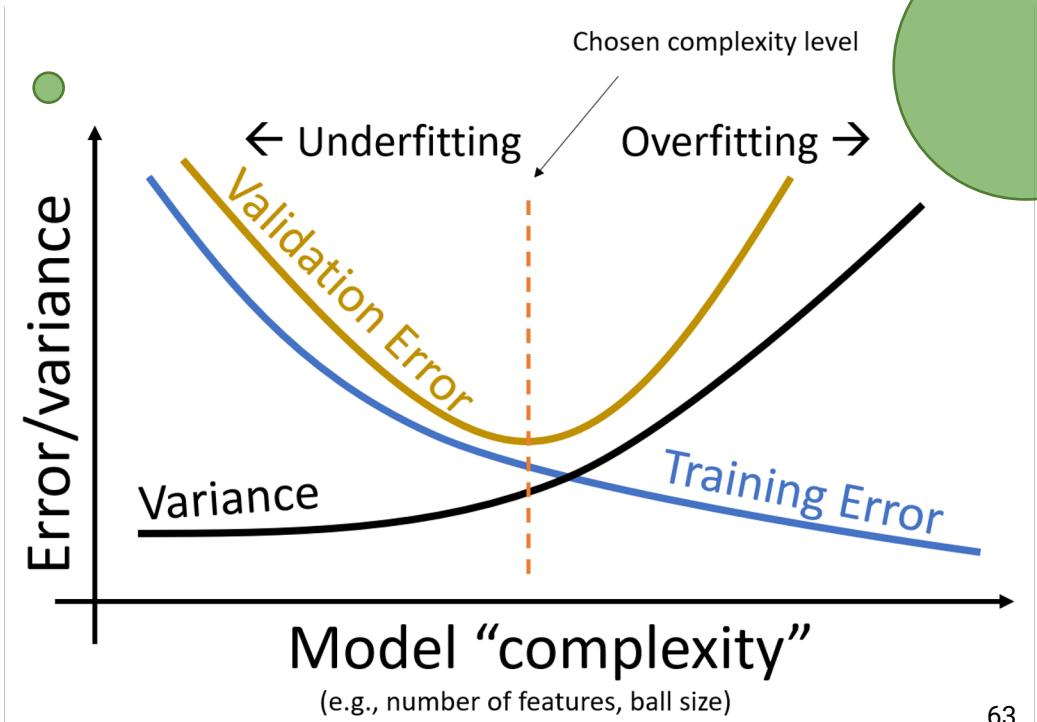
For very small ball size:

- Model behaves like a constant model. Can't actually use our 9 features!
- High **training error**, low variance, high **validation error**.

For very large ball size:

- Model behaves like OLS.
- **If we have tons of features**, results in overfitting. Low **training error**, high variance, high **validation error**.

hp	weight	displacement	hp^2	hp * weight	hp * displacement	weight^2	weight * displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	1180696.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2825.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0



L2 Regularization

Constraining our model's parameters to a ball around the origin is called **L2 Regularization**.

- The smaller the ball, the simpler the model.

Ordinary least squares. Find thetas that minimize:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_d \phi_{i,d}))^2$$

Ordinary least squares with **L2 regularization**. Find thetas that minimize:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_d \phi_{i,d}))^2$$

Such that θ_1 through θ_d live
inside a ball of radius Q.

L2 Regularization

Constraining our model's parameters to a ball around the origin is called **L2 Regularization**.

- The smaller the ball, the simpler the model.

Ordinary least squares. Find thetas that minimize:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_d \phi_{i,d}))^2$$

Ordinary least squares with **L2 regularization**. Find thetas that minimize:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_d \phi_{i,d}))^2$$

such that $\sum_{j=1}^d \theta_j^2 \leq Q$

Note, intercept term not included!

L2 Regularized Least Squares in sklearn

We can run least squares with an **L2 regularization term** by using the “Ridge” class.

```
from sklearn.linear_model import Ridge  
ridge_model = Ridge(alpha = 10000)  
ridge_model.fit(vehicle_data_with_squared_features, vehicle_data["mpg"])
```

Coefficients we get back:

```
ridge_model.coef_
```

```
array([-5.56414449e-02, -7.93804083e-03, -8.22081425e-02, -6.18785466e-04,  
       -2.55492157e-05,  9.47353944e-04,  7.58061062e-07,  1.07439477e-05,  
       -1.64344898e-04])
```

Note: sklearn’s “alpha” parameter is proportional to the inverse of the ball radius!

- Large alpha means small ball.

L2 Regularized Least Squares in sklearn

We can run least squares with an **L2 regularization term** by using the “Ridge” class.

```
from sklearn.linear_model import Ridge  
ridge_model = Ridge(alpha = 10**-5)  
ridge_model.fit(vehicle_data_with_squared_features, vehicle_data["mpg"])
```

For a tiny alpha, the coefficients are larger:

```
ridge_model.coef_
```

```
array([-1.35872588e-01, -1.46864458e-04, -1.18230336e-01, -4.03590098e-04,  
       -1.12862371e-05,  8.25179864e-04, -1.17645881e-06,  2.69757832e-05,  
       -1.72888463e-04])
```

Note: sklearn’s “alpha” parameter is proportional to the inverse of the ball radius!

- Small alpha means large ball.

L2 Regularized Least Squares in sklearn

We can run least squares with an **L2 regularization term** by using the “Ridge” class. For a tiny alpha, the coefficients are also about the same as a standard OLS model’s coefficients!

```
ridge_model.coef_
```

```
array([-1.35872588e-01, -1.46864458e-04, -1.18230336e-01, -4.03590098e-04,
       -1.12862371e-05,  8.25179864e-04, -1.17645881e-06,  2.69757832e-05,
       -1.72888463e-04])
```

```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(vehicle_data_with_squared_features, vehicle_data["mpg"])
linear_model.coef_
```

```
array([-1.35872588e-01, -1.46864447e-04, -1.18230336e-01, -4.03590097e-04,
       -1.12862370e-05,  8.25179863e-04, -1.17645882e-06,  2.69757832e-05,
       -1.72888463e-04])
```

Green ball includes the OLS solution!



Terminology Note

Why does sklearn use the word “Ridge”?

Because least squares with an **L2 regularization term** is also called “**Ridge Regression**”.

- Term is historical. Doesn’t really matter.

Problem 1: Find thetas that minimize:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_d \phi_{i,d}))^2$$

such that $\sum_{j=1}^d \theta_j^2 \leq Q$

Problem 2: Find thetas that minimize:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_d \phi_{i,d}))^2 + \alpha \sum_{j=1}^d \theta_j^2$$

The “objective function” that gradient descent is minimizing now has an extra term.

Intuitively, this extra **right term penalizes large thetas**.

Ridge Regression has a closed form solution which we will not derive.

- Note: The solution exists even if the feature matrix has collinearity between its columns.

$$\hat{\theta}_{ridge} = (\mathbb{X}^T \mathbb{X} + n\alpha I)^{-1} \mathbb{X}^T \mathbb{Y}$$


Identity matrix

- This solution exists **even if** \mathbb{X} is not full rank – an important reason why we often prefer L2 regularization.

Scaling Data for Regularization

Cross Validation

- The Holdout Method
- K-Fold Cross Validation
- Test Sets

Regularization

- L2 Regularization (Ridge)
- **Scaling Data for Regularization**
- L1 Regularization (LASSO)

One Issue With Our Approach

Our data from before has features of quite different numerical scale!

- Optimal theta for hp will probably be much further from origin than theta for weight².

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

Theta will tend to be smaller for weight² than other parameters

Coefficients from Earlier

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

ridge_model.coef_

```
array([-1.35872588e-01, -1.46864458e-04, -1.18230336e-01, -4.03590098e-04,
       -1.12862371e-05,  8.25179864e-04, -1.17645881e-06,  2.69757832e-05,
       -1.72888463e-04])
```

Ideally, our data should all be on the same scale.

- One approach: Standardize the data, i.e. replace everything with its Z-score.

$$z_k = \frac{x_k - \mu_k}{\sigma_k}$$

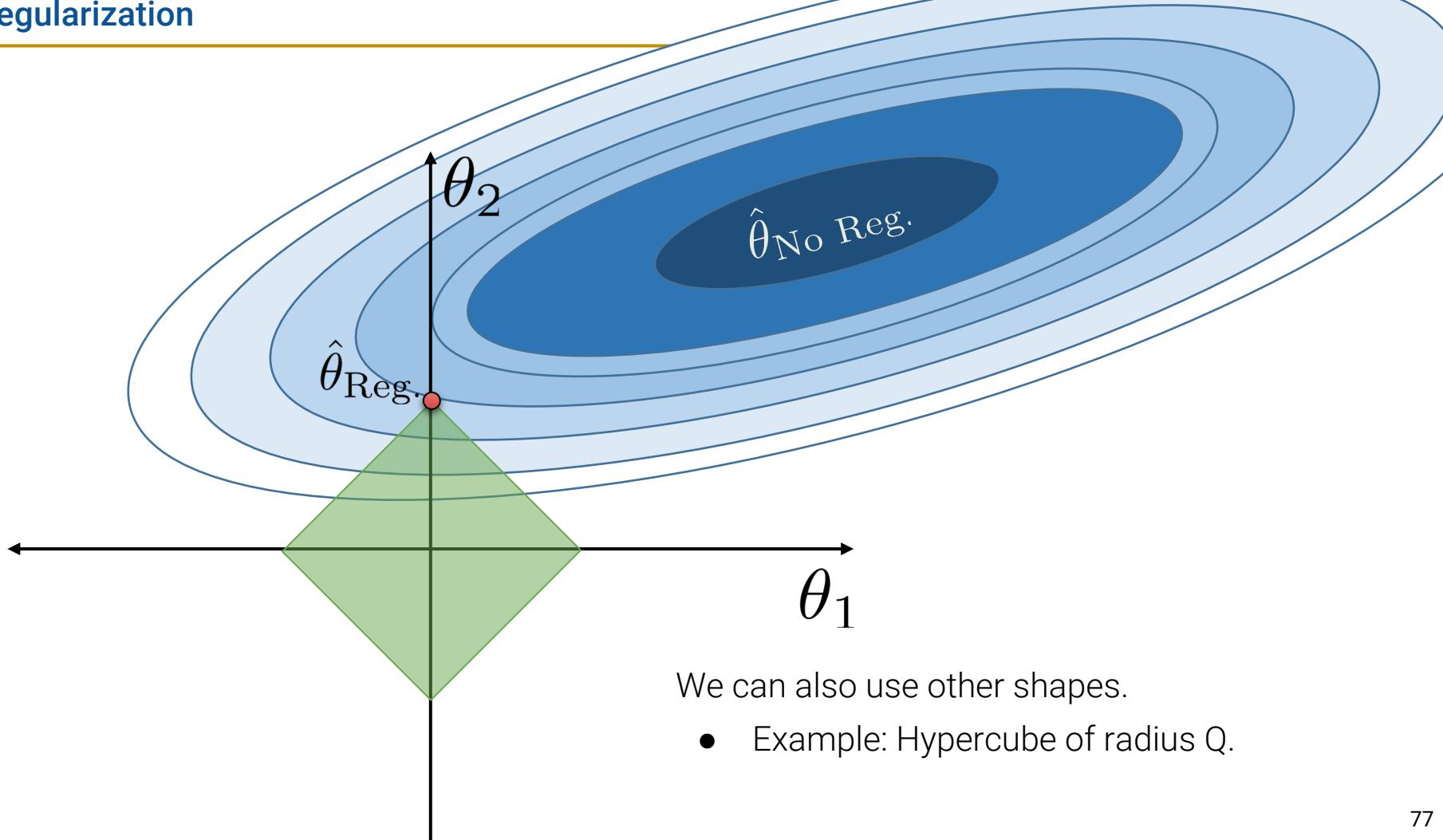
L1 Regularization (LASSO)

Cross Validation

- The Holdout Method
- K-Fold Cross Validation
- Test Sets

Regularization

- L2 Regularization (Ridge)
- Scaling Data for Regularization
- **L1 Regularization (LASSO)**



L1 Regularization in Equation Form

Using a hypercube is known as **L1 regularization**. Expressed mathematically in the two equivalent forms below:

Problem 1: Find thetas that minimize:

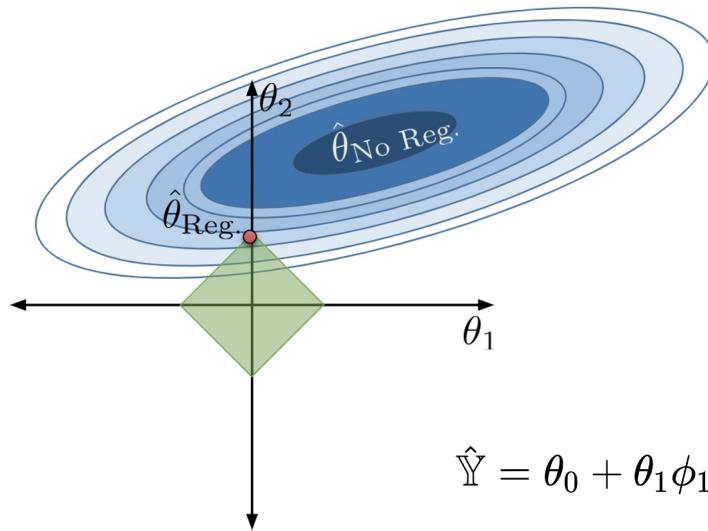
$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1\phi_1 + \dots + \theta_d\phi_d))^2 \quad \text{such that } \sum_{j=1}^d |\theta_j| \leq Q$$

Problem 2: Find thetas that minimize:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1\phi_1 + \dots + \theta_d\phi_d))^2 + \alpha \sum_{j=1}^d |\theta_j|$$

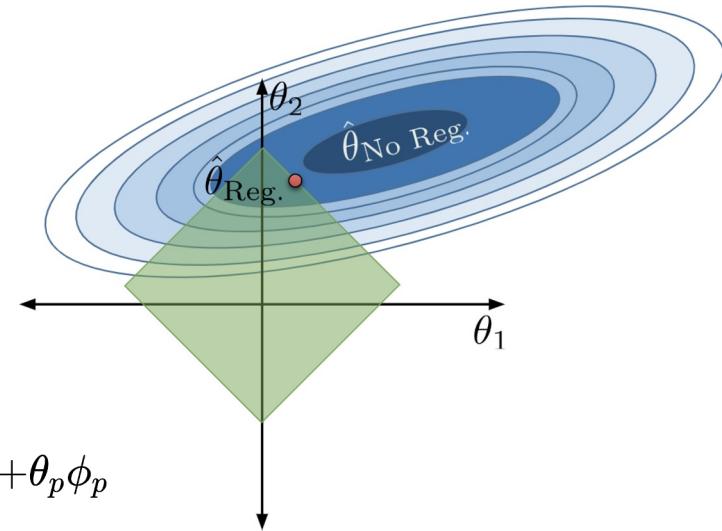
Size of Q

If we change the value of Q, the region of allowed parameter combinations changes.



$$\hat{Y} = \theta_0 + \theta_1\phi_1 + \theta_2\phi_2 \dots + \theta_p\phi_p$$

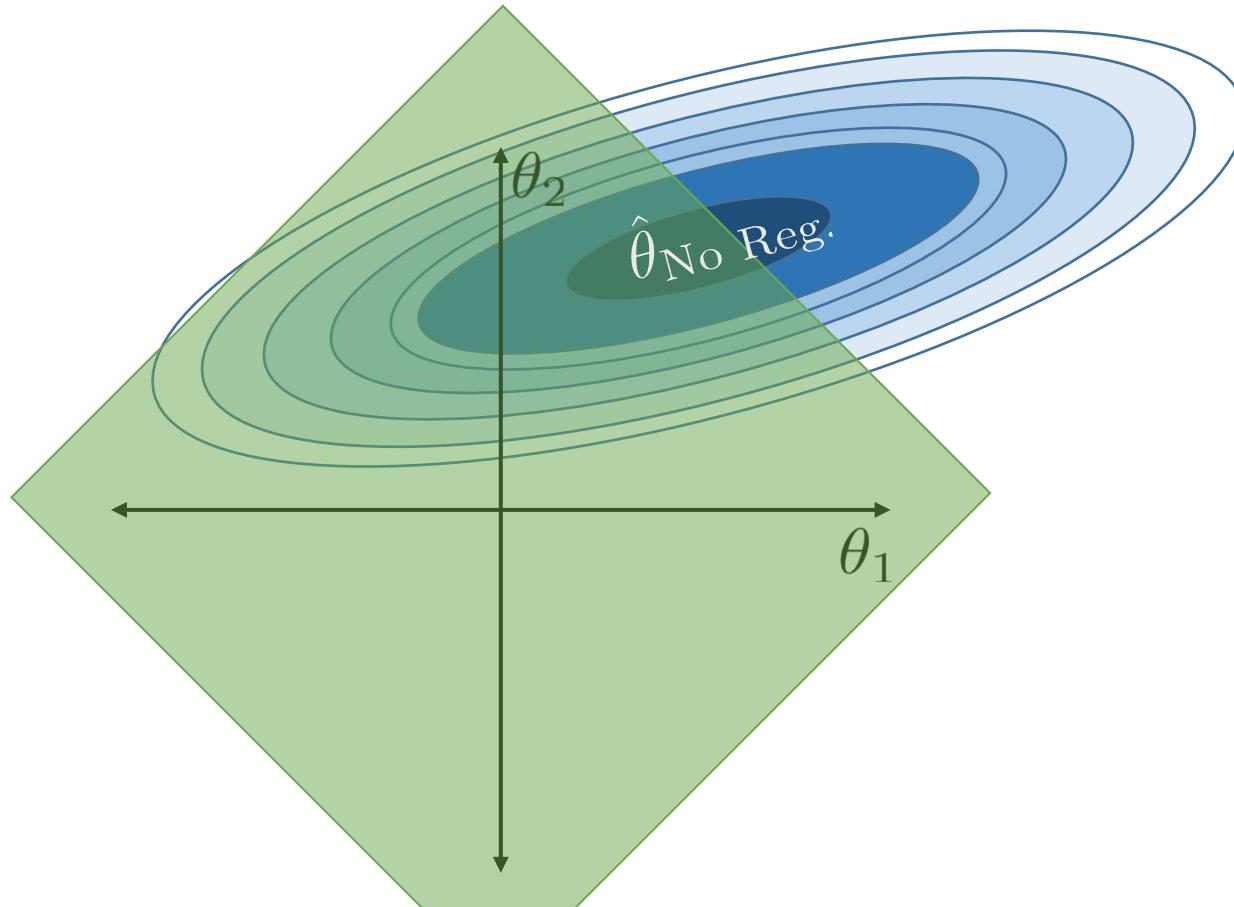
Small Q: θ_i are small in value; feature ϕ_i only contributes a little to the model → model becomes simpler.



Large Q: θ_i are large in value; feature ϕ_i contributes more to the model → model becomes more complex.

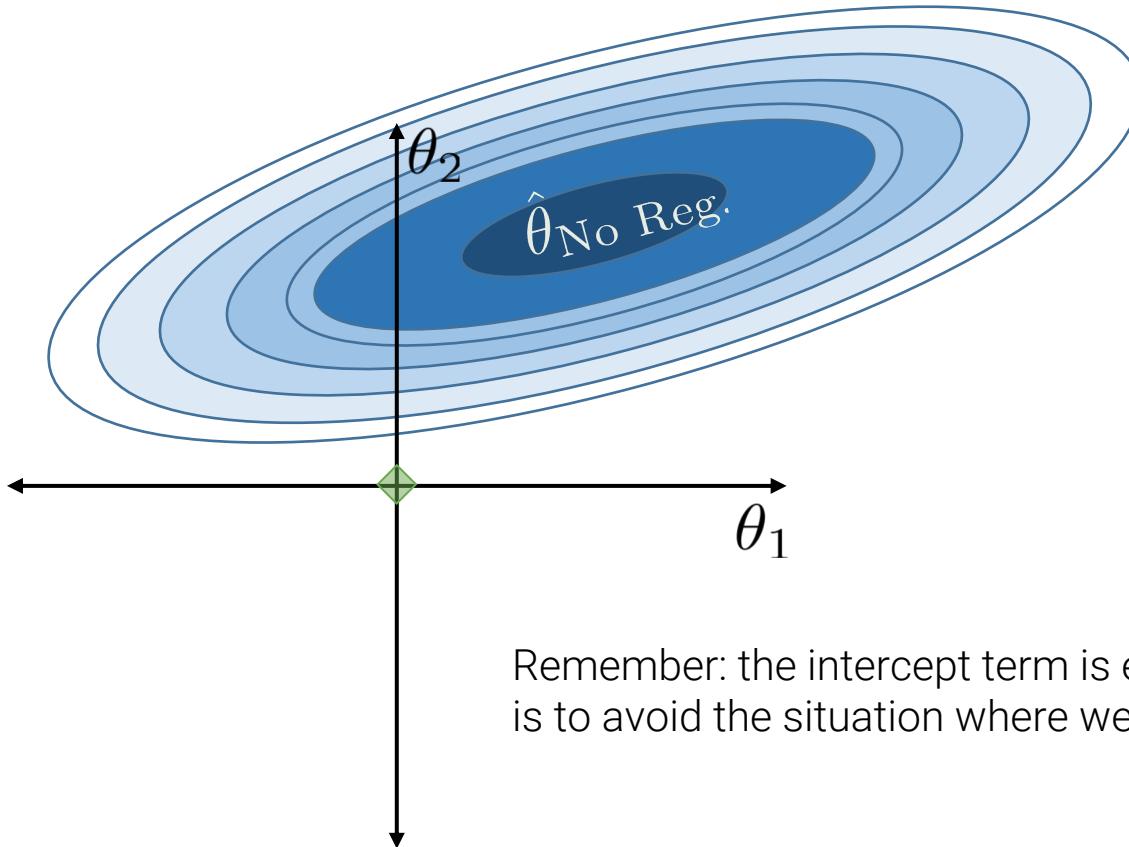
Size of Q

When Q is very large, our restriction essentially has no effect. The allowed region includes the OLS solution!



Size of Q

When Q is very small, parameters are set to (essentially) 0.



If the model has no intercept term:

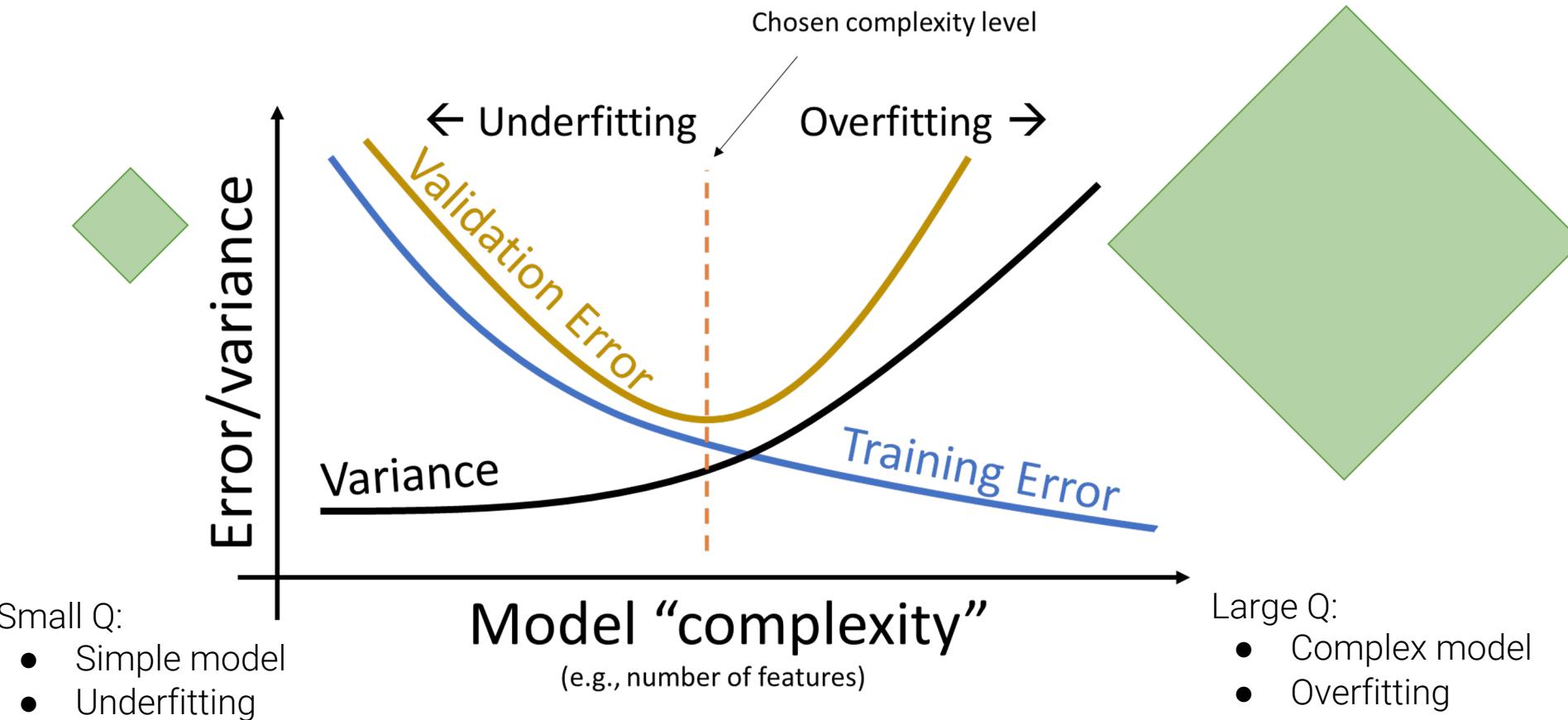
$$\hat{Y} = (0)\phi_1 + (0)\phi_2 + \dots = 0$$

If the model has an intercept term:

$$\hat{Y} = \theta_0 + (0)\phi_1 + (0)\phi_2 + \dots = \theta_0$$

Remember: the intercept term is excluded from the constraint – this is to avoid the situation where we always predict 0.

Complexity and Q



L1 Regularized OLS in sklearn

In sklearn, we use the Lasso module.

- Note: Performing OLS with L1 regularization is also called **LASSO regression**.

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha = 10**1)
lasso_model.fit(vehicle_data_with_squared_features, vehicle_data["mpg"])
lasso_model.coef_
```

```
array([-0.0000000e+00, -1.88104942e-02, -0.0000000e+00, -1.19625308e-03,
       8.84657720e-06,  8.77253835e-04,  3.16759194e-06, -3.21738391e-05,
      -1.29386937e-05])
```

The optimal parameters for a LASSO model tend to include a lot of zeroes! In other words, LASSO effectively selects only a subset of the features.

Summary of Regression Methods

Our regression models are summarized below.

- The “Objective” column gives the function that our gradient descent optimizer minimizes.
- Note that this table uses lambda instead of alpha for regularization strength. Both are common.

Name	Model	Loss	Reg.	Objective	Solution
OLS	$\hat{Y} = \mathbb{X}\theta$	Squared loss	None	$\frac{1}{n} Y - \mathbb{X}\theta _2^2$	$\hat{\theta}_{\text{OLS}} = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T Y$
Ridge Regression	$\hat{Y} = \mathbb{X}\theta$	Squared loss	L2	$\frac{1}{n} Y - \mathbb{X}\theta _2^2 + \lambda \sum_{j=1}^d \theta_i^2$	$\hat{\theta}_{\text{ridge}} = (\mathbb{X}^T \mathbb{X} + n\lambda I)^{-1} \mathbb{X}^T Y$
LASSO	$\hat{Y} = \mathbb{X}\theta$	Squared loss	L1	$\frac{1}{n} Y - \mathbb{X}\theta _2^2 + \lambda \sum_{j=1}^d \theta_i $	No closed form