

# Cross-validation and Regularization

Introduction to Data Science  
Spring 1403

Yadollah Yaghoobzadeh

# Goals for this Lecture

Circling back to two questions raised in the modeling lectures:

- How does model performance change on "seen" vs. "unseen" data?
- How do we control model complexity?

# Agenda

## Cross-Validation

- Training, Test, and Validation Sets
- K-Fold Cross-Validation

## Regularization

- Constraining Model Parameters
- L2 Regularization (Ridge)
- L1 Regularization (LASSO)

# Training, Test, and Validation Sets

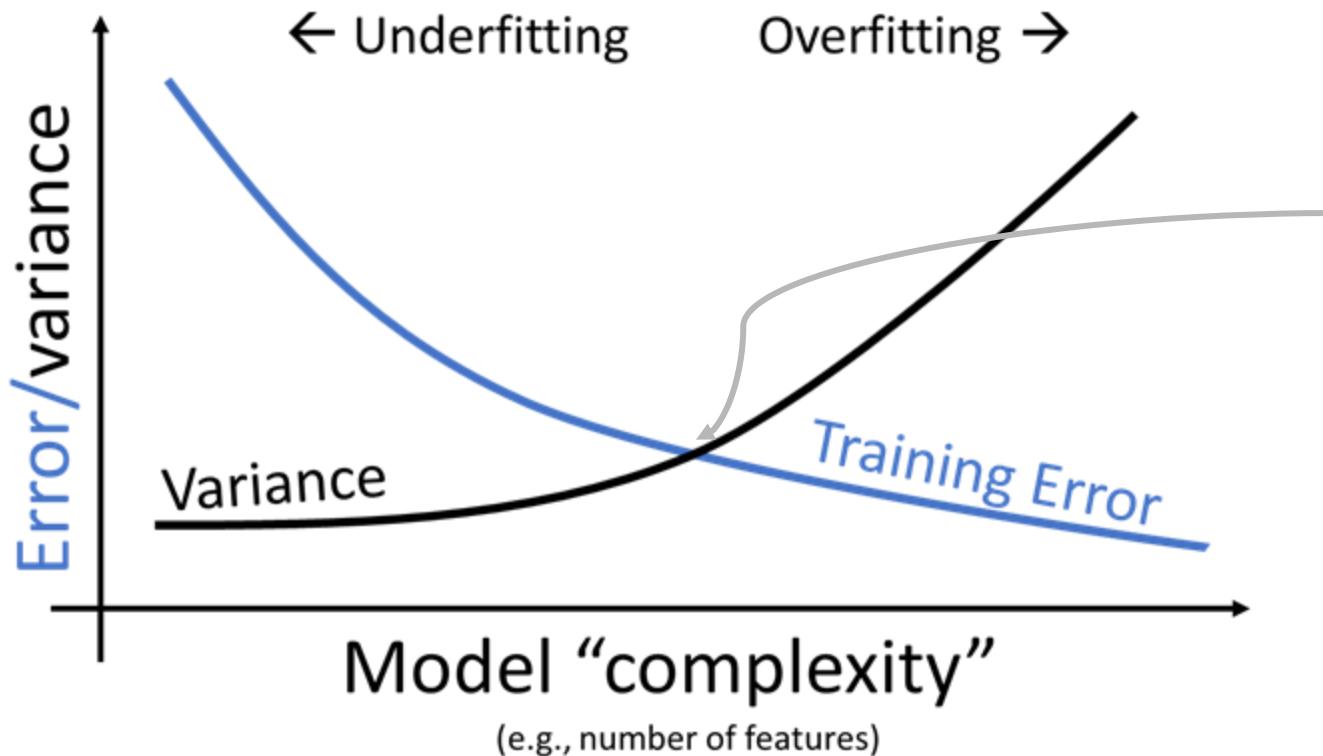
## Cross-Validation

- **Training, Test, and Validation Sets**
- K-Fold Cross-Validation

## Regularization

- Constraining Model Parameters
- L2 Regularization (Ridge)
- L1 Regularization (LASSO)

# Where We Left Things



Our goal: find this “sweet spot”  
Stay tuned for ~30 min from now

# Where We Left Things

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

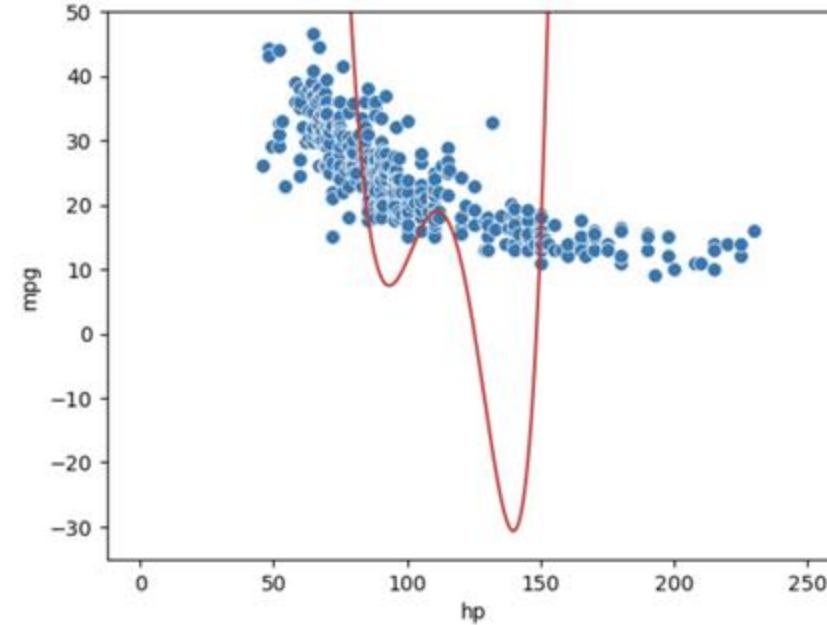
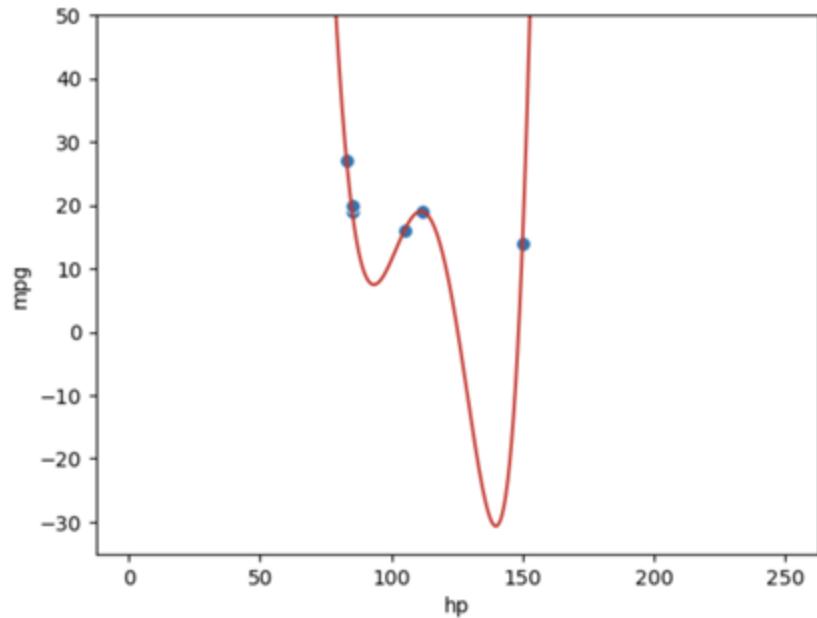
- Formalize the idea of training and test data.
- Introduce a method to "preview" how a model will perform on unseen data.

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.



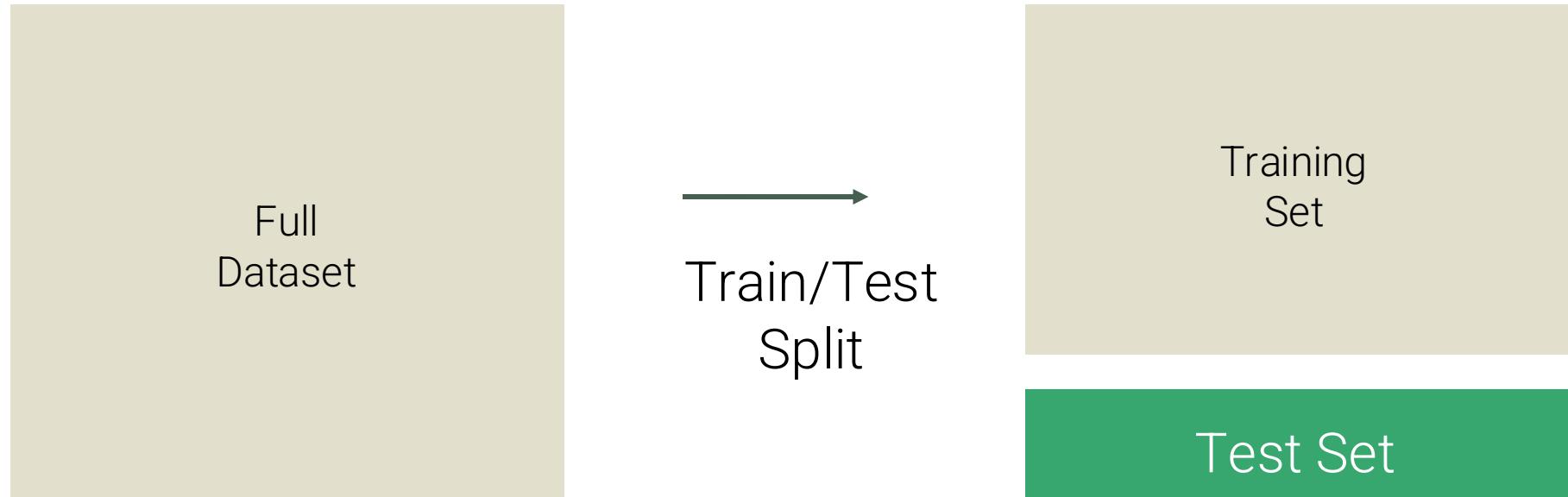
How to quantify performance on this "unseen" data? Introduce a **test set**.

# 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?

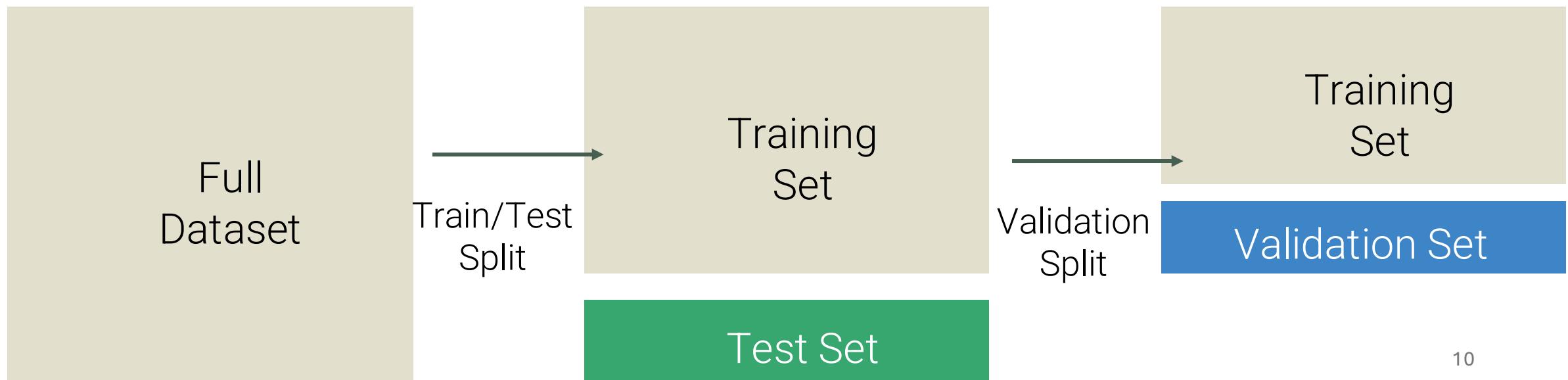
In our current framework, we'd be stuck – we can't then go back and adjust our model, because we'd be *factoring in information from the test set* to design our model. The test set would no longer represent performance on unseen data.

**Solution:** introduce a **validation set**.

# Validation Sets

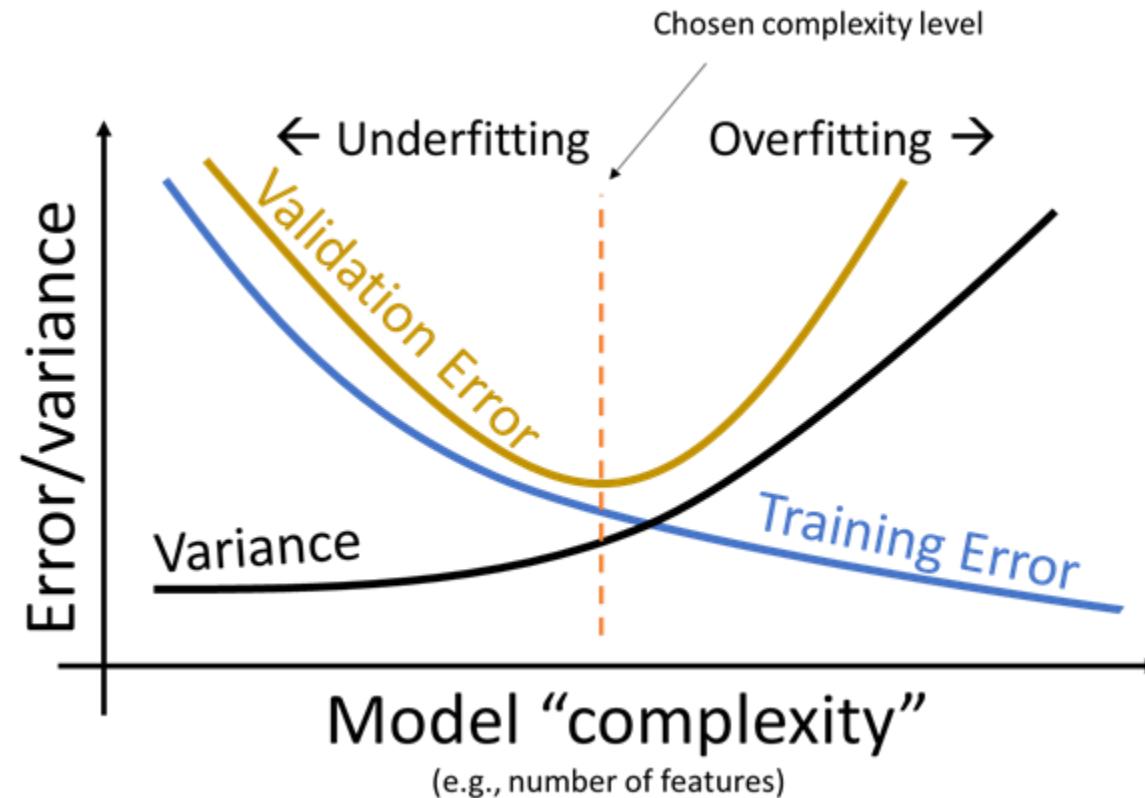
A **validation set** is a portion of our *training set* that we set aside for assessing model performance while it is *still being developed*.

- 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.

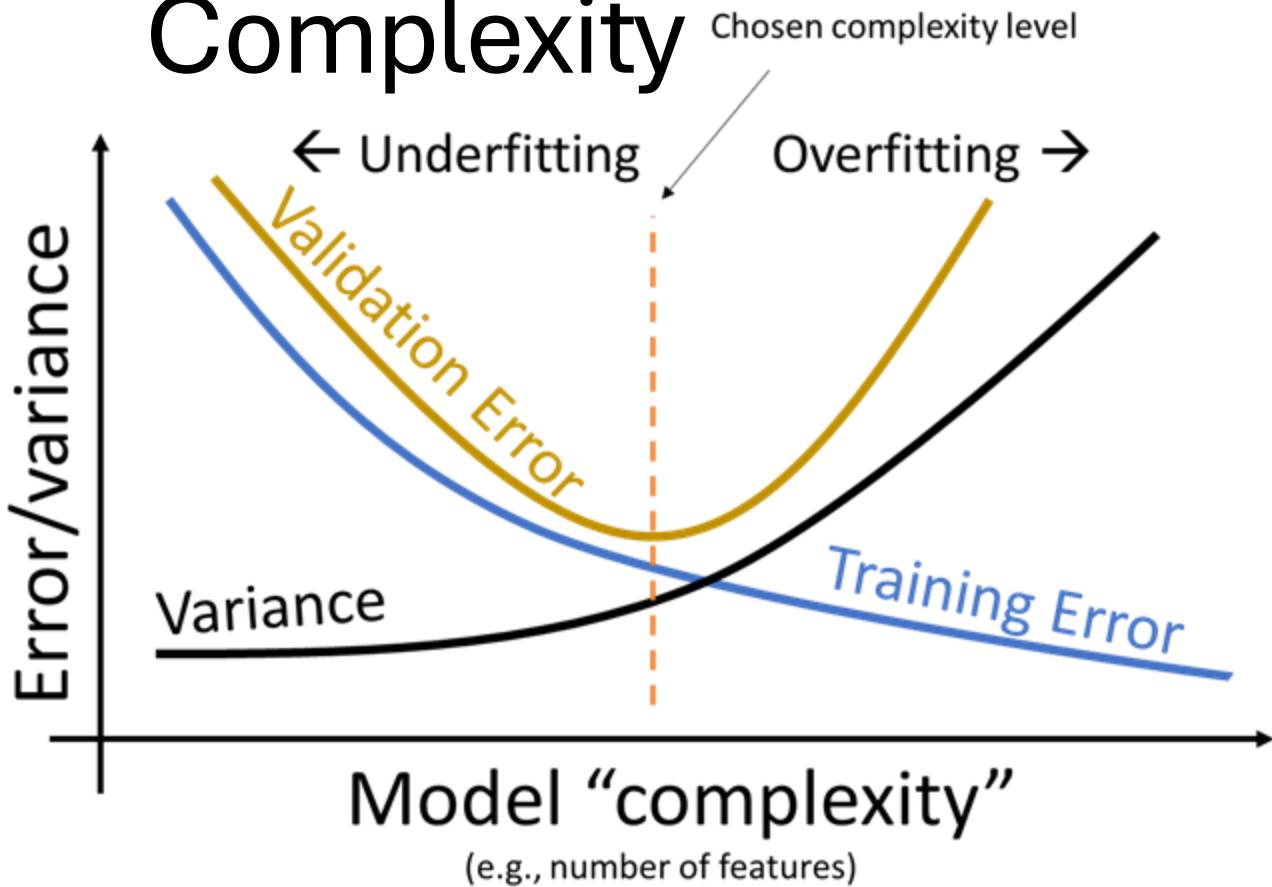


# Updating Our Understanding of Model Complexity

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



# Updating Our Understanding of Model Complexity



In Lectures 17 and 18, we'll learn the mathematical origins of these relationships

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

- Training, Test, and Validation Sets
- **K-Fold Cross-Validation**

## Regularization

- Constraining Model Parameters
- L1 Regularization (LASSO)
- L2 Regularization (Ridge)

# 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.

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

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

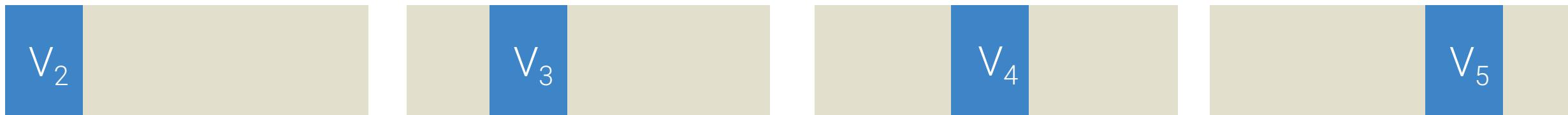
# Validation Folds

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  
validation  
folds

$V_1$   
error #1

Compute  
MSE on  
Validation  
error #2

Validation  
error #2

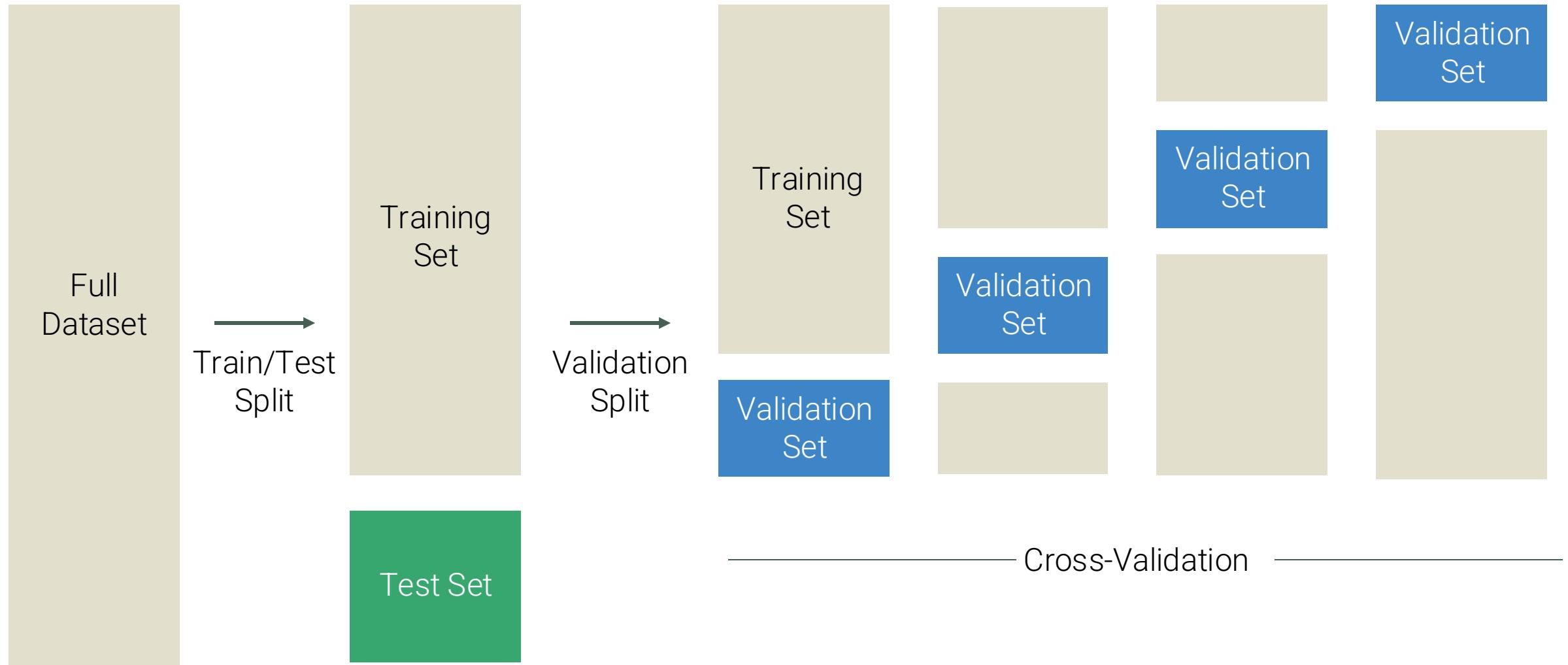
Cross-validation error = mean of validation  
errors #1 to #5

Validation  
error #3

Validation  
error #4

Validation  
error #5

# Model Selection Workflow



# Hyperparameters

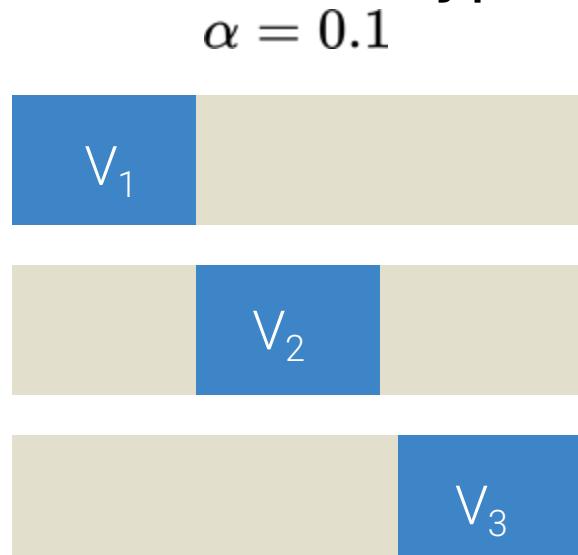
Cross-validation is often used for **hyperparameter** selection.

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

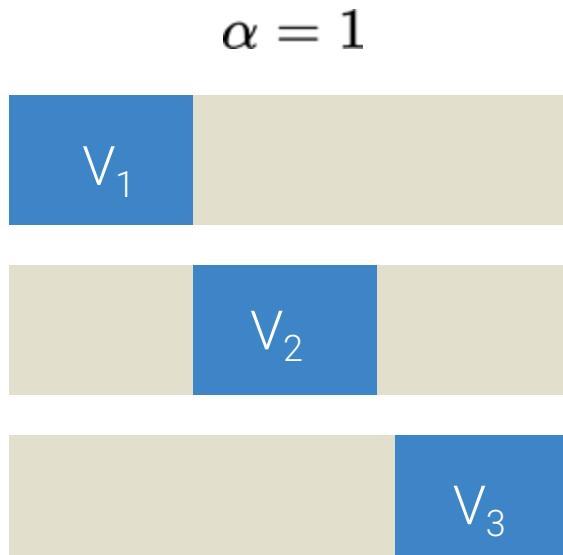
- Cannot solve for hyperparameters via calculus, OLS, gradient descent, etc – we must choose it ourselves.
- Examples
  - Degree of polynomial model
  - Gradient descent learning rate,
  - Regularization penalty, (to be introduced later this lecture)

# Hyperparameter Tuning

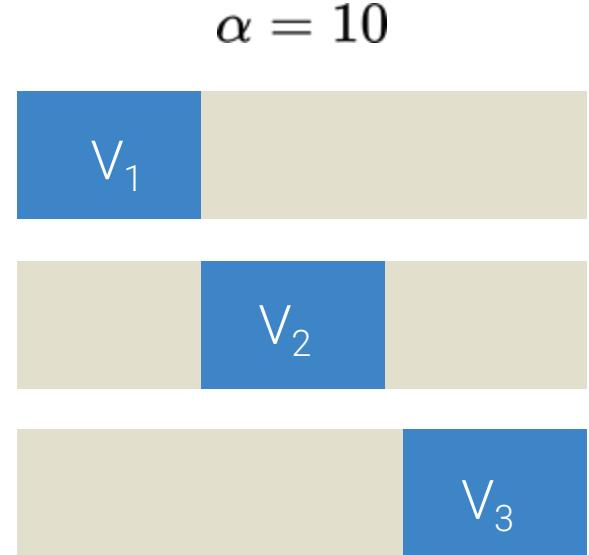
- To select a hyperparameter value via cross-validation:
- List out several different “guesses” for the best hyperparameter.
- For each guess, run cross-validation to compute the CV error for that choice of hyperparameter value.
- Select the hyperparameter value with lowest CV error.



CV error:  
4.67



CV error:  
7.01



CV error: 10.22

# Constraining Model Parameters

## Cross-Validation

- Training, Test, and Validation Sets
- K-Fold Cross-Validation

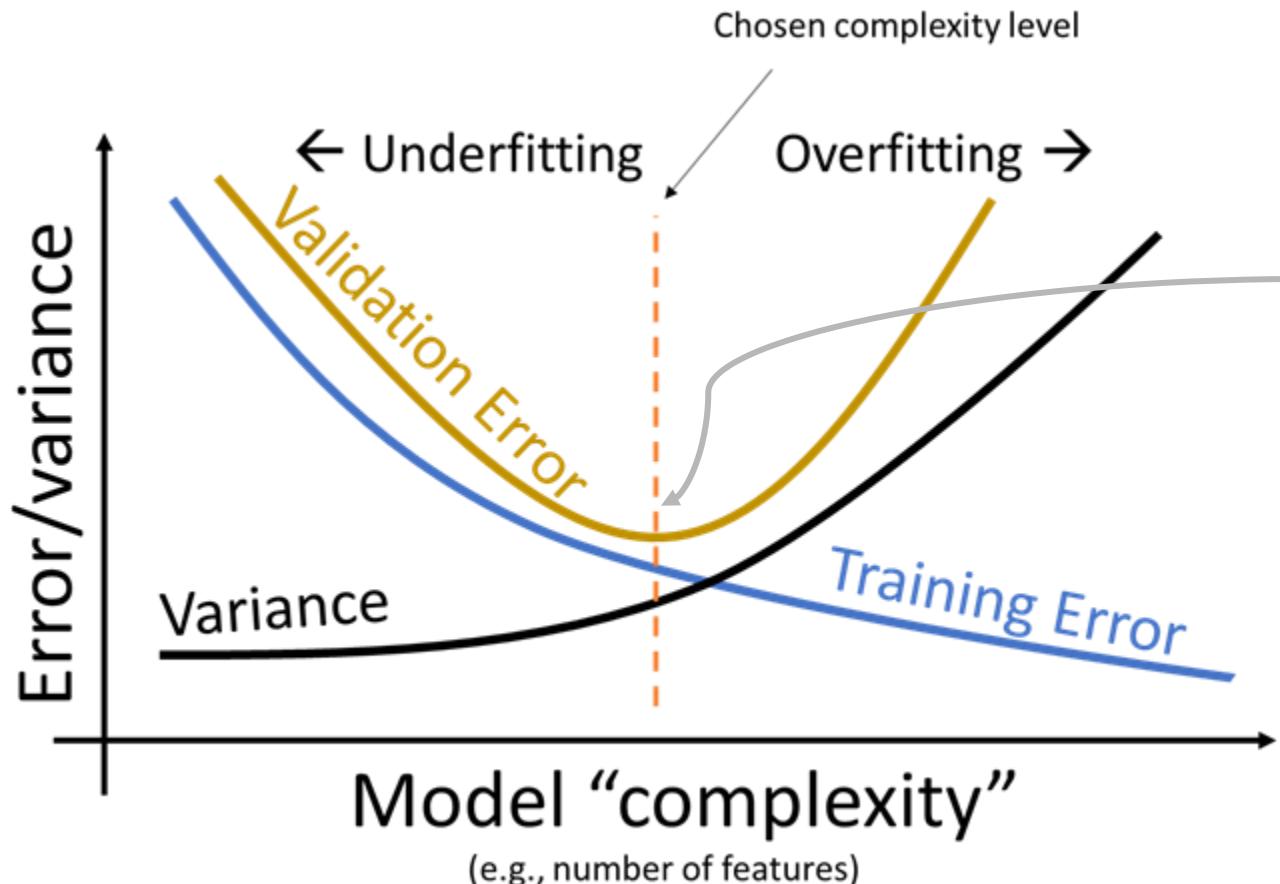
## Regularization

- **Constraining Model Parameters**
- L2 Regularization (Ridge)
- L1 Regularization (LASSO)

# Restricting Model Complexity

We've seen now that model complexity needs to be chosen carefully:

- Too complex: model overfits – "memorizes" training data too closely.
- Too simple: model underfits – does not take full advantage of the features available in the dataset.



Our goal: find this "sweet spot"

How do we control complexity?

# 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  $\theta_i$  can be, we restrict how much each feature contributes to the model.
- When  $\theta_i$  is close to or equal to 0, the model decreases in complexity because feature  $\phi_i$  barely impacts the prediction.

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

# Restricting Model Complexity

---

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

Example: Suppose we specify that the sum of all absolute parameters can be no larger than some number Q

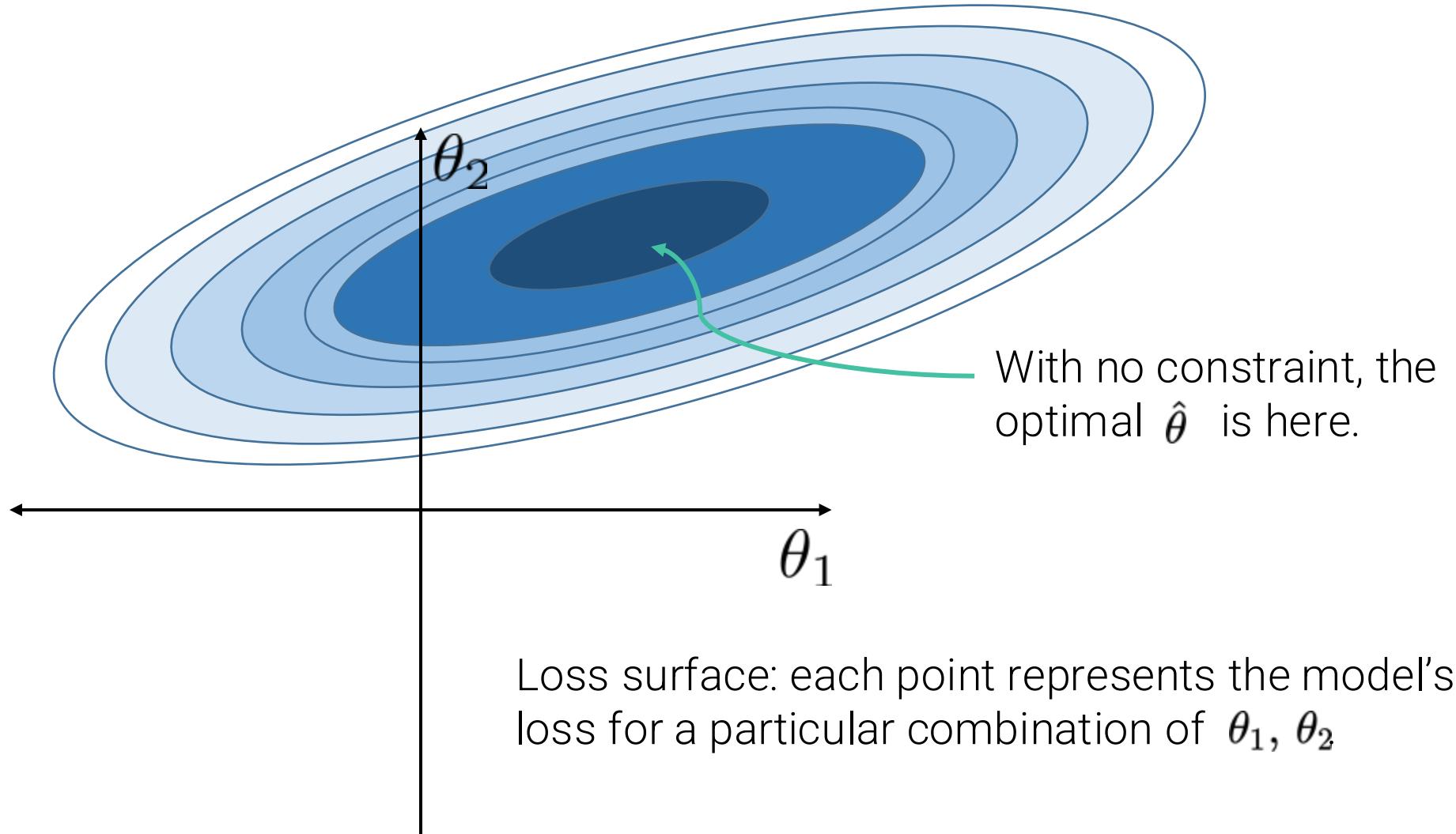
$$\sum_{i=1}^p |\theta_i| \leq Q$$

We've given the model a "budget" for how much weight to assign to each feature. Some parameters  $\theta_i$  will need to be small in value so the sum remains below Q.

Note that the intercept term,  $\theta_0$ , is typically excluded from this constraint.

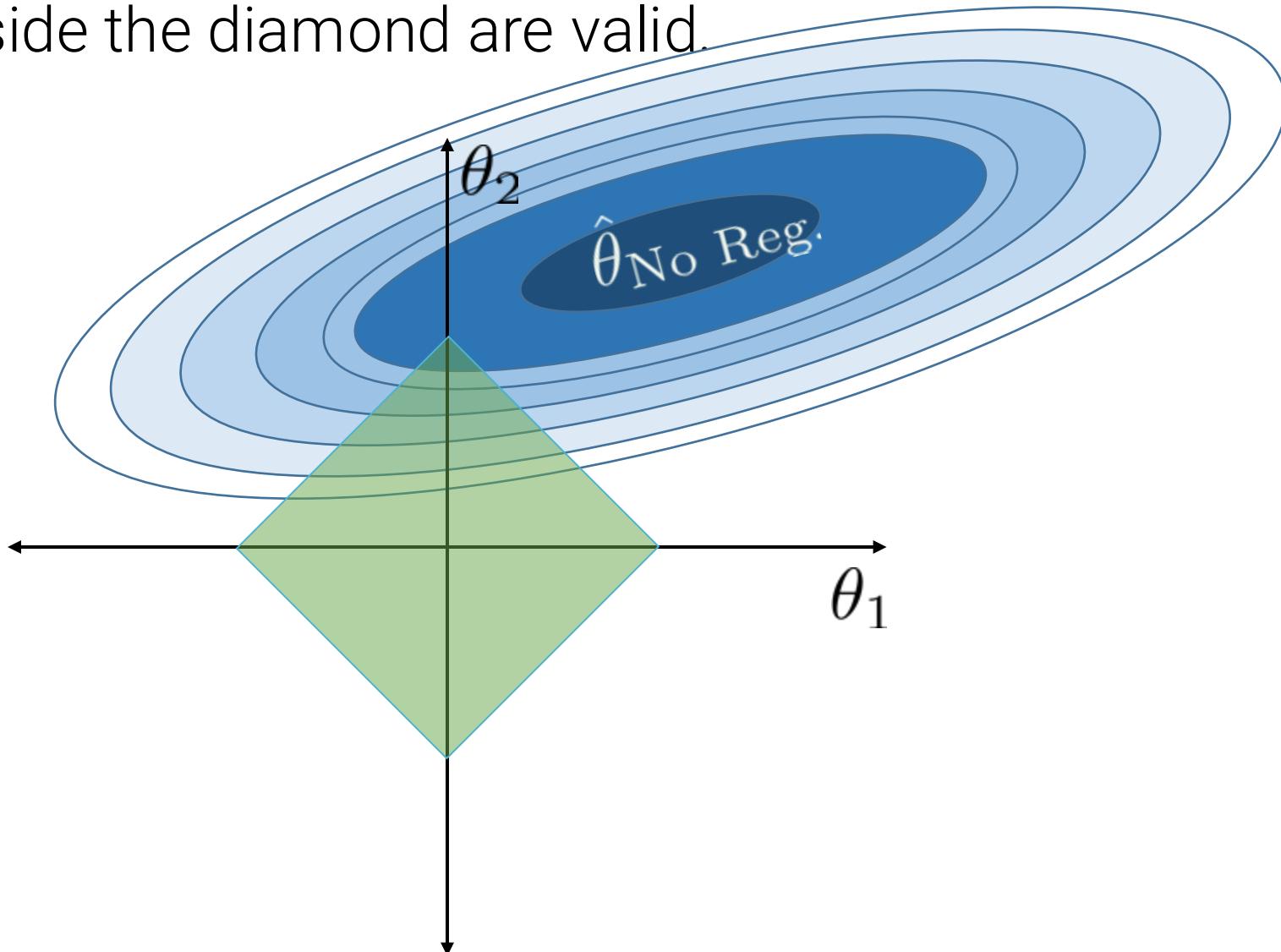
# Think Back to Gradient Descent

In gradient descent: visualize the loss surface as a contour map. Our goal is to find the combination of parameters that gives the lowest loss.



# Constraining Model Parameters

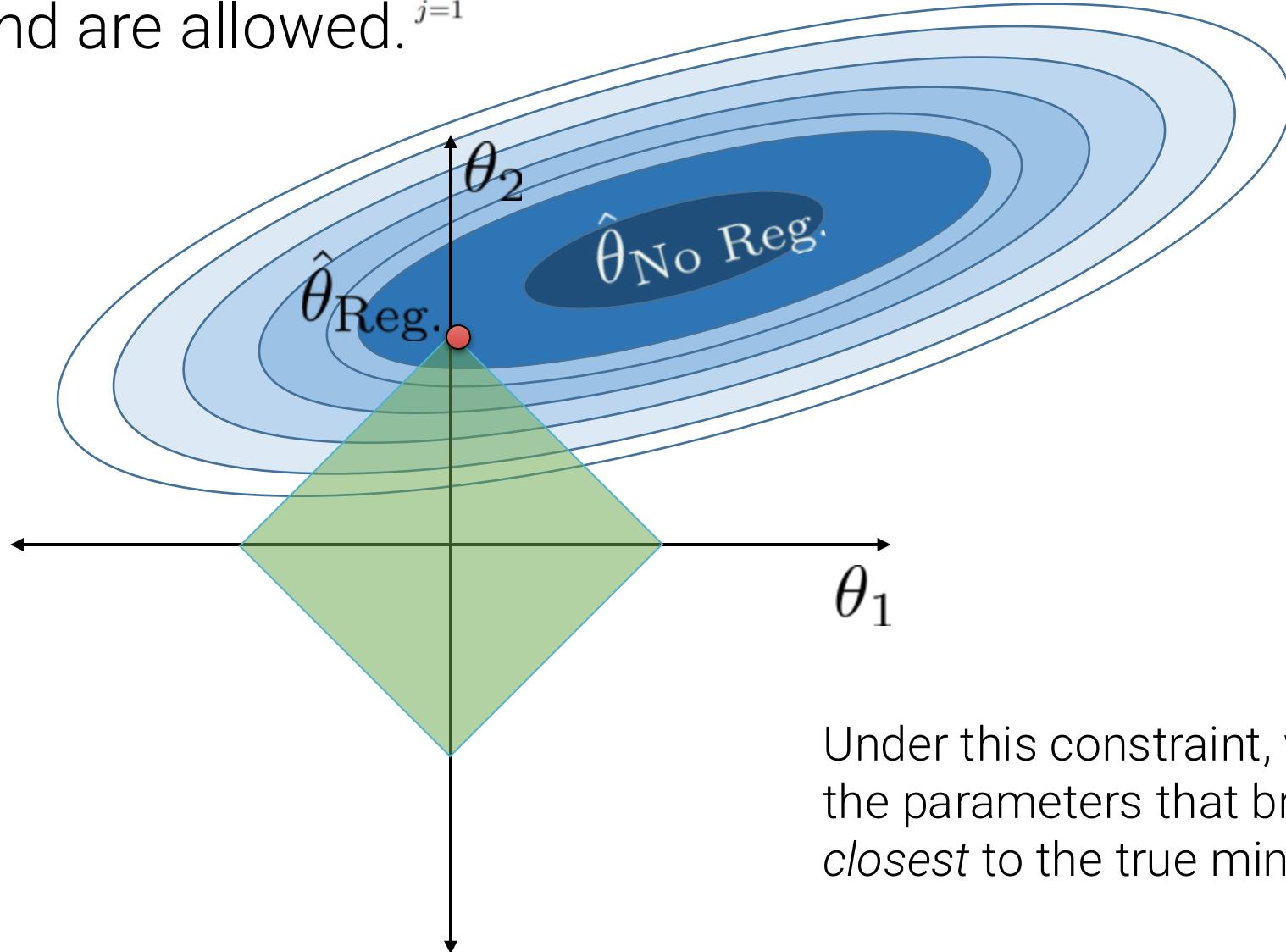
When we apply the constraint  $\sum_{i=1}^p |\theta_i| \leq Q$ , only parameter combinations inside the diamond are valid.



$Q$  is some arbitrary number.

# Constraining Model Parameters

When we apply the constraint  $\sum_{j=1}^d |\theta_j| \leq Q$ , only parameter combinations inside the diamond are allowed.



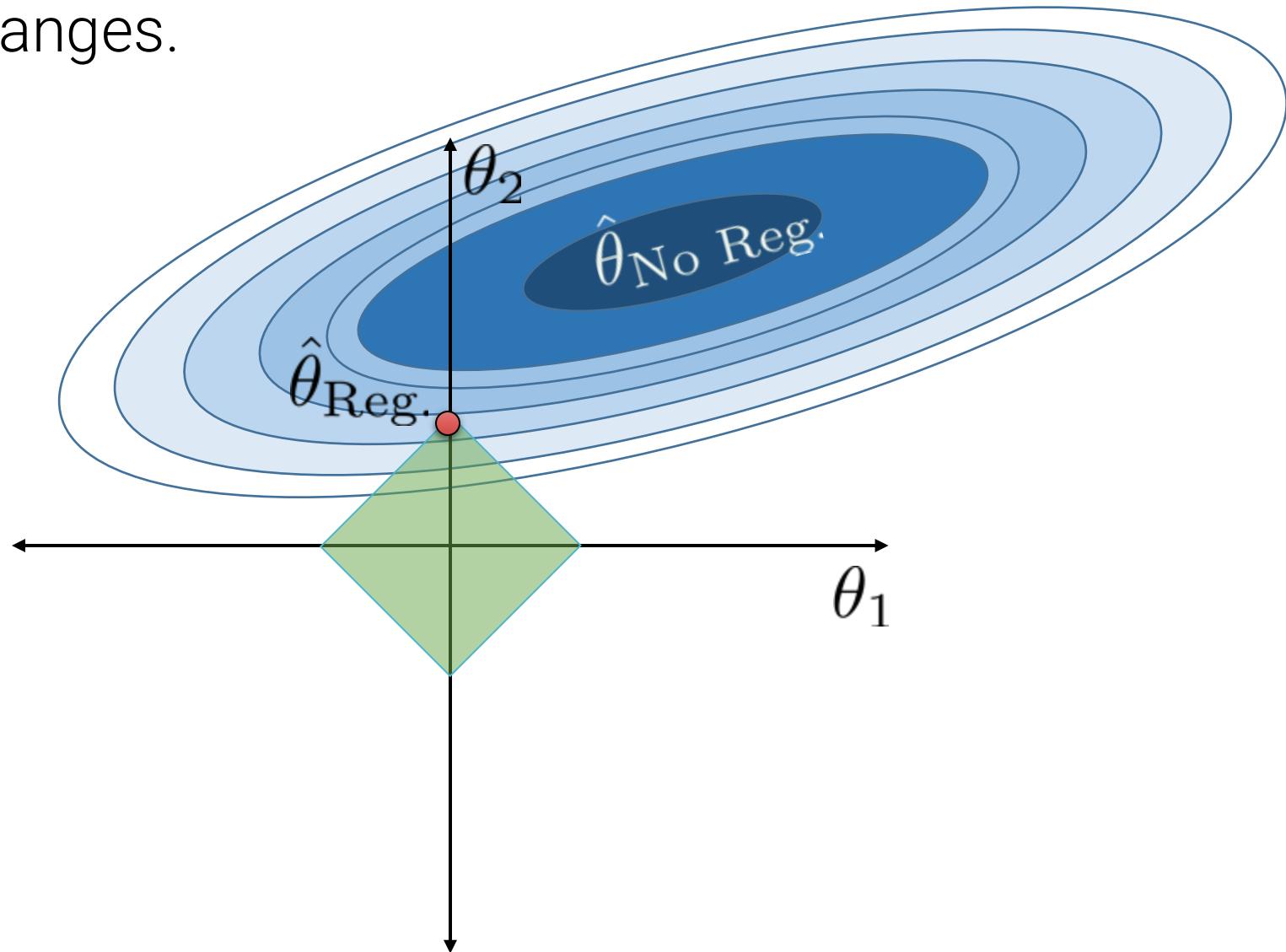
$Q$  is some arbitrary number.

Under this constraint, we choose the parameters that bring us *closest* to the true minimum loss.

# Smaller Q

---

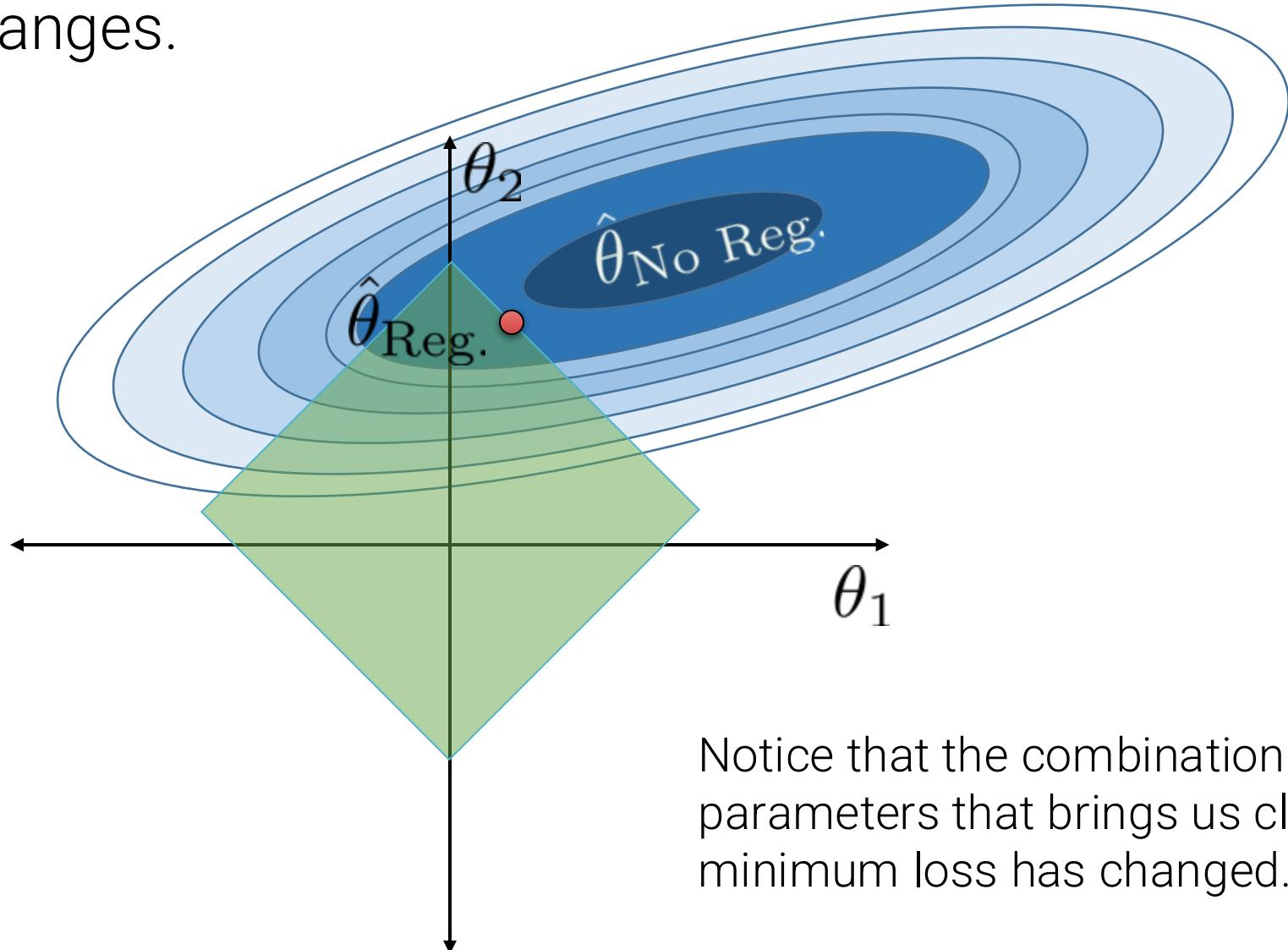
If we change the value of  $Q$ , the region of allowed parameter combinations changes.



# Larger Q

---

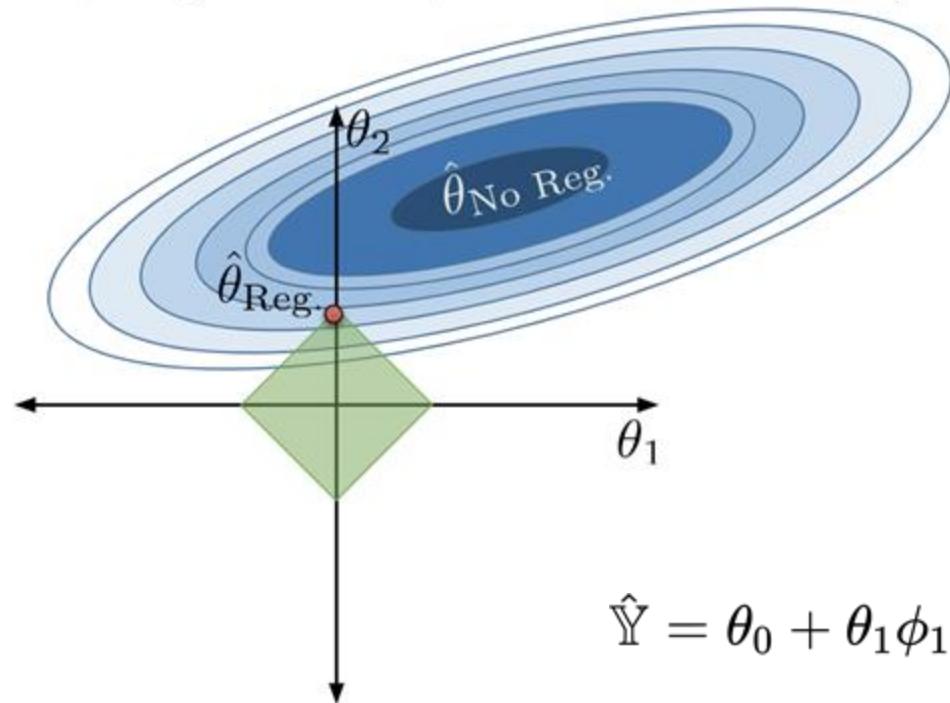
If we change the value of Q, the region of allowed parameter combinations changes.



**How does the size of Q  
relate to model complexity?**

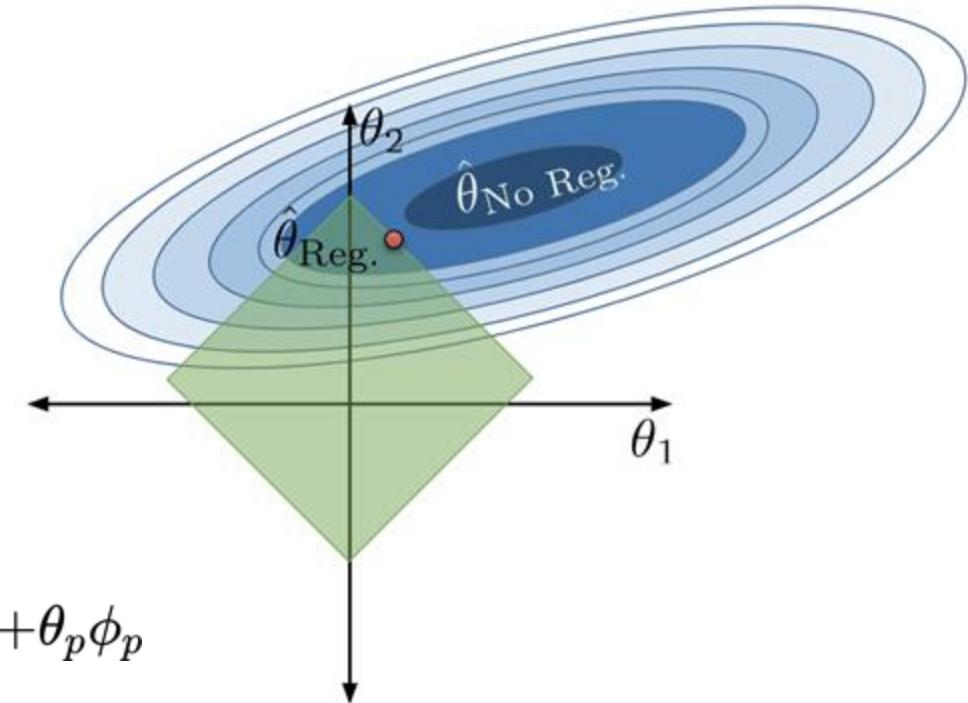
# Size of Q

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:  $\theta_i$  are small in value; feature  $\phi_i$  only contributes a little to the model → model becomes simpler.

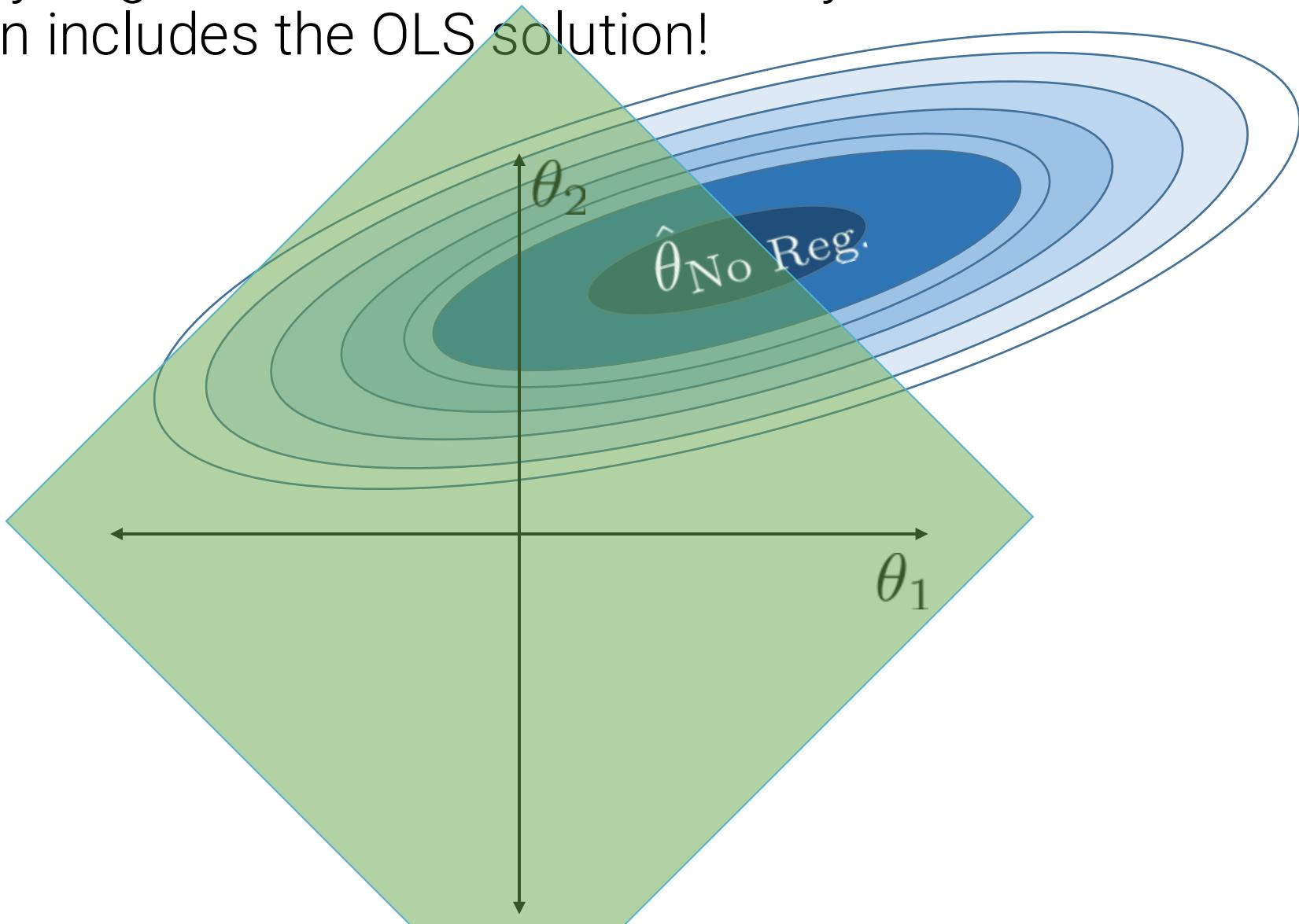


Large Q:  $\theta_i$  are large in value; feature  $\phi_i$  contributes more to the model → model becomes more complex.

# Size of Q

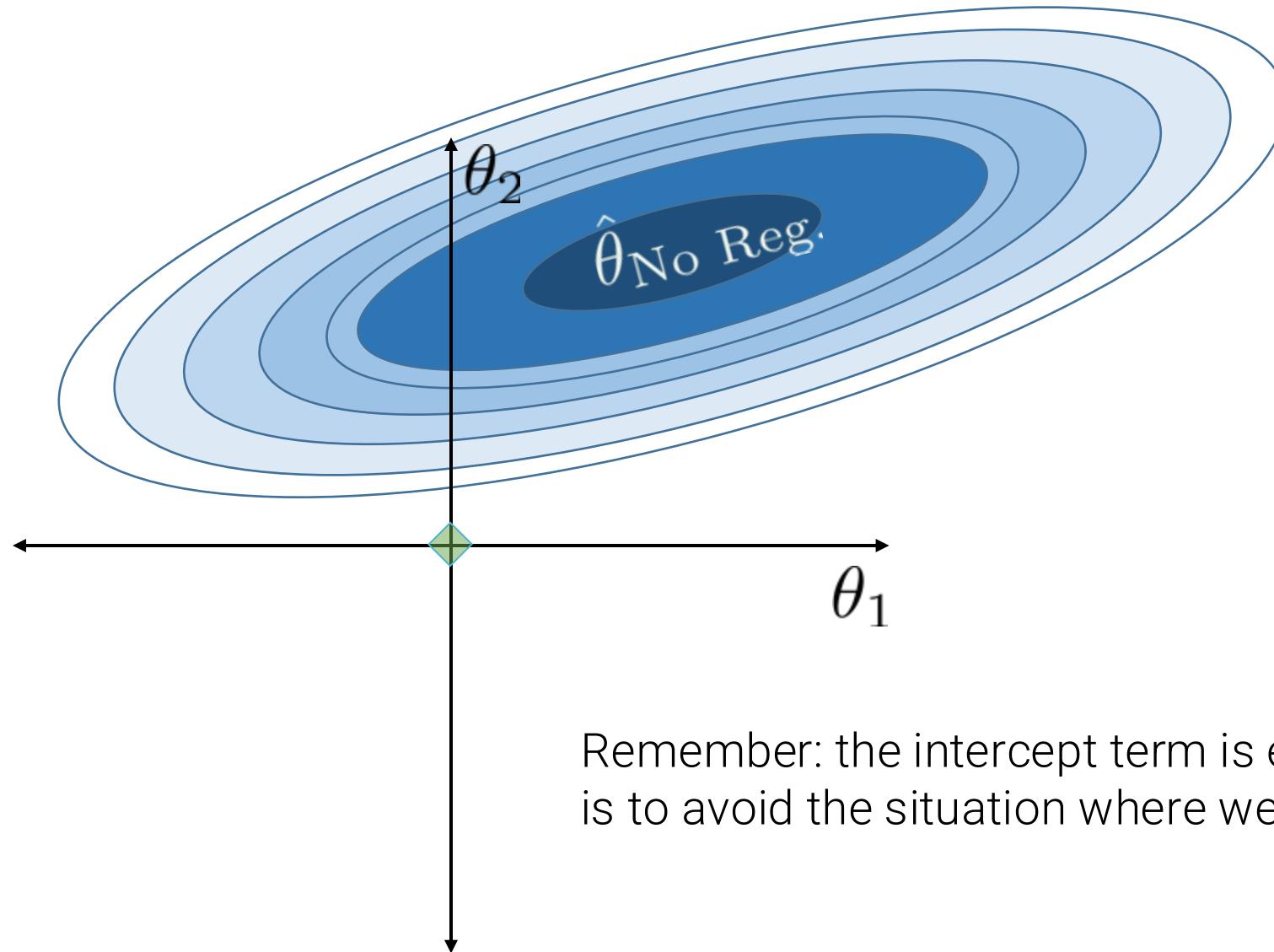
---

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.



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

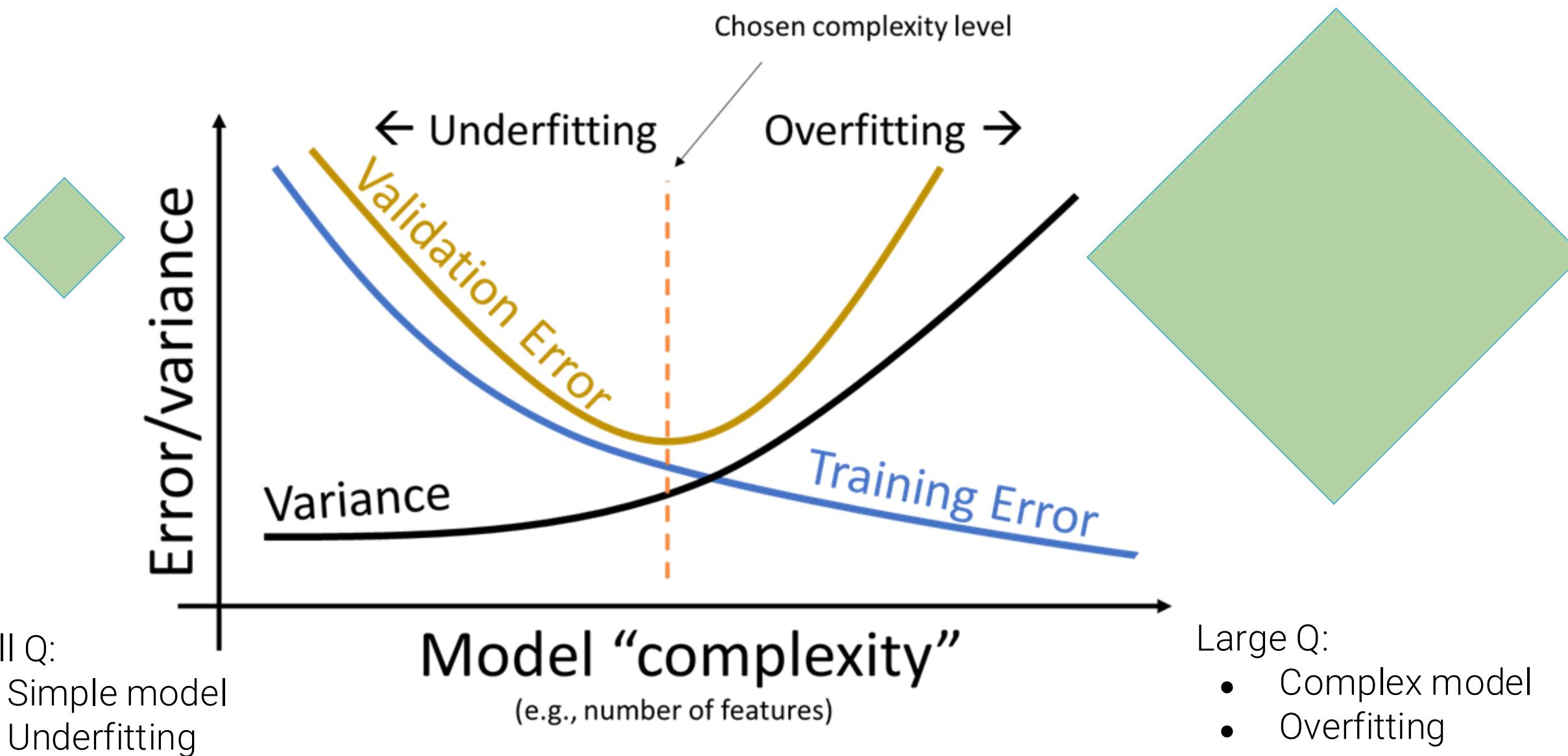
If the model has no intercept term:

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

If the model has an intercept term:

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

# Complexity and Q



# L1 Regularization (LASSO)

## Cross-Validation

- Training, Test, and Validation Sets
- K-Fold Cross-Validation

## Regularization

- Constraining Model Parameters
- **L1 Regularization (LASSO)**
- L2 Regularization (Ridge)

# L1 Regularization

---

How do we actually apply our constraint  $\sum_{i=1}^p |\theta_i| \leq Q$  ?

Recall our OLS framework: Find thetas that minimize the objective function:

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_p \phi_{i,p}))^2$$

In **L1 regularization**: Find thetas that minimize the objective function:

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_p \phi_{i,p}))^2 \quad \text{such that} \quad \sum_{i=1}^p |\theta_i| \leq Q$$

# L1 Regularization

---

By the Lagrangian Duality\*, these two problems are equivalent.

Our original problem: find thetas that minimize the objective function:

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_p \phi_{i,p}))^2 \quad \text{such that} \quad \sum_{i=1}^p |\theta_i| \leq Q$$

Equivalent problem: find thetas that minimize the **augmented objective function**:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_p \phi_{i,p}))^2 + \lambda \sum_{i=1}^p |\theta_i|$$

# L1 Regularization

In L1 regularization, we find thetas that minimize our **new objective function**:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_p \phi_{i,p}))^2 + \lambda \sum_{i=1}^p |\theta_i|$$

Keep MSE on the  
data low...

...while also keeping the  
size of parameters small

$\lambda$  is the **regularization penalty hyperparameter**. When  $\lambda$  is large, our objective function is penalized more for choosing larger thetas → model will adjust by reducing thetas and decreasing complexity.

- How to choose the value for  $\lambda$ ? Cross-validation!

L1 regularization is also called LASSO: "least absolute shrinkage and selection operator".

# LASSO and Feature Selection

---

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.

- We often use L1 regularization for **feature selection** – the features with non-zero parameters are more informative for modeling than those with parameters set to zero.
- Intuition: We can get closer to the lowest loss contour at a corner of our constraint diamond.

LASSO: “least absolute **shrinkage** and **selection** operator”

Shrink parameter  
sizes

Select important  
features



# One Issue With Our Approach

---

Our dataset has features with wildly different numerical scales!

	hp	hp^2	hp^3	hp^4	hp^5	hp^6	hp^7	hp^8
72	150.0	22500.0	3375000.0	5.062500e+08	7.593750e+10	1.139062e+13	1.708594e+15	2.562891e+17
89	150.0	22500.0	3375000.0	5.062500e+08	7.593750e+10	1.139062e+13	1.708594e+15	2.562891e+17
92	158.0	24964.0	3944312.0	6.232013e+08	9.846580e+10	1.555760e+13	2.458100e+15	3.883799e+17
124	180.0	32400.0	5832000.0	1.049760e+09	1.889568e+11	3.401222e+13	6.122200e+15	1.101996e+18
88	137.0	18769.0	2571353.0	3.522754e+08	4.826172e+10	6.611856e+12	9.058243e+14	1.240979e+17
...	...	...	...	...	...	...	...	...
2	150.0	22500.0	3375000.0	5.062500e+08	7.593750e+10	1.139062e+13	1.708594e+15	2.562891e+17
104	167.0	27889.0	4657463.0	7.777963e+08	1.298920e+11	2.169196e+13	3.622558e+15	6.049671e+17
159	148.0	21904.0	3241792.0	4.797852e+08	7.100821e+10	1.050922e+13	1.555364e+15	2.301939e+17
180	115.0	13225.0	1520875.0	1.749006e+08	2.011357e+10	2.313061e+12	2.660020e+14	3.059023e+16
394	52.0	2704.0	140608.0	7.311616e+06	3.802040e+08	1.977061e+10	1.028072e+12	5.345973e+13

# Coefficients From Earlier

	hp	hp^2	hp^3	hp^4	hp^5	hp^6	hp^7	hp^8
72	150.0	22500.0	3375000.0	5.062500e+08	7.593750e+10	1.139062e+13	1.708594e+15	2.562891e+17
89	150.0	22500.0	3375000.0	5.062500e+08	7.593750e+10	1.139062e+13	1.708594e+15	2.562891e+17
92	158.0	24964.0	3944312.0	6.232013e+08	9.846580e+10	1.555760e+13	2.458100e+15	3.883799e+17
124	180.0	32400.0	5832000.0	1.049760e+09	1.889568e+11	3.401222e+13	6.122200e+15	1.101996e+18
88	137.0	18769.0	2571353.0	3.522754e+08	4.826172e+10	6.611856e+12	9.058243e+14	1.240979e+17
...	...	...	...	...	...	...	...	...
2	150.0	22500.0	3375000.0	5.062500e+08	7.593750e+10	1.139062e+13	1.708594e+15	2.562891e+17
104	167.0	27889.0	4657463.0	7.777963e+08	1.298920e+11	2.169196e+13	3.622558e+15	6.049671e+17
159	148.0	21904.0	3241792.0	4.797852e+08	7.100821e+10	1.050922e+13	1.555364e+15	2.301939e+17
180	115.0	13225.0	1520875.0	1.749006e+08	2.011357e+10	2.313061e+12	2.660020e+14	3.059023e+16
394	52.0	2704.0	140608.0	7.311616e+06	3.802040e+08	1.977061e+10	1.028072e+12	5.345973e+13

```
lasso_model.coef_
```

```
array([-5.14100640e-01,  1.16422594e-03,  2.70209864e-06, -8.05153574e-10,
       -2.78280269e-11, -1.02040718e-13, -5.44295812e-17,  1.83589942e-18])
```

# Pitfalls of Unscaled Data

Our model parameter for a feature with small numeric values (hp) is much, much larger than the parameter for a feature with large numeric values ( $hp^8$ ).

- The feature with larger values will naturally contribute more to the predicted  $y_{\text{hat}}$  for each observation.
- The LASSO model needs to “spend” more of its parameter budget to allow  $hp$  to have much of an impact on each prediction.

First datapoint:  $\hat{y}_i = \theta_0 + \theta_1(150) + \dots + \theta_8(2.56 \times 10^{17})$

 The large values of  $hp^8$  dominate the prediction.

$$\hat{y}_i = \theta_0 - 0.51(150) + \dots + 1.84 \times 10^{-18}(2.56 \times 10^{17})$$

 The parameter for  $hp$  must be very large for  $hp$  to influence the prediction.

# Making Things Fair

---

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

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

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

- Resulting features will be all on the same scale with mean 0 and SD 1.

# L2 Regularization (Ridge)

## Cross-Validation

- Training, Test, and Validation Sets
- K-Fold Cross-Validation

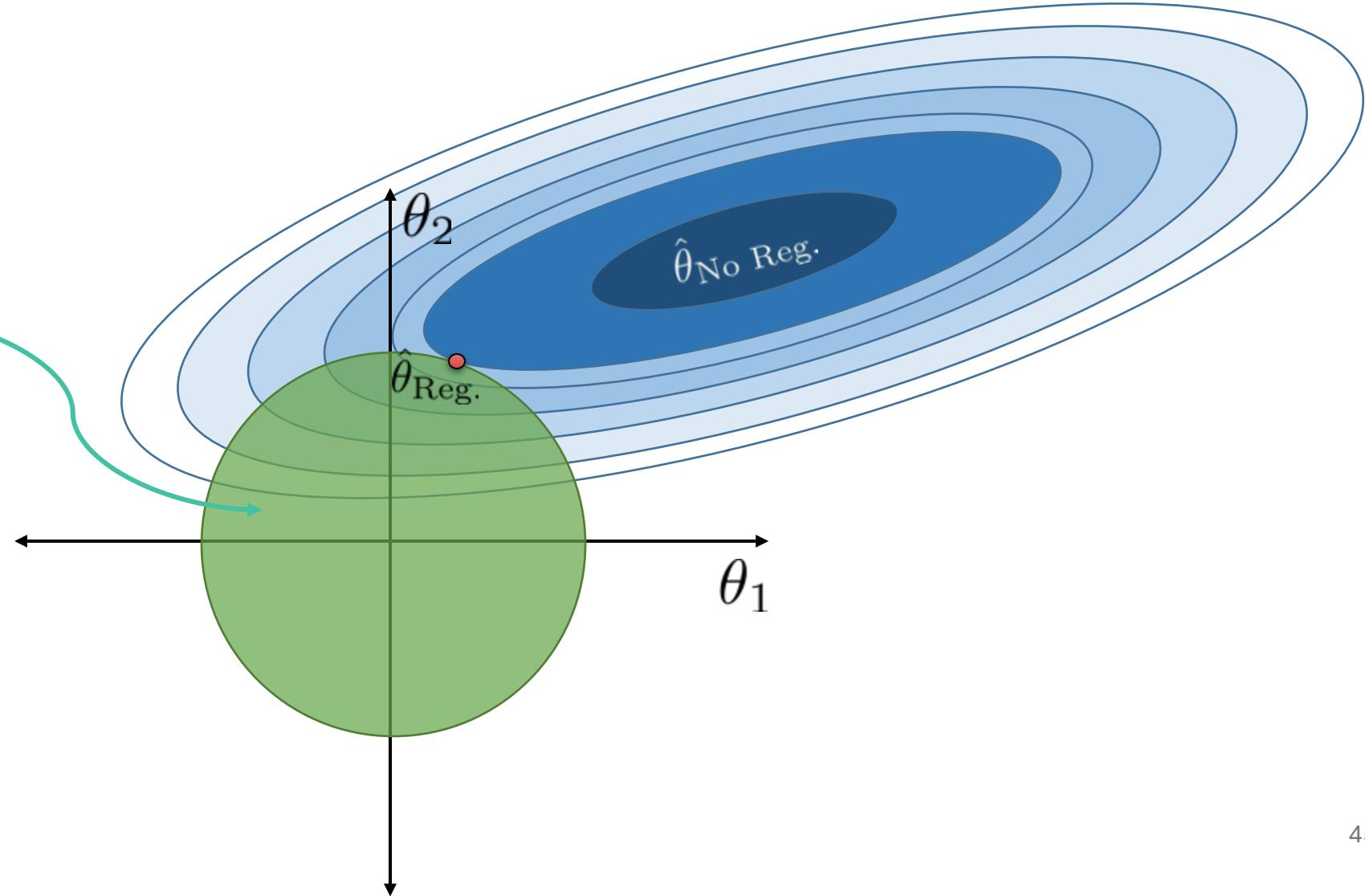
## Regularization

- Constraining Model Parameters
- L1 Regularization (LASSO)
- **L2 Regularization (Ridge)**

# Changing The Constraint

We could have applied a different constraint to our parameters: the sum of their squares must be less than some number Q.

$$\sum_{i=1}^p \theta_i^2 \leq Q$$



# L2 Regularization

---

As with L1 regularization, we can express this constraint in two forms:

Original formulation:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_p \phi_{i,p}))^2 \quad \text{such that} \quad \sum_{i=1}^p \theta_i^2 \leq Q$$

L2 objective function:

$$\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1 \phi_{i,1} + \dots + \theta_p \phi_{i,p}))^2 + \lambda \sum_{i=1}^p \theta_i^2$$

# L2 Regularization

In L2 regularization, we find thetas that minimize our **new objective function**:

$$\underbrace{\frac{1}{n} \sum_{i=1}^n (y_i - (\theta_0 + \theta_1\phi_{i,1} + \dots + \theta_p\phi_{i,p}))^2}_{\text{Keep MSE on the data low...}} + \lambda \underbrace{\sum_{i=1}^p \theta_i^2}_{\text{...while also keeping the size of parameters small}}$$

L2 regularization is commonly called **ridge regression**.

# Regularization for other models

- We can add the same L1 and L2 regularizations any loss function
  - Logistic regression
  - Neural networks