LECTURE 31

Cross Validation

Methods for ensuring the generalizability of our models to unseen data.

.

CSCI 3022, Fall 2023

Maribeth Oscamou

Content credit: Acknowledgments

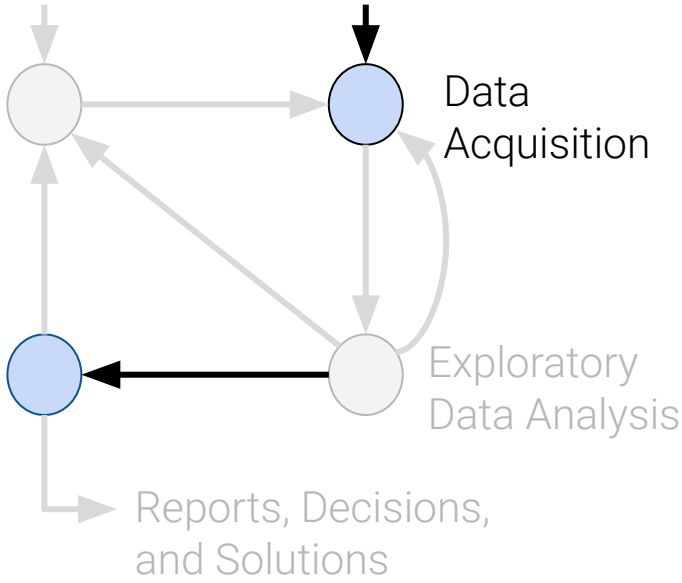
Announcements

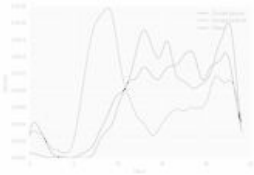
Project Part 1 due this Thursday at 11:59pm MT (plan accordingly - no late submissions accepted)

Last quiz (quiz 10) Friday

Project Part 2 released tomorrow - due next Thursday

Plan for This Week: Model Selection





(today)

Model Selection Basics:
Cross Validation

Probability II:
Bias and Variance
Inference/Multicollinearity

Today's Roadmap

Cross Validation

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

Review: Dataset

Today we will use the `mpg` dataset from the `seaborn` library.

The dataset has 392 rows and 9 column. Our task is to use some of the columns and their transformations to predict the value of the `mpg` column.

	mpg	cylinders	displacement	hp	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford ranger
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s-10

392 rows × 9 columns

Model Performance on Unseen Data

Our `vehicle` models from before considered a somewhat artificial scenario – we trained the models on the *entire* dataset, then evaluated their ability to make predictions on this same dataset

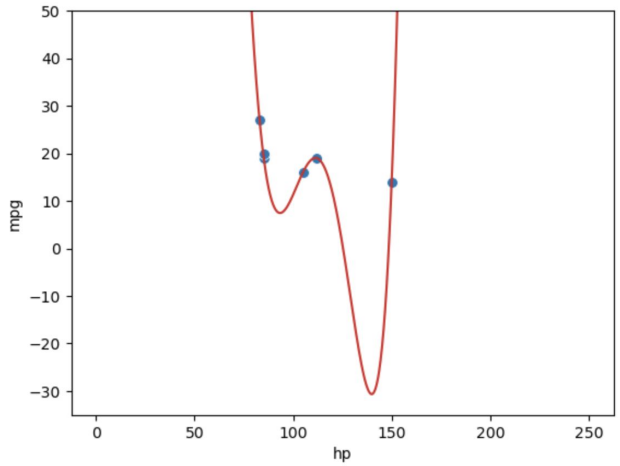
More realistic situation: we train the model on a *sample* from the population, then use it to make predictions on data it didn't encounter during training

Model Performance on Unseen Data

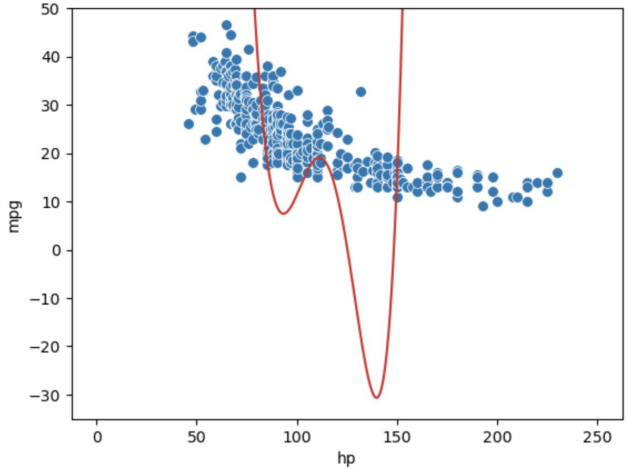
New (more realistic) example:

- We are given a training dataset of just 6 datapoints
- We want to train a model to then make predictions on a *different* set of points

We may be tempted to make a highly complex model (eg degree 5)



Complex model makes perfect predictions on the training data…



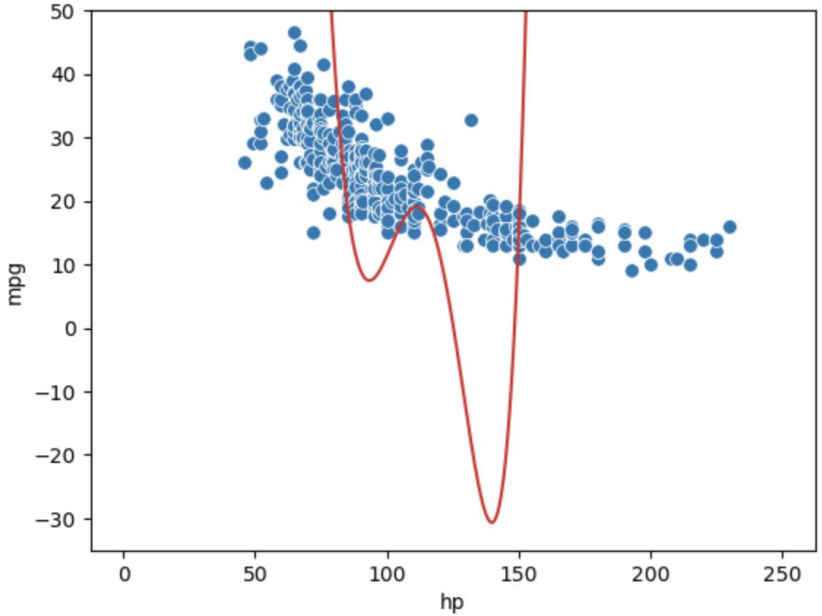
…but performs *horribly* on the rest of the population!

Model Performance on Unseen Data

What went wrong?

- The complex model **overfit** to the training data – it essentially "memorized" these 6 training points
- The overfitted model does not **generalize** well to data it did not encounter during training

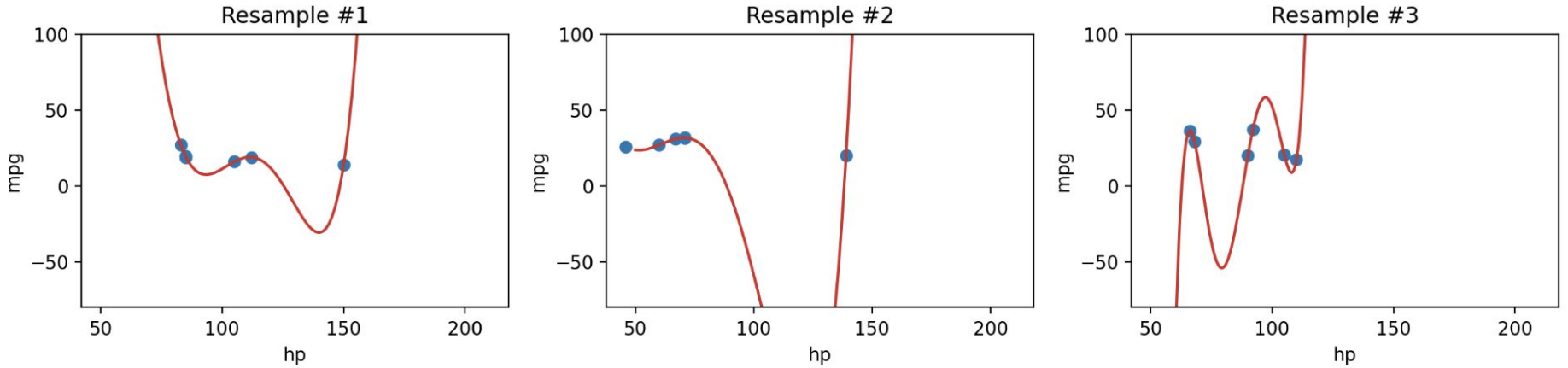
This is a problem: we want models that are generalizable to "unseen" data



Model Variance

Complex models are sensitive to the specific dataset used to train them – they have high **variance**, because they will *vary* depending on what datapoints are used for training them

Our degree-5 model varies erratically when we fit it to different samples of 6 points from `vehicles`

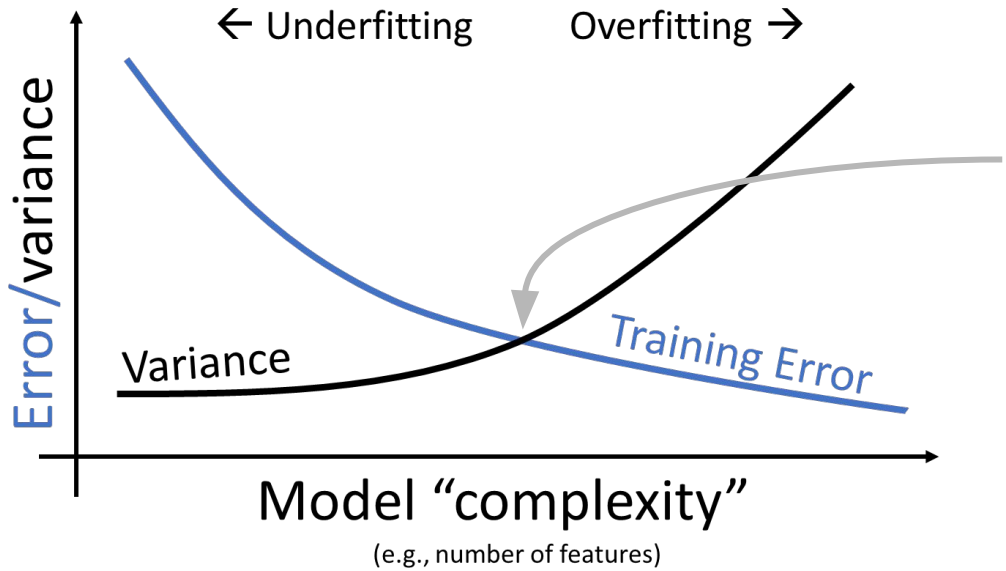


In Machine Learning, this **sensitivity** to data is known as **model variance**.

Error, Variance, and Complexity

We face a dilemma:

- We know that we can **decrease training error** by increasing model complexity
- However, models that are *too* complex start to overfit and do not generalize well – their **high variance** means they can't be reapplied to new datasets



Our goal: find this "sweet spot"

Training, Test, and Validation Sets

Cross-Validation

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

Review: Dataset

Today we will use the `mpg` dataset from the `seaborn` library.

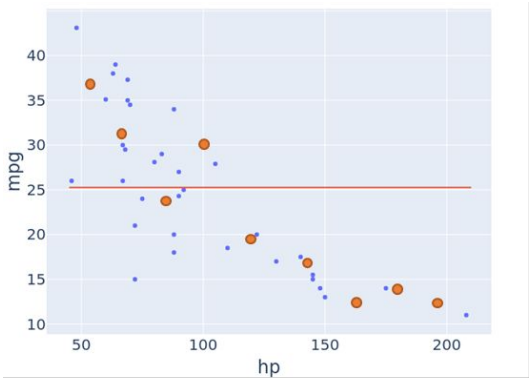
The dataset has 392 rows and 9 column. Our task is to use some of the columns and their transformations to predict the value of the `mpg` column.

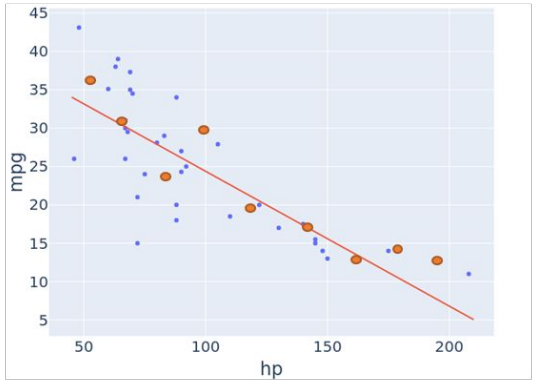
	mpg	cylinders	displacement	hp	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford ranger
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s-10

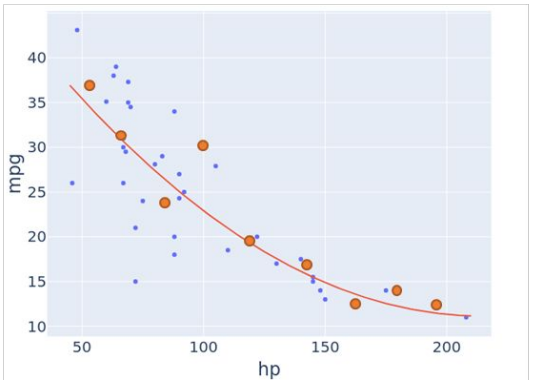
392 rows × 9 columns

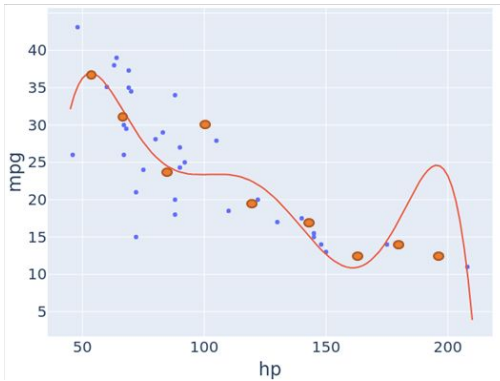
Review: Collecting More Data to Detect Overfitting

Suppose we use 35 sample to fit the regression model and collect the 9 new orange data points. We can compute MSE for our original models **without refitting using the new orange data points**.









Best?

k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001

Original 35 data points

Best?

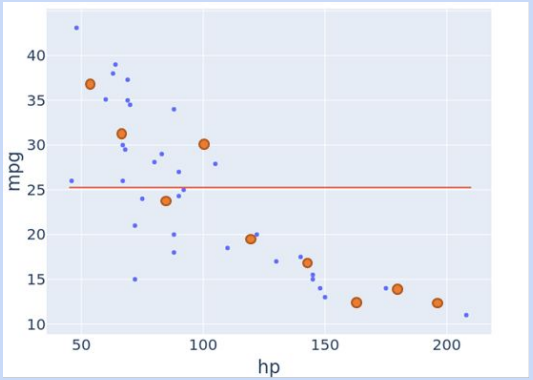
k	MSE
0	69.198210
1	31.189267
2	27.387612
3	29.127612
4	34.198272
5	37.182632
6	53.128712

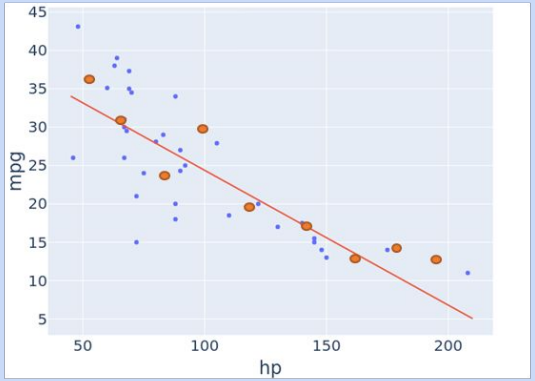
New 9 data points

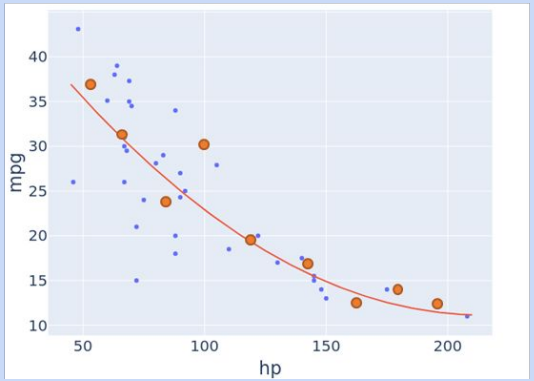
Review: Collecting More Data to Detect Overfitting

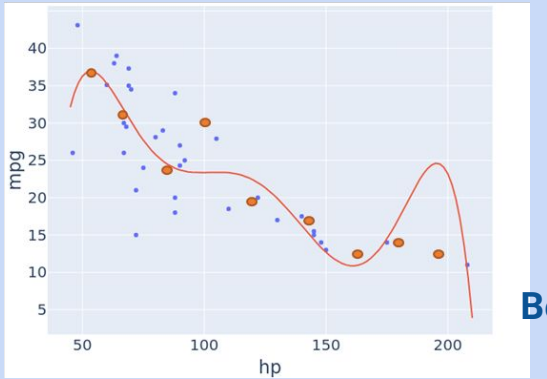
Which model do you like best? And why?

Poll: A) Degree 2 B) Degree 3 C) Degree 4 D) Degree 5 E)Degree 6









Best?

k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001

Original 35 data points

Best?

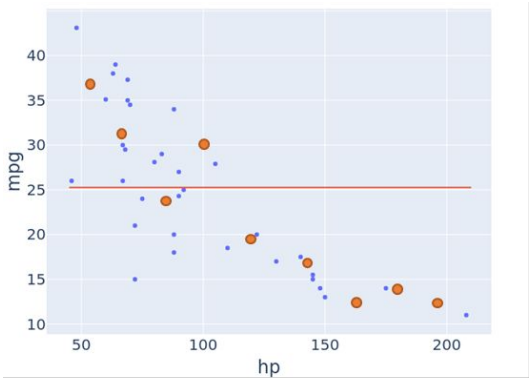
k	MSE
0	69.198210
1	31.189267
2	27.387612
3	29.127612
4	34.198272
5	37.182632
6	53.128712

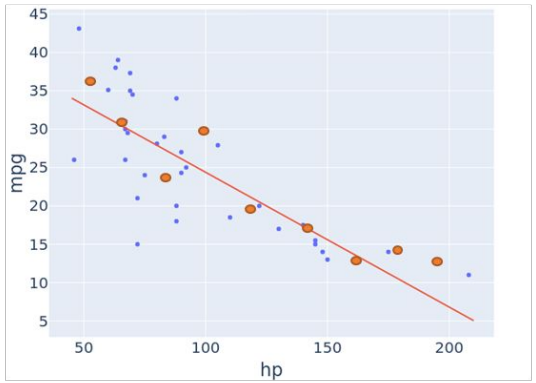
New 9 data points

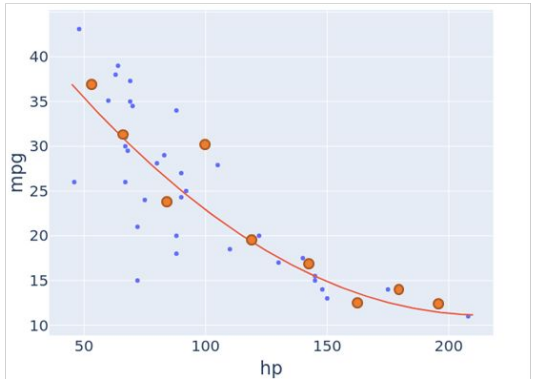
Review: Collecting More Data to Detect Overfitting

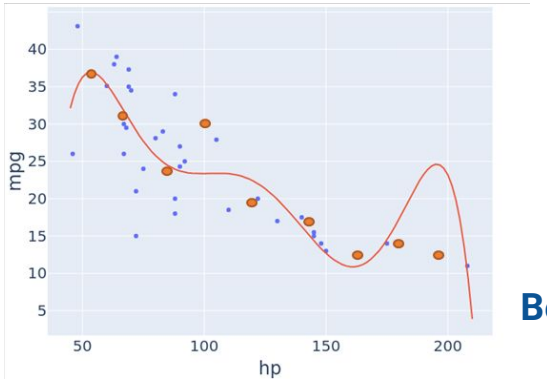
The order 2 model seems best to me.

- Performs best on data that has not yet been seen.









Best?

k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001

Original 35 data points

Best?

k	MSE
0	69.198210
1	31.189267
2	27.387612
3	29.127612
4	34.198272
5	37.182632
6	53.128712

New 9 data points

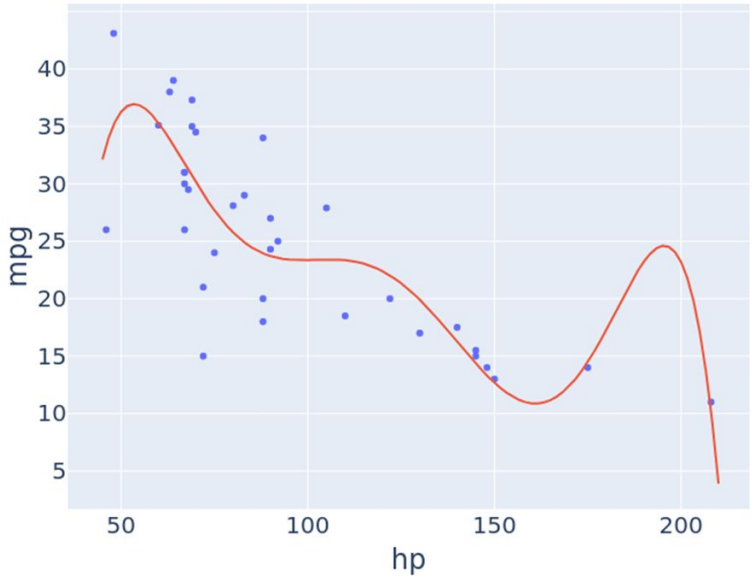
Review: Collecting More Data to Detect Overfitting

Suppose we have 7 models and don't know which is best.

- Can't necessarily trust the training error. We may have overfit!

We could wait for more data and see which of our 7 models does best on the new points.

- Unfortunately, that means we need to wait for more data. May be very expensive or time-consuming.
- "Will see an alternate approach next week."
  - As promised! Let's do it.

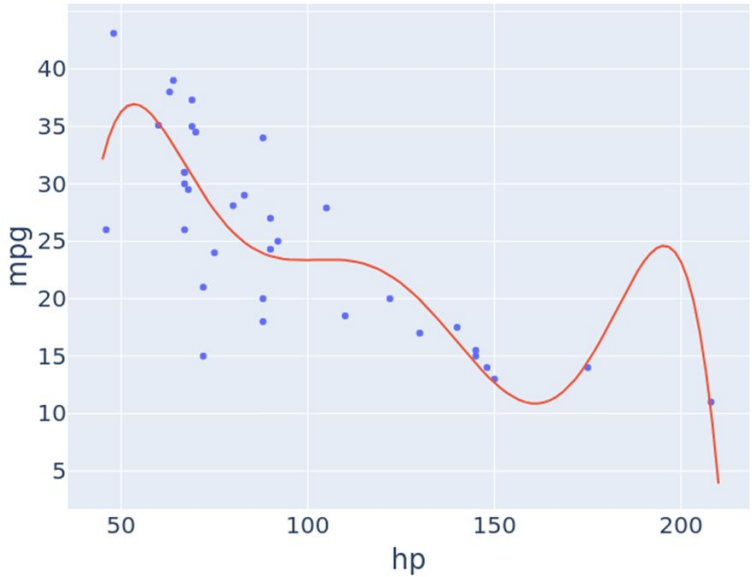


Idea 1: The Holdout Method

The simplest approach for avoiding overfitting is to keep some of our data secret from ourselves.

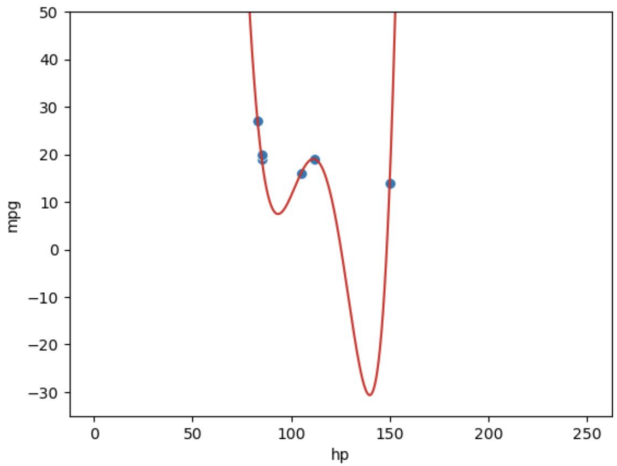
Example:

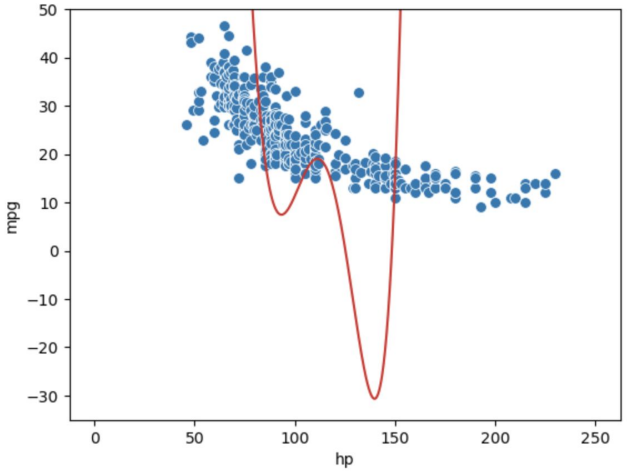
- Previous approach: We fit 7 models on all 35 of the available data points. Then waited for 9 new data points to decide which is best.
- Holdout Method: We **train** our models on all **25**/35 of the available data points. Then we **evaluate** the models' performance on the **remaining 10 data points**.
  - Data used to train is called the "**training set**".
  - **Held out data** is often called the "**validation set**" or "**development set**" or "**dev set**". These terms are all synonymous and used by different authors.



Where We Left Things

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





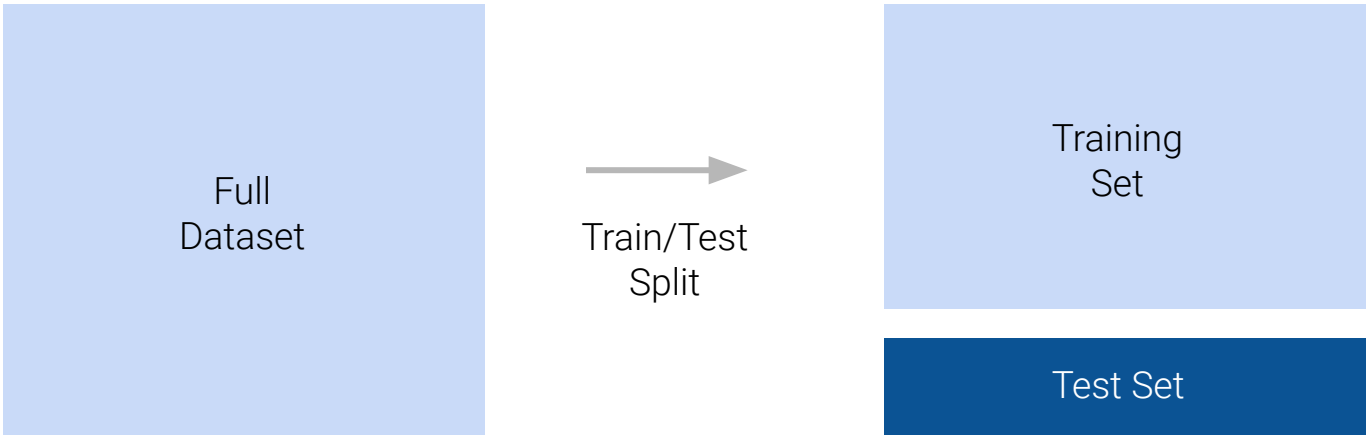
How to quantify performance on this "unseen" data? Introduce a **test set**.

Test Sets

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

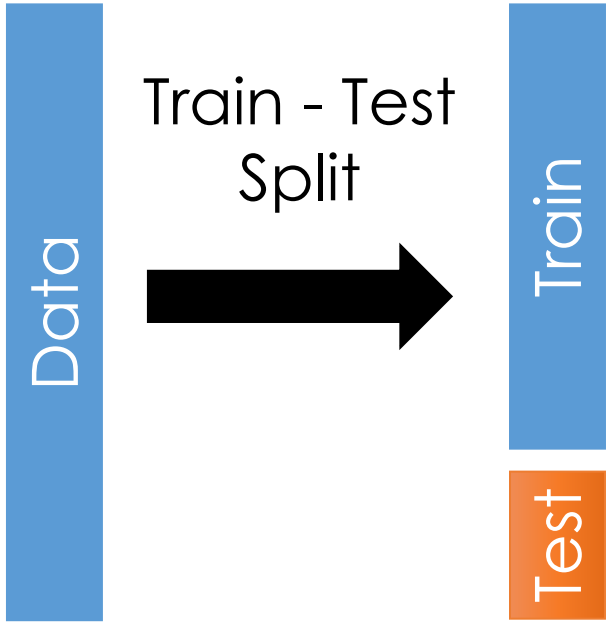
- We do *not* consider the test set when fitting/training the model.
- The test set is only ever touched once: to compute the performance (MSE, RMSE, etc) of the model *after* all fine-tuning has been completed.

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.



Generalization: *The Train-Test Split*

- **Training Data:** used to fit model
- **Test Data:** check generalization error
- How to split?
  - Randomly, Temporally, Geo…
  - Depends on application (usually randomly)
- What size? (90%-10%)
  - Larger training set – more complex models
  - Larger test set – better estimate of generalization error
  - Typically between 75%-25% and 90%-10%



You can only use the test dataset once after deciding on the model.

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

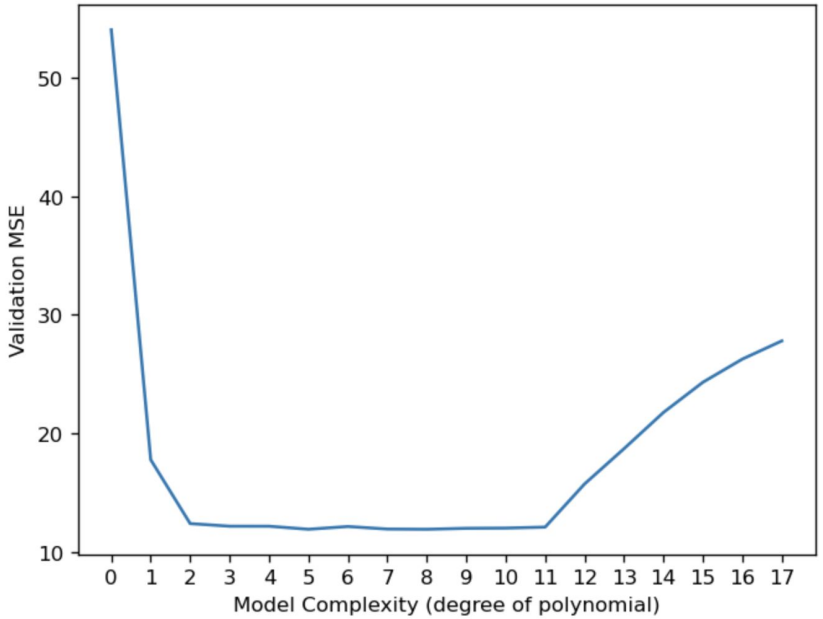
A **validation set** is a portion of our *training set* that we set aside for assessing model performance while it is *still being developed*.

- 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.

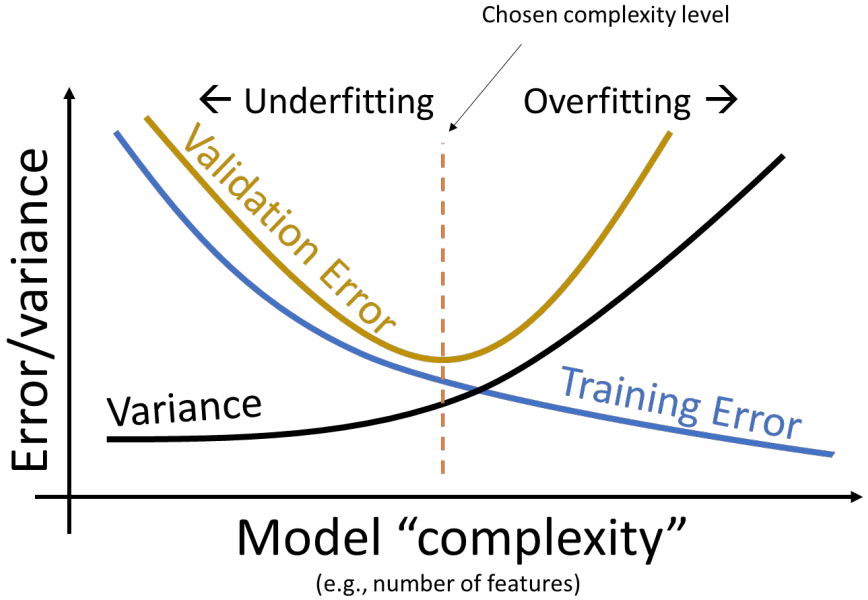
Updating Our Understanding of Model Complexity

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

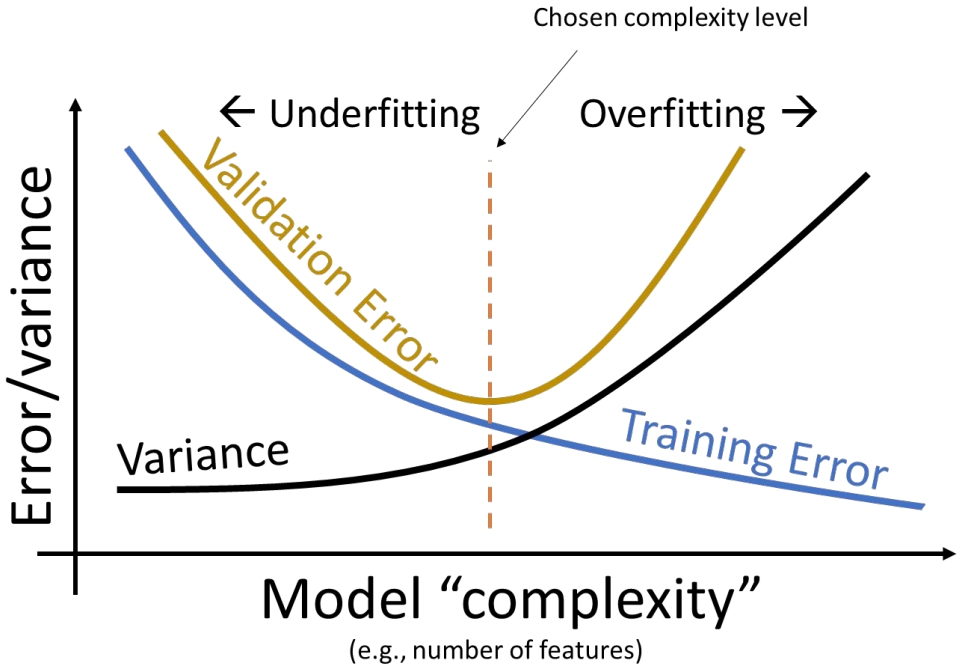
Our experiment using `vehicles`



General trend for an arbitrary dataset



Updating Our Understanding of Model Complexity



In the next lecture, 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.

Hyperparameter: Terminology

In machine learning, a **hyperparameter** is a value that controls the learning process itself.

- For our example today, we built a series of models each with increasing orders of horsepower. In this case, the hyperparameter is the degree or k that controlled the order of our polynomial.

We use:

- The **training set** to **select parameters**.
- The **validation set** (a.k.a. development set) (a.k.a. cross validation set) to **select hyperparameters**, or more generally, between different competing models.

K-Fold Cross Validation

Cross Validation

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

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.

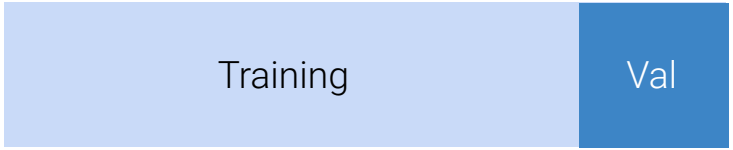
```
Val error from train/validation split #1: 14.6104005581132
Val error from train/validation split #2: 24.755706579814404
Val error from train/validation split #3: 22.23208329959848
```

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.

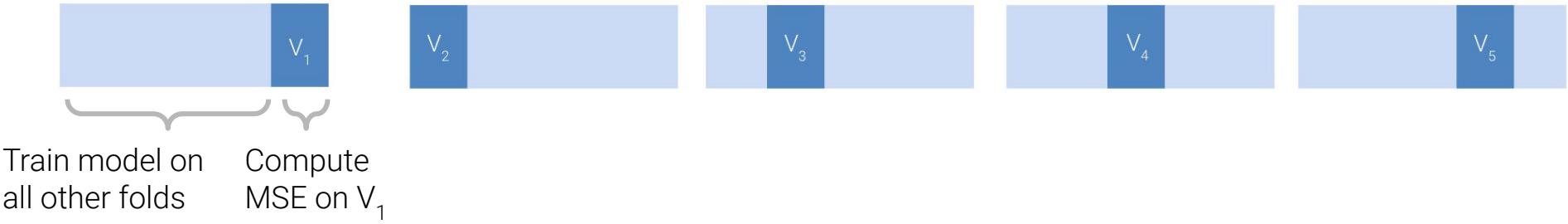
In **cross-validation**, we perform validation splits for *each* of these folds.

K-Fold Cross-Validation

For a dataset with K folds:

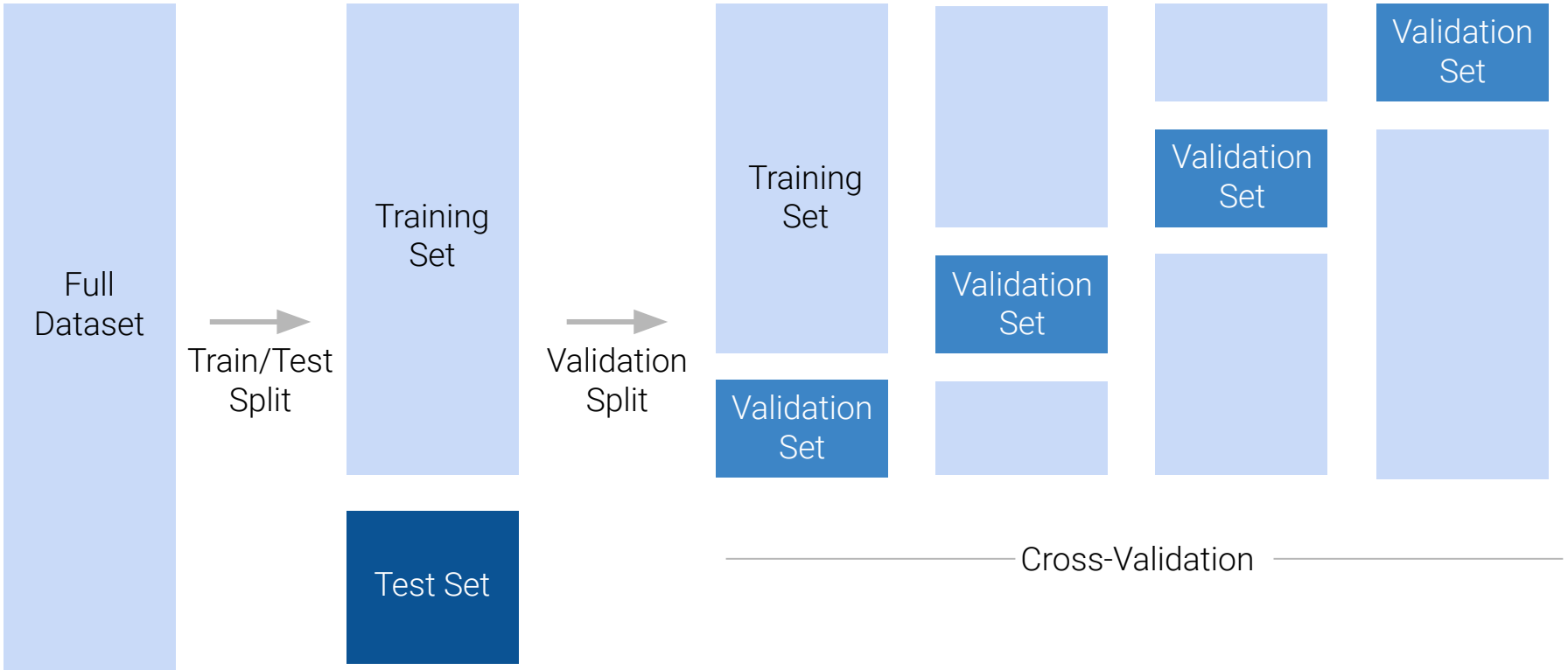
- Pick one fold to be the validation fold.
- Train model on data from every fold *other* than the validation fold.
- Compute the model's error on the validation fold and record it.
- Repeat for all K folds.

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



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

Model Selection Workflow



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.

*Equivalently, I could have said "**cross validation loss**" instead of "**cross validation score**".

Test Sets

Cross Validation

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

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 dataset 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**.

Test Set Terminology in Real World Practice

Warning: The terms “**test set**” and “**validation set**” are sometimes used interchangeably.

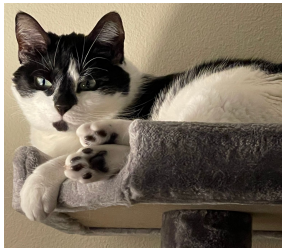
- You’ll see authors saying things like “then we used a **test set** to select hyperparameters”.
  - While this violates my personal definition of **test set**, it’s clear to me what they meant, namely: “we used a **holdout set** to select hyperparameters”.
  - This is a terminological confusion, not a procedural error! They didn’t do anything wrong.
    - Imagine they said “We used a **bloop-blop-set**”. Same thing , just weird name.
  - The error would be if they claimed later that the loss on their **test set** was unbiased. Since “**validation set**” error and actually completely unseen “**test set**” errors are typically very close, this terminology error is very minor.

In practice, you may not need a **test set** at all!

- If all you need to do is pick the best model, and you don’t care about providing a numerical measure of model quality, you don’t need a **test set**.

Validation Sets and Test Sets in Real World Practice

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



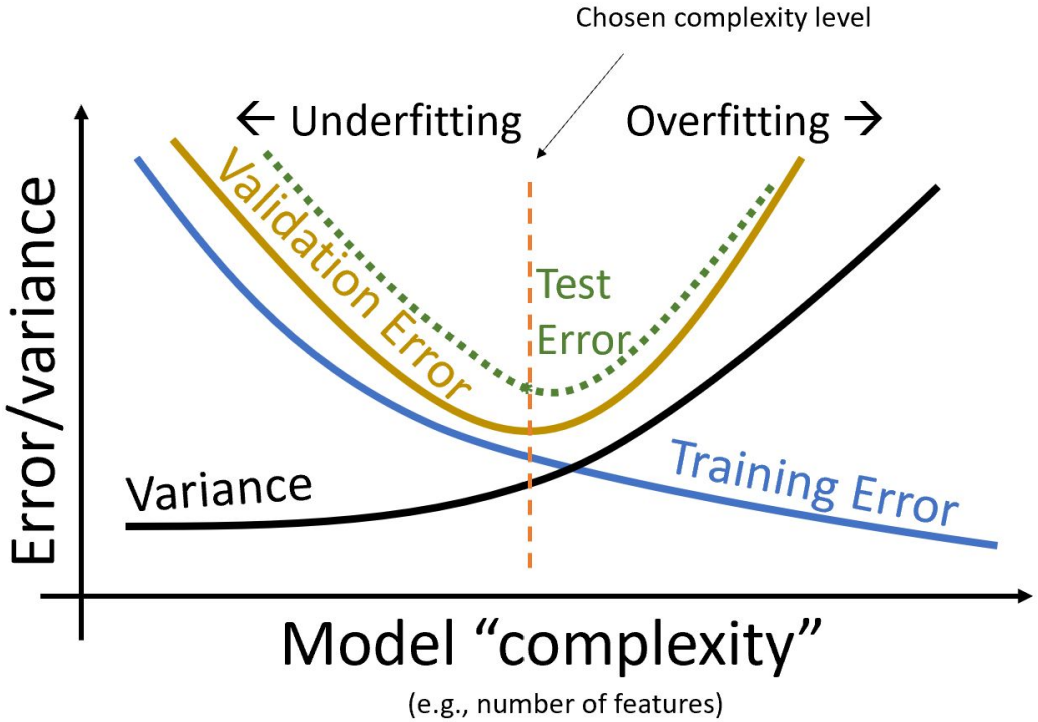
- Example: ImageNet is a dataset / competition used to compare to different image classification and localization algorithms on 1000 object classes (a "cat").
  - 1,281,167 **training images**. Images and correct label provided.
  - 50,000 **validation images**. Images and correct label provided.
  - 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.

Note: Since the competition uses the **test set scores** compare image classification algorithms, the best **test set score** is no longer an unbiased estimate of the best algorithm's performance.

Idealized Training, Validation, and Test Error

As we increase the complexity of our model:

- **Training error** decreases.
- Variance increases.
- Typically, **validation error** decreases, then increases.
- The **test error** is the essentially the same thing as the **validation error**! Only difference is that we are much restrictive about computing the **test error**
  - Don't get to see the whole curve!



Recipe for Successful Generalization

1. Split your data into **training** and **test** sets (80%, 20%)
2. Use **only the training data** when designing, training, and tuning the model
   - Use **cross validation** to test *generalization* during this phase
   - **Do not look at the test data**
3. Commit to your final model and train once more using **only the training data.**
4. Test the final model using the **test data.** If accuracy is not acceptable return to (2). (*Get more test data if possible.*)
5. Train **on all available data** and ship it!

Appendix: Using Regularization To Constrain Model Parameters

Cross-Validation

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

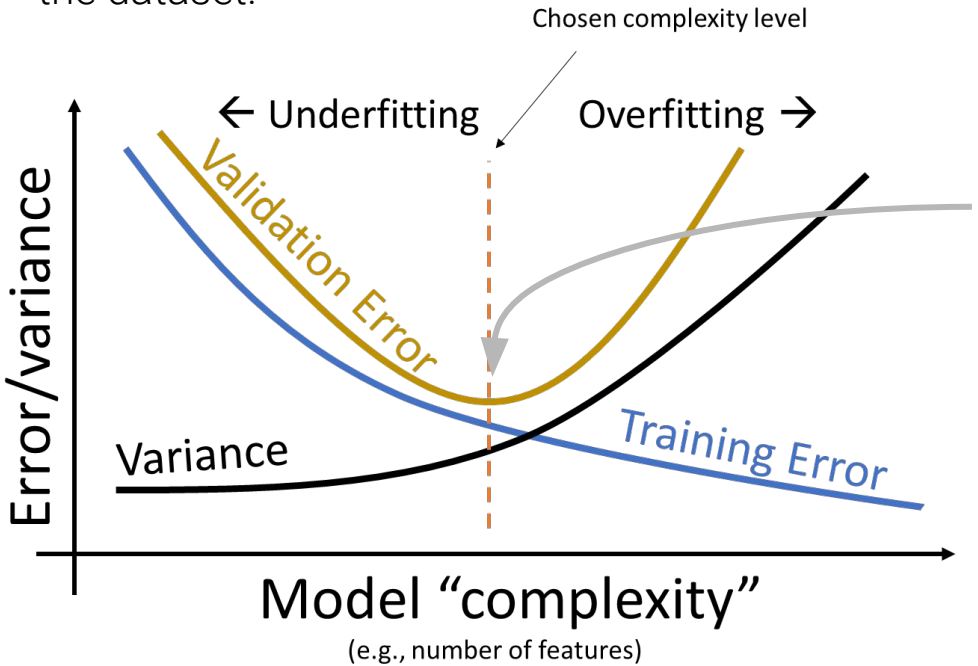
Regularization

- **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.

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

- 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.

Restricting Model Complexity

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

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

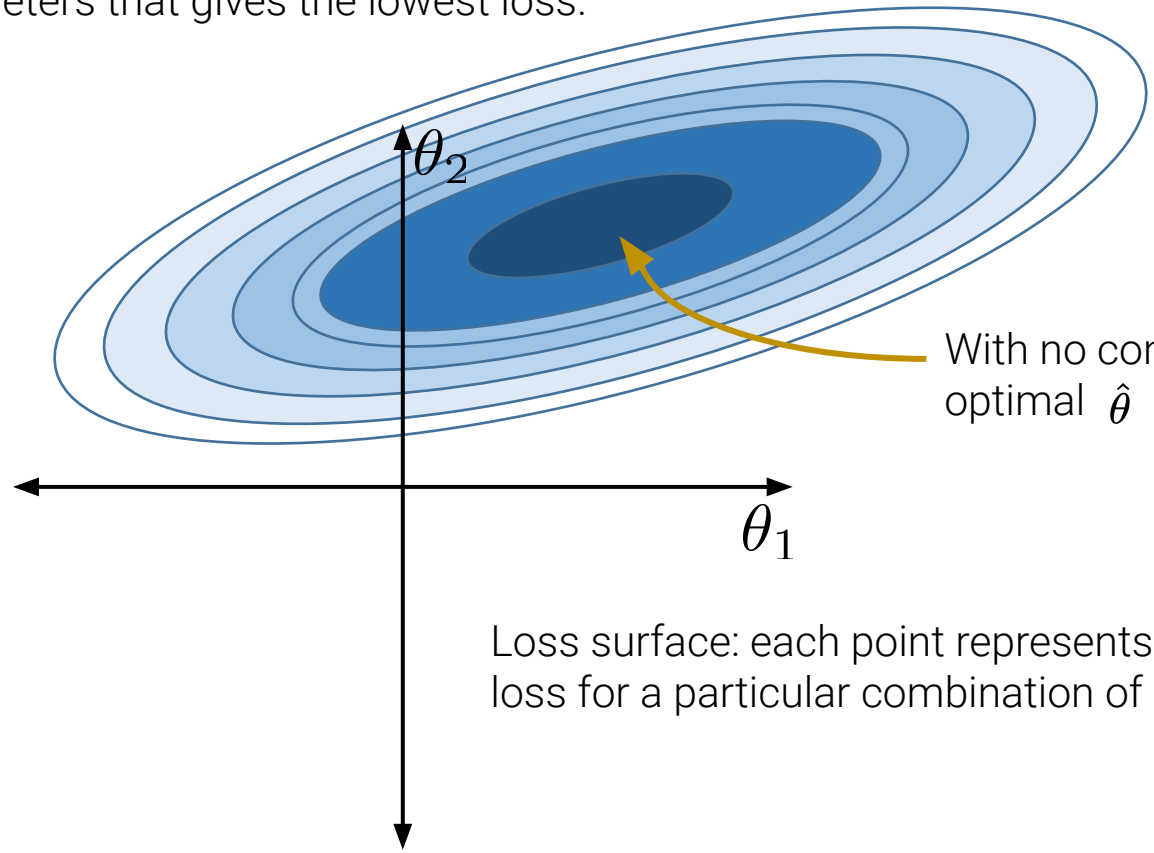
$$\sum_{i=1}^{p} |\theta_i| \leq Q$$

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

Note that the intercept term, θ_0, is typically excluded from this constraint.

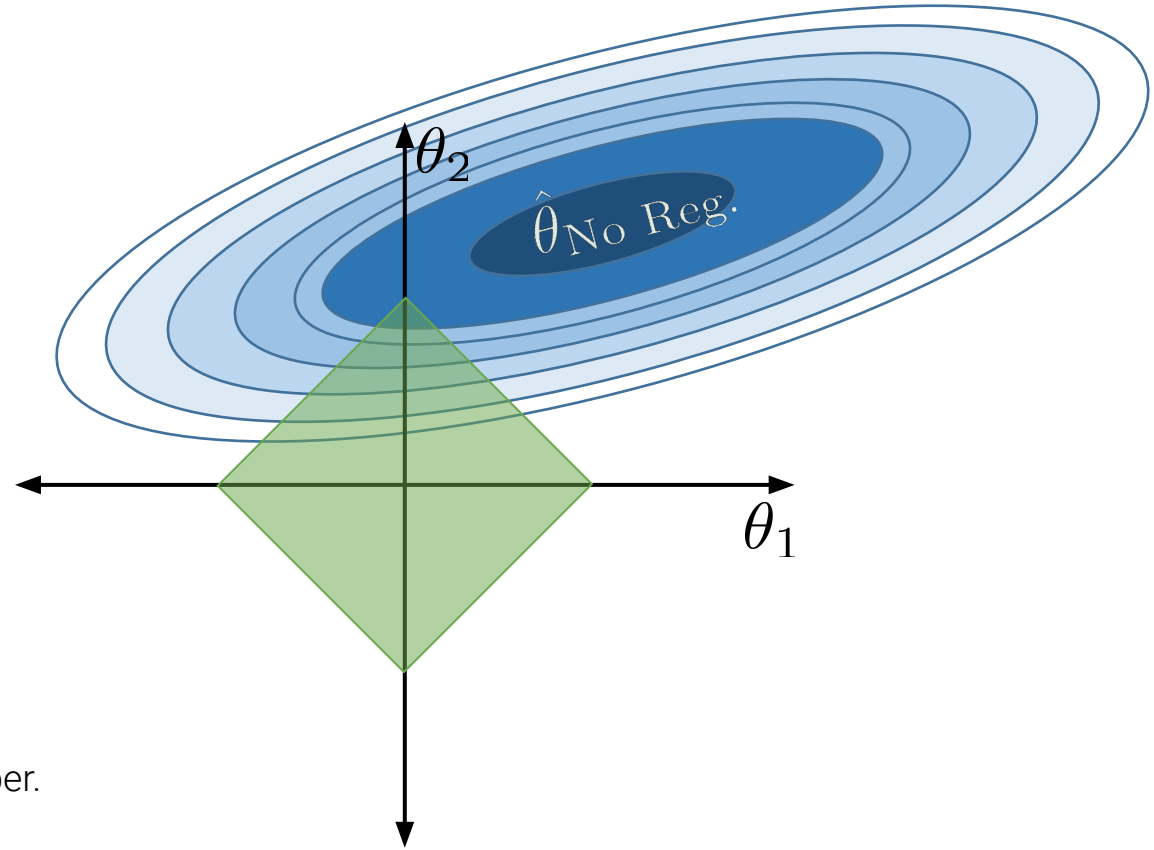
Think Back to Gradient Descent

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

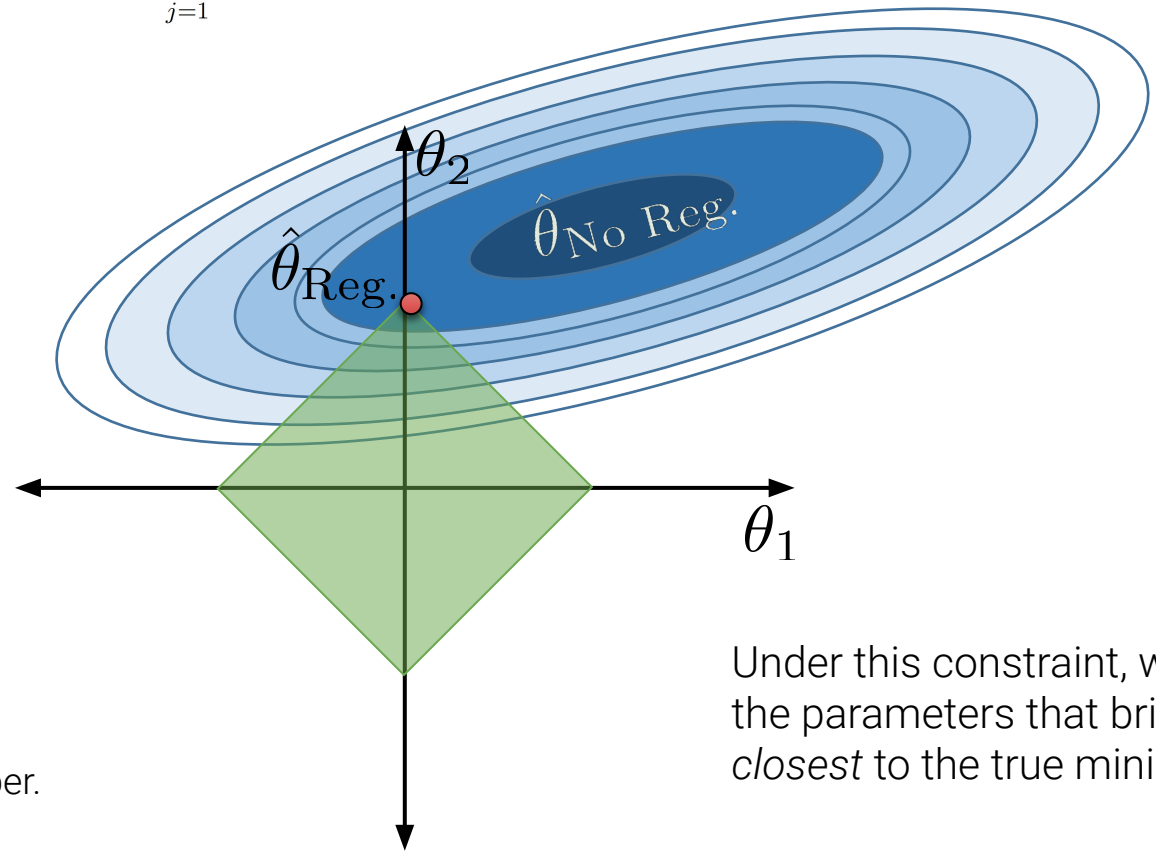
When we apply the constraint $\sum_{i=1}^{p} |\theta_i| \leq Q$, only parameter combinations inside the diamond are valid.



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.

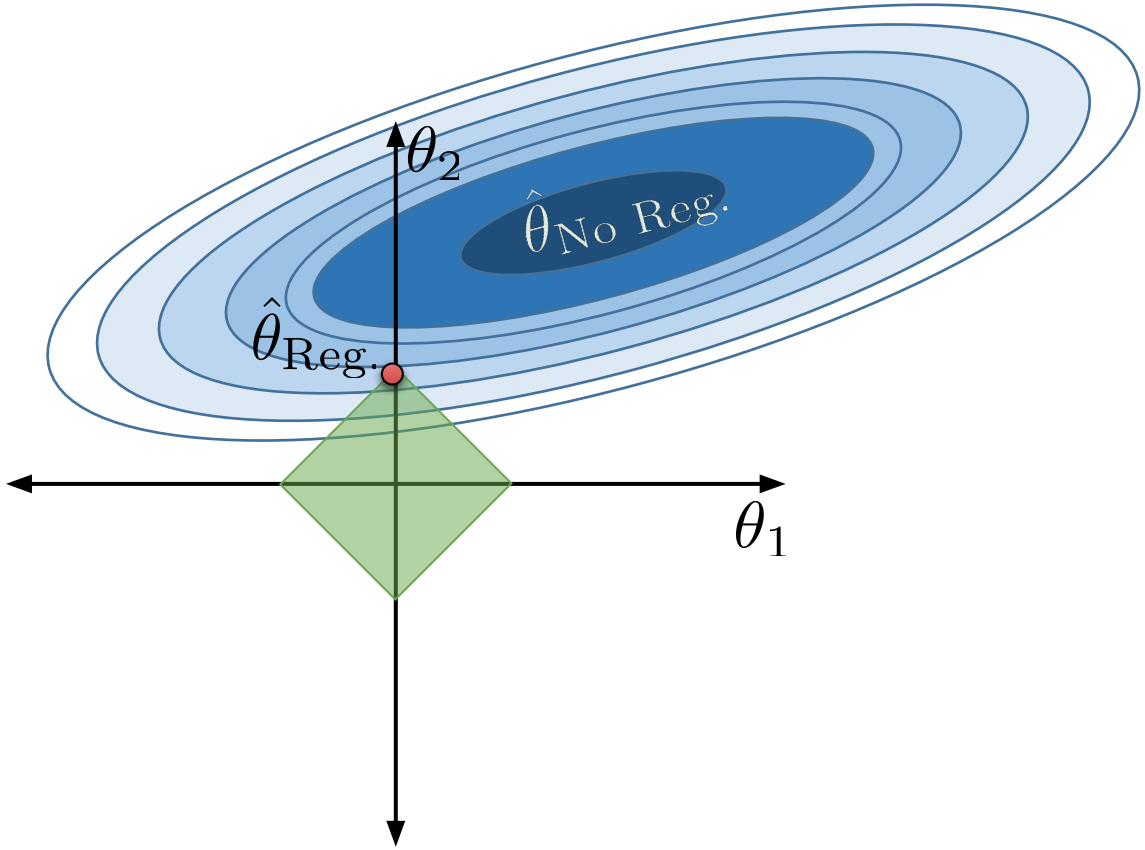


Under this constraint, we choose the parameters that bring us *closest* to the true minimum loss.

Q is some arbitrary number.

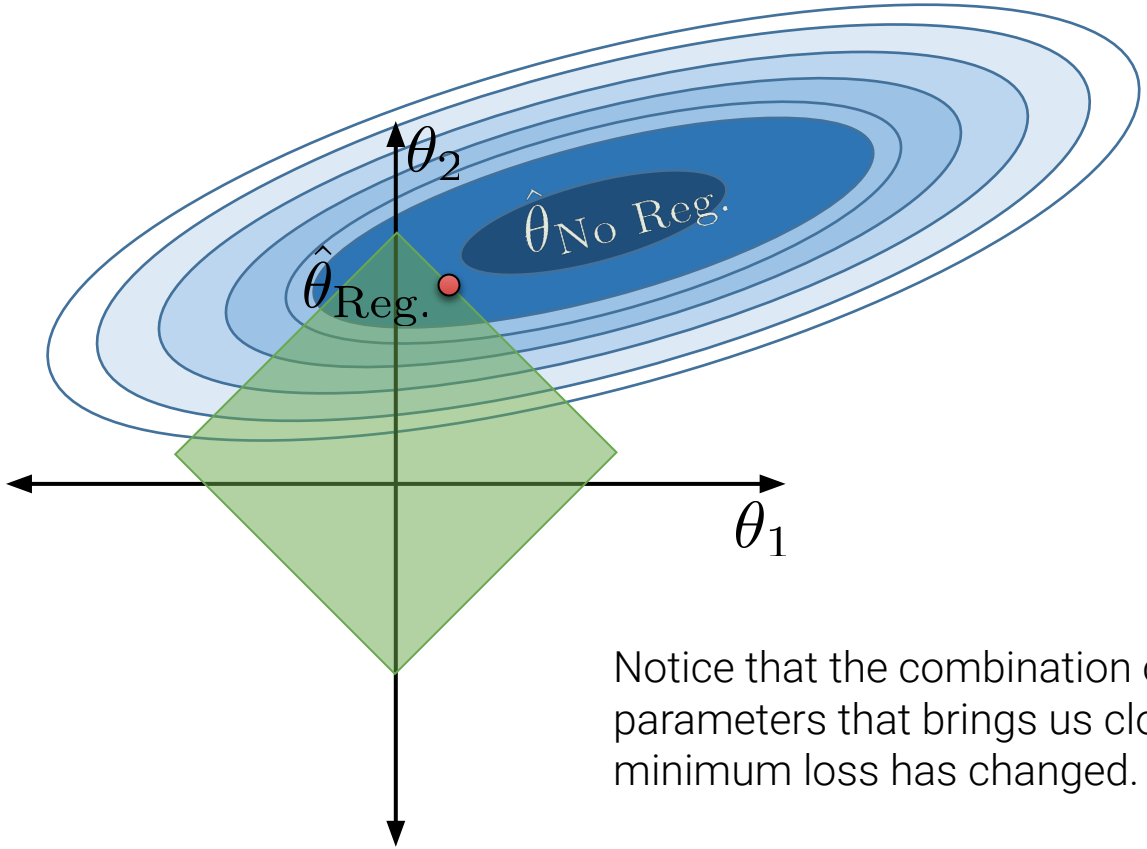
Smaller Q

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.



Notice that the combination of model parameters that brings us closest to minimum loss has changed.

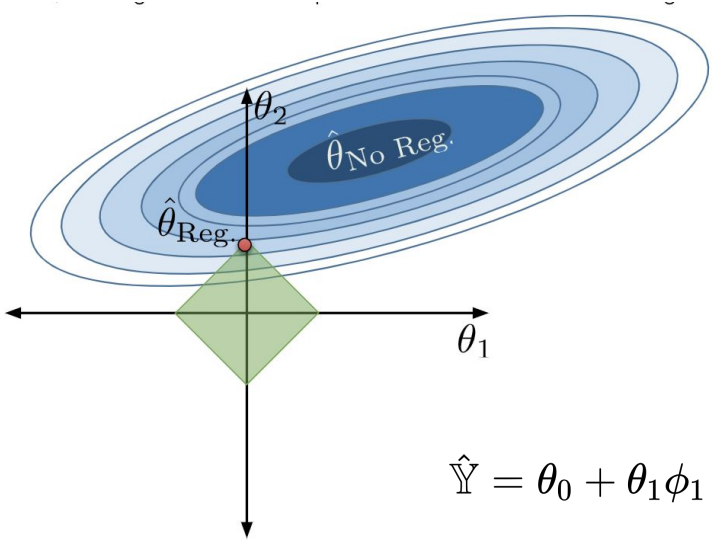
slido

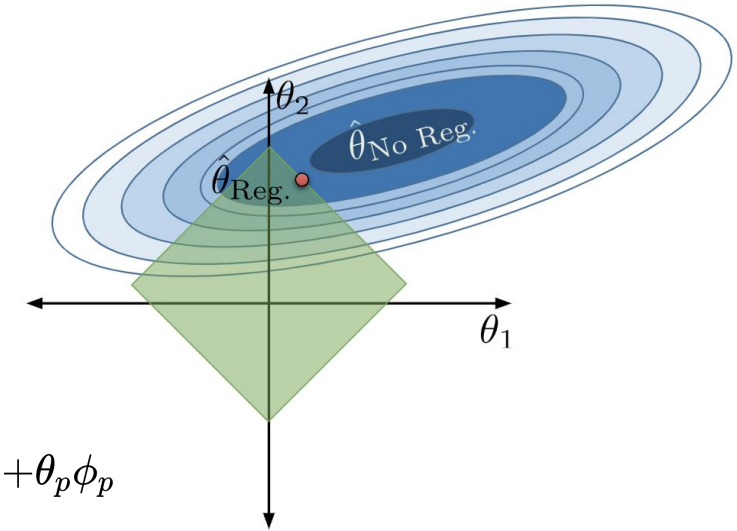
How does the size of Q relate to model complexity?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Size of Q

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





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

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.

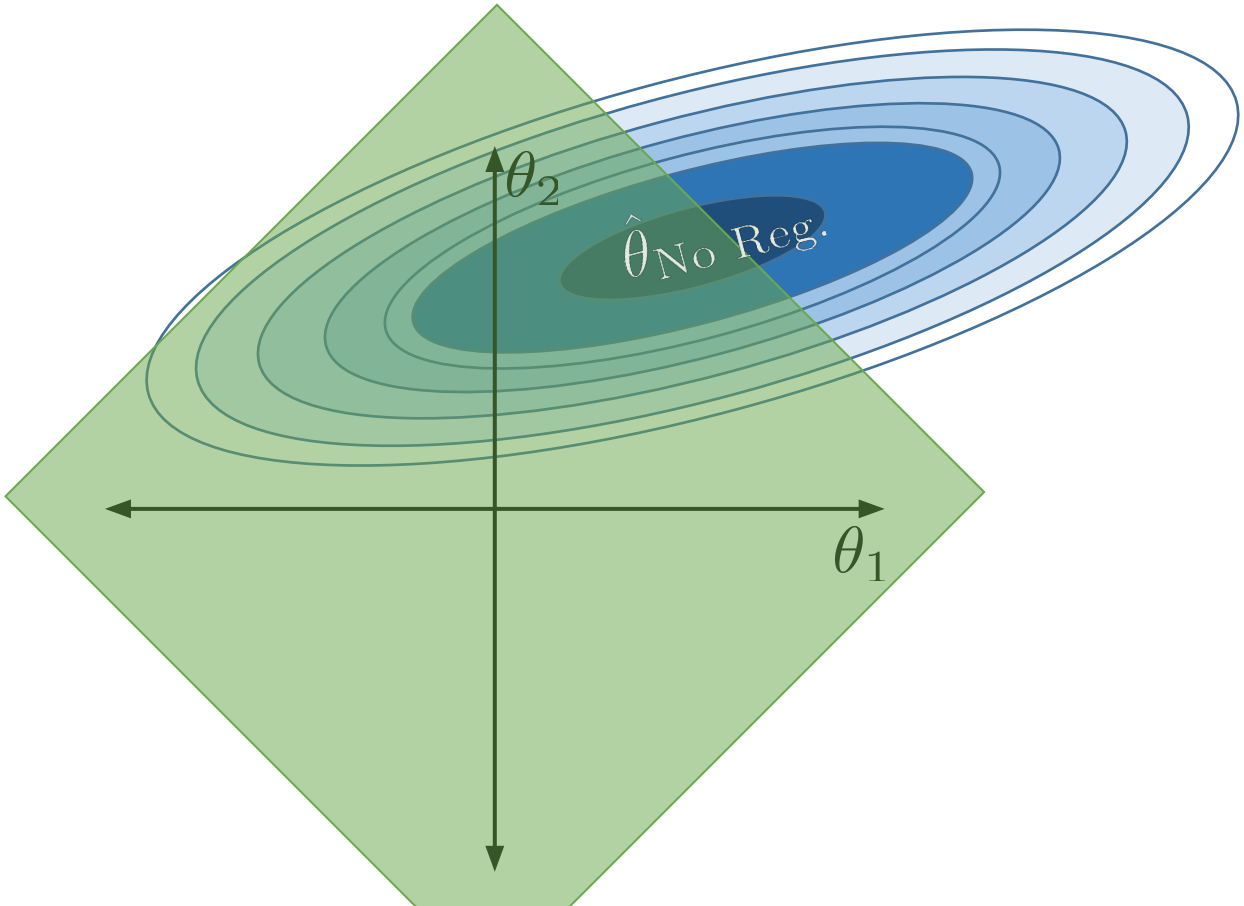
slido

If we pick a very large value of Q, what kind of model will we have?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Size of Q

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



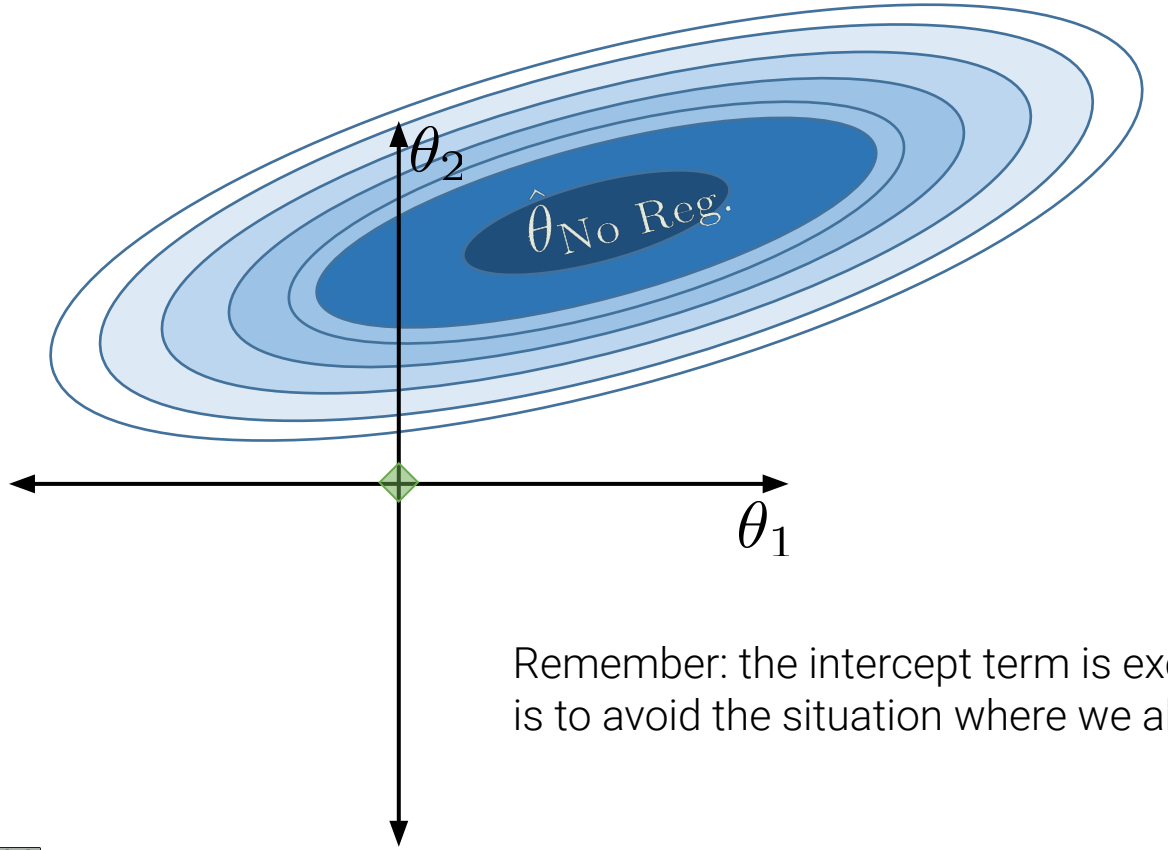
slido

If we pick a very small value of Q, what kind of model will we have?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Size of Q

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



If the model has no intercept term:

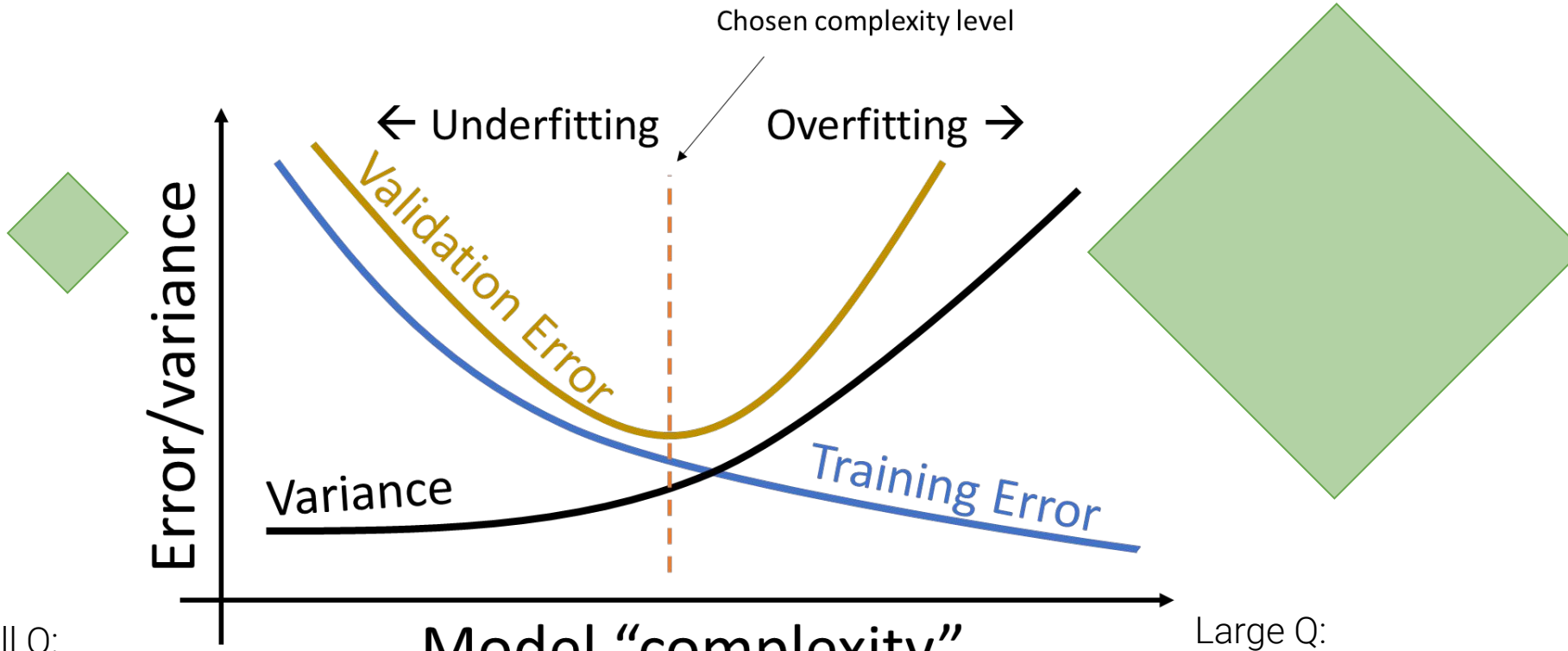
$$\hat{\mathbb{Y}} = (0)\phi_1 + (0)\phi_2 + \ldots = 0$$

If the model has an intercept term:

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

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

Complexity and Q



Small Q:

- Simple model
- Underfitting

Large Q:

- Complex model
- Overfitting

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:

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

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

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

L1 Regularization

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

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

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

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

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

*Details out of scope for Data 100; take EECS 127 to learn more

L1 Regularization

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

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

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

- In our earlier approach: $\lambda \approx \frac{1}{Q}$
- How to choose the value for λ? Cross-validation!

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

L1 Regularized OLS in `sklearn`

There is no closed form solution for the optimal LASSO theta. Instead, use **`sklearn`**.

In **`sklearn`**, we use the **`Lasso`** model class.

- Runs gradient descent to **minimize the L1 objective function.**

```
import sklearn.linear_model as lm
lasso_model = lm.Lasso(alpha = 1) # alpha represents the hyperparameter lambda
lasso_model.fit(X_train, Y_train)
lasso_model.coef_
```

```
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])
```

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_model_large_lambda.coef_
```

```
array([-0.00000000e+00, -3.37446532e-03,  1.31817186e-05,  1.71062658e-08,
       -2.44893438e-11, -2.11314339e-13, -5.38994214e-16,  7.05457777e-19])
```

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_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: $\hat{y}_i = \theta_0 + \theta_1(150) + \ldots + \theta_8\left(2.56 \times 10^{17}\right)$

The large values of hp^8 dominate the prediction.

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

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.

```
lasso_model_scaled = lm.Lasso(alpha=1)
lasso_model_scaled.fit(X_train_standardized, Y_train)
lasso_model_scaled.coef_
```

```
array([-9.31789105,  0.        ,  0.        ,  2.89288682,  0.65909948,
        0.        ,  0.        ,  0.        ])
```

L2 Regularization (Ridge)

Cross-Validation

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

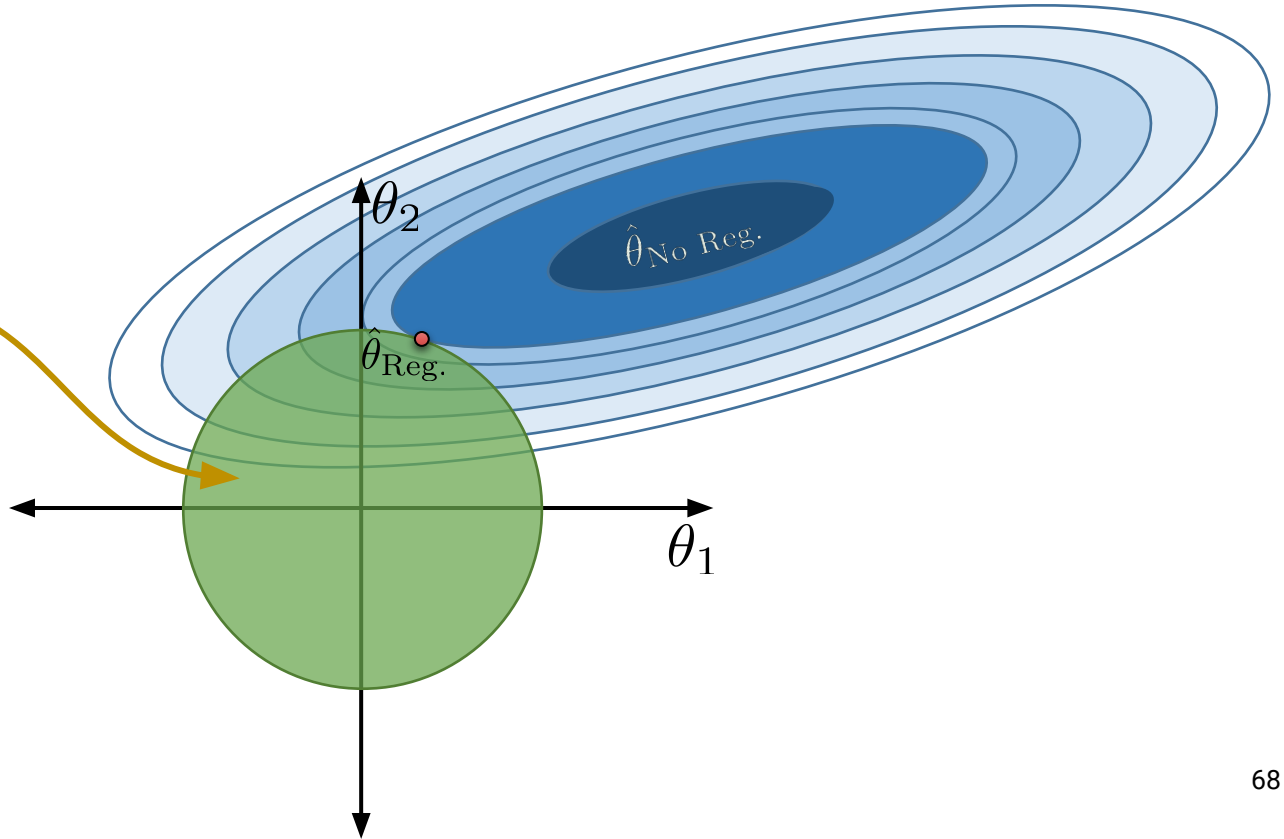
Regularization

- 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:

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

L2 objective function:

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

L2 Regularization

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

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

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

L2 Regularized OLS in `sklearn`

In **`sklearn`**, we use the **`Ridge`** model class.

- Runs gradient descent to **minimize the L2 objective function**

```
import sklearn.linear_model as lm
ridge_model = lm.Ridge(alpha = 1) # alpha represents the hyperparameter lambda
ridge_model.fit(X_train, Y_train)
ridge_model.coef_
```

```
ridge_model.coef_
```

```
array([-16.85961652,   3.26398097,   9.1167183 ,   4.53790201,
        -2.32110639,  -5.6066523 ,  -3.15831859,   4.75104822])
```

Closed Form Solution for L2 Regularization

Applying vector calculus (out of scope) allows us to find a closed-form solution for L2 regularization!

- Recall that L1 regularization has no closed-form solution:

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

This solution exists **even if** $\mathbb{X}$ is not full rank – an important reason why we often prefer L2 regularization. This will be important once we discuss multicolinearity next week.

Summary of Regression Methods

Our regression models are summarized below.

- The objective function is what the gradient descent optimizer minimizes.

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

2327602

Appendix: Geometric Derivation

OLS Problem Formulation

- Multiple Linear Regression Model
- Mean Squared Error

Geometric Derivation

- Lin Alg Review: Orthogonality, Span
- Least Squares Estimate Proof

Vector Notation

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p$$

$$= \theta_0 + \sum_{j=1}^{p} \theta_j x_j$$

To combine the two terms into one matrix operation, we can assume that there is an additional term $x_0 = 1$ in $\vec{x}$ and hence:

$$= x^T \theta \qquad \vec{x} = \begin{bmatrix} 1 \\ 0.4 \\ 0.8 \\ 1.5 \end{bmatrix} \quad \vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \qquad \vec{x}, \vec{\theta} \in \mathbb{R}^{(p+1)}$$

$$= \begin{bmatrix} 1 & 0.4 & 0.8 & 1.5 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \hat{y} \in \mathbb{R}$$

NBA Data

	FG	AST	3PA	PTS
1	1.8	0.6	4.1	5.3
2	0.4	0.8	1.5	1.7
3	1.1	1.9	2.2	3.2
4	6.0	1.6	0.0	13.9
5	3.4	2.2	0.2	8.9
6	0.6	0.3	1.2	1.7

Rows correspond to individual players.

Note that:

$$\vec{x}^T \times \vec{\theta} = \vec{x}.\vec{\theta}$$

Matrix Notation

To make predictions on all n datapoints in our sample:

Data

	FG	AST	3PA	PTS
1	1.8	0.6	4.1	5.3
2	0.4	0.8	1.5	1.7
3	1.1	1.9	2.2	3.2

$$\hat{y}_1 = x_1^T\theta \quad \text{where } x_1^T = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \end{bmatrix} \quad \text{Datapoint 1}$$

$$\hat{y}_2 = x_2^T\theta \quad \text{where } x_2^T = \begin{bmatrix} 1 & x_{21} & x_{22} & \dots & x_{2p} \end{bmatrix} \quad \text{Datapoint 2}$$

$$\vdots \qquad \vdots \qquad\qquad\qquad \vdots$$

$$\hat{y}_n = x_n^T\theta \quad \text{where } x_n^T = \begin{bmatrix} 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \quad \text{Datapoint n}$$

same $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$ for all preds

Matrix Notation

To make predictions on all n datapoints in our sample:

$$\begin{aligned}
\hat{y}_1 &= \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \end{bmatrix} \theta &= x_1^T \theta \\
\hat{y}_2 &= \begin{bmatrix} 1 & x_{21} & x_{22} & \dots & x_{2p} \end{bmatrix} \theta &= x_2^T \theta \\
&\vdots & & \vdots & \\
\hat{y}_n &= \begin{bmatrix} 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \theta &= x_n^T \theta
\end{aligned}$$

n row vectors, each with dimension **(p+1)**

Data

	FG	AST	3PA	PTS
1	1.8	0.6	4.1	5.3
2	0.4	0.8	1.5	1.7
3	1.1	1.9	2.2	3.2

same θ for all preds $= \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

Expand out each data point's (transposed) input

Matrix Notation

To make predictions on all n datapoints in our sample:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ & & \vdots & & \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \theta$$

n row vectors, each with dimension **(p+1)**

Data

	FG	AST	3PA	PTS
1	1.8	0.6	4.1	5.3
2	0.4	0.8	1.5	1.7
3	1.1	1.9	2.2	3.2

same θ for all preds $= \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

Vectorize predictions and parameters to encapsulate all n equations into a single matrix equation.

Matrix Notation

To make predictions on all n datapoints in our sample:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \mathbb{X} \theta$$

Design matrix with dimensions n x (p + 1)

Data

	FG	AST	3PA	PTS
1	1.8	0.6	4.1	5.3
2	0.4	0.8	1.5	1.7
3	1.1	1.9	2.2	3.2

same $\theta =$ for all preds $\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

The Design Matrix $\mathbb{X}$

We can use linear algebra to represent our predictions of all n data points at once.

One step in this process is to stack all of our input features together into a **design matrix**:

$$\mathbb{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ 1 & x_{31} & x_{32} & \dots & x_{3p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

What do the **rows** and **columns** of the design matrix represent in terms of the observed data?

Field Goals, Assists, 3-Point Attempts

Bias	FG	AST	3PA	PTS
1	1.8	0.6	4.1	5.3
1	0.4	0.8	1.5	1.7
1	1.1	1.9	2.2	3.2
1	6.0	1.6	0.0	13.9
1	3.4	2.2	0.2	8.9
...	...	...	...	...
1	4.0	0.8	0.0	11.5
1	3.1	0.9	0.0	7.8
1	3.6	1.1	0.0	8.9
1	3.4	0.8	0.0	8.5
1	3.8	1.5	0.0	9.4

Example design matrix
708 rows x (3+1) cols

The Design Matrix $\mathbb{X}$

We can use linear algebra to represent our predictions of all n data points at once.

One step in this process is to stack all of our input features together into a **design matrix**:

$$\mathbb{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ 1 & x_{31} & x_{32} & \dots & x_{3p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

A **row** corresponds to one **observation**, e.g., all (p+1) features for datapoint 3

A **column** corresponds to a **feature**, e.g. feature 1 for all n data points

Special all-ones feature often called the **bias/intercept**

Bias	FG	AST	3PA	PTS
1	1.8	0.6	4.1	5.3
1	0.4	0.8	1.5	1.7
1	1.1	1.9	2.2	3.2
1	6.0	1.6	0.0	13.9
1	3.4	2.2	0.2	8.9
...	...	...	...	...
1	4.0	0.8	0.0	11.5
1	3.1	0.9	0.0	7.8
1	3.6	1.1	0.0	8.9
1	3.4	0.8	0.0	8.5
1	3.8	1.5	0.0	9.4

Example design matrix
708 rows x (3+1) cols

The Multiple Linear Regression Model using Matrix Notation

We can express our linear model on our entire dataset as follows:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ 1 & x_{31} & x_{32} & \dots & x_{3p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}$$

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

Prediction vector $\mathbb{R}^n$

Design matrix $\mathbb{R}^{n \times (p+1)}$

Parameter vector $\mathbb{R}^{(p+1)}$

Note that our **true output** is also a vector:

$$\mathbb{Y} \in \mathbb{R}^n$$

[Linear Algebra] Vector Norms and the L2 Vector Norm

The **norm** of a vector is some measure of that vector's **size**.

- The two norms we need to know for Data 100 are the L_1 and L_2 norms (sound familiar?).
- Today, we focus on L_2 norm. We'll define the L_1 norm another day.

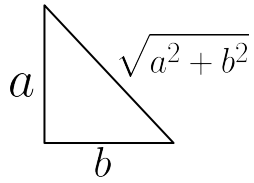
For the n-dimensional vector $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, the **L2 vector norm** is

$$||\vec{x}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

[Linear Algebra] The L2 Norm Is a Measure of Distance

$$||\vec{x}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

In $\mathbb{R}^2$

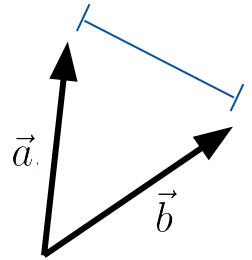


The L2 vector norm is a generalization of the Pythagorean theorem into ***n*** dimensions.

It can therefore be used as a measure of **distance** between two vectors.

- For n-dimensional vectors $\vec{a}, \vec{b}$, their distance is $||\vec{a} - \vec{b}||_2$.

In $\mathbb{R}^n$



Note: The square of the L2 norm of a vector is the sum of the squares of the vector's elements:

$$(||\vec{x}||_2)^2 = \sum_{i=1}^{n} x_i^2$$

Looks like Mean Squared Error!!

Mean Squared Error with L2 Norms

We can rewrite mean squared error as a squared L2 norm:

$$R(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
$$= \frac{1}{n}||\mathbb{Y} - \hat{\mathbb{Y}}||_2^2$$

With our linear model $\hat{\mathbb{Y}} = \mathbb{X}\theta$:

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

Ordinary Least Squares

The **least squares estimate** $\hat{\theta}$ is the parameter that **minimizes** the objective function $R(\theta)$:

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

How should we interpret the OLS problem?

A. Minimize the mean squared error for the linear model $\hat{\mathbb{Y}} = \mathbb{X}\theta$

B. Minimize the **distance** between true and predicted values $\mathbb{Y}$ and $\hat{\mathbb{Y}}$

C. Minimize the **length** of the residual vector, $e = \mathbb{Y} - \hat{\mathbb{Y}} = \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ \vdots \\ y_n - \hat{y}_n \end{bmatrix}$

D. All of the above

E. Something else

Ordinary Least Squares

The **least squares estimate** $\hat{\theta}$ is the parameter that **minimizes** the objective function $R(\theta)$:

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

How should we interpret the OLS problem?

A. Minimize the mean squared error for the linear model $\hat{\mathbb{Y}} = \mathbb{X}\theta$

B. Minimize the **distance** between true and predicted values $\mathbb{Y}$ and $\hat{\mathbb{Y}}$

C. Minimize the **length** of the residual vector, $e = \mathbb{Y} - \hat{\mathbb{Y}} = \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ \vdots \\ y_n - \hat{y}_n \end{bmatrix}$ Important for today

D. All of the above

E. Something else

Today's Goal: Ordinary Least Squares

1. Choose a model ✅

Multiple Linear Regression

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

2. Choose a loss function ✅

L2 Loss

Mean Squared Error (MSE)

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

3. Fit the model

Minimize average loss with ~~calculus~~ geometry

The calculus derivation requires matrix calculus (out of scope, but here's a link if you're interested).

Instead, we will derive $\hat{\theta}$ using a **geometric argument**.

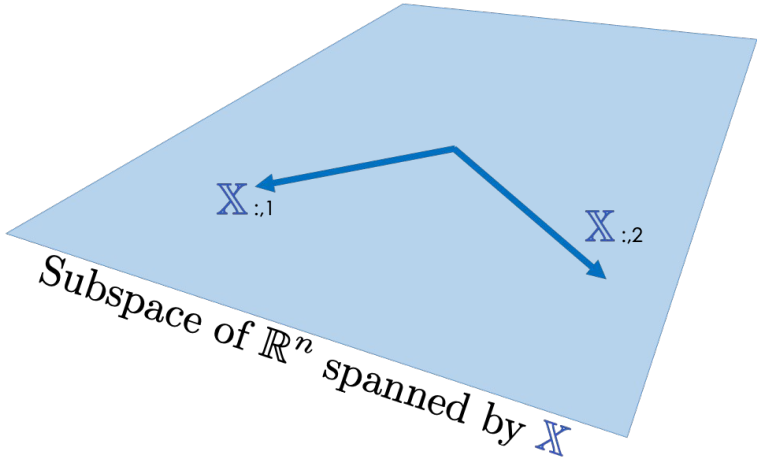
4. Evaluate model performance

Visualize,
~~Root MSE~~
Multiple R^2

[Linear Algebra] Span

The set of all possible linear combinations of the columns of $\mathbb{X}$ is called the **span** of the columns of $\mathbb{X}$ (denoted $span(\mathbb{X})$), also called the **column space**.

- Intuitively, this is all of the vectors you can "reach" using the columns of $\mathbb{X}$.
- If each column of $\mathbb{X}$ has length n, $span(\mathbb{X})$ is a subspace of $\mathbb{R}^n$.



A Linear Combination of Columns

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

So far, we've thought of our model as horizontally stacked predictions per datapoint:

$$\underset{n}{\begin{pmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{pmatrix}}_{1} = \begin{bmatrix} \text{——} x_1^T \text{——} \\ \text{——} x_2^T \text{——} \\ \vdots \\ \text{——} x_n^T \text{——} \end{bmatrix} \underset{1}{\begin{pmatrix} | \\ \theta \\ | \end{pmatrix}}_{p+1}$$

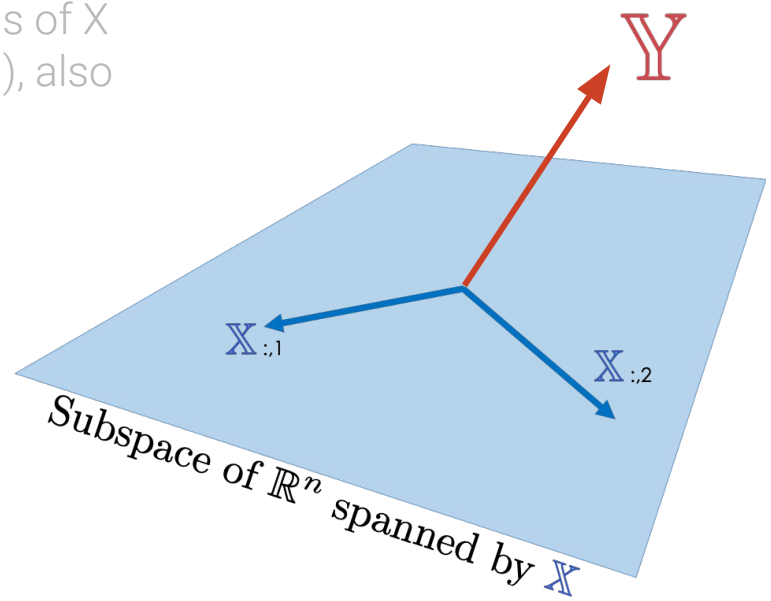
We can also think of $\hat{\mathbb{Y}}$ as a **linear combination of feature vectors**, scaled by **parameters**.

$$\underset{n}{\begin{pmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{pmatrix}}_{1} = \underset{n}{\begin{pmatrix} | & | \\ \mathbb{X}_{:,1} & \mathbb{X}_{:,2} \\ | & | \end{pmatrix}}_{p+1} \underset{1}{\begin{pmatrix} | \\ \theta \\ | \end{pmatrix}}_{p+1} = \theta_1 \begin{pmatrix} | \\ \mathbb{X}_{:,1} \\ | \end{pmatrix} + \theta_2 \begin{pmatrix} | \\ \mathbb{X}_{:,2} \\ | \end{pmatrix} + \ldots$$

A Linear Combination of Columns

The set of all possible linear combinations of the columns of X is called the **span** of the columns of X (denoted $span(\mathbb{X})$), also called the **column space**.

- Intuitively, this is all of the vectors you can "reach" using the columns of X.
- If each column of X has length n, $span(\mathbb{X})$ is a subspace of $\mathbb{R}^n$.

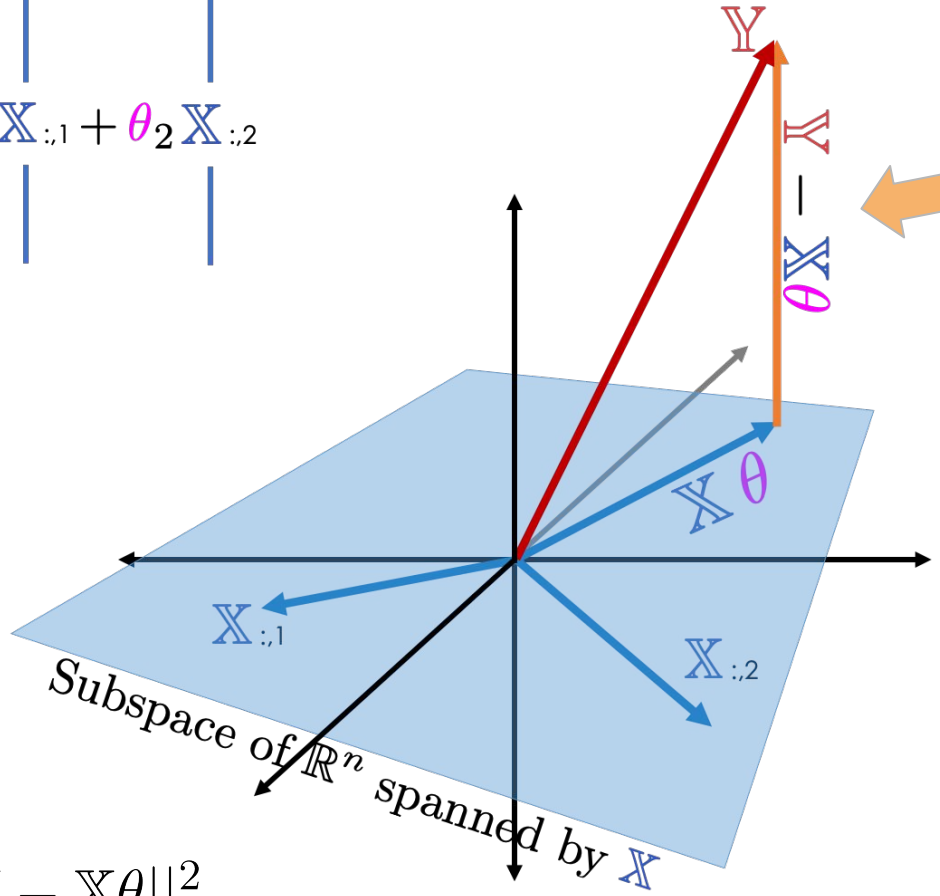


Our prediction $\hat{\mathbb{Y}} = \mathbb{X}\theta$ is a **linear combination** of the columns of $\mathbb{X}$. Therefore $\hat{\mathbb{Y}} \in span(\mathbb{X})$.

Interpret: Our linear prediction $\hat{\mathbb{Y}}$ will be in $span(\mathbb{X})$, even if the true values $\mathbb{Y}$ might not be.

Goal: Find the vector in $span(\mathbb{X})$ that is **closest** to $\mathbb{Y}$.

$$\begin{bmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2}$$

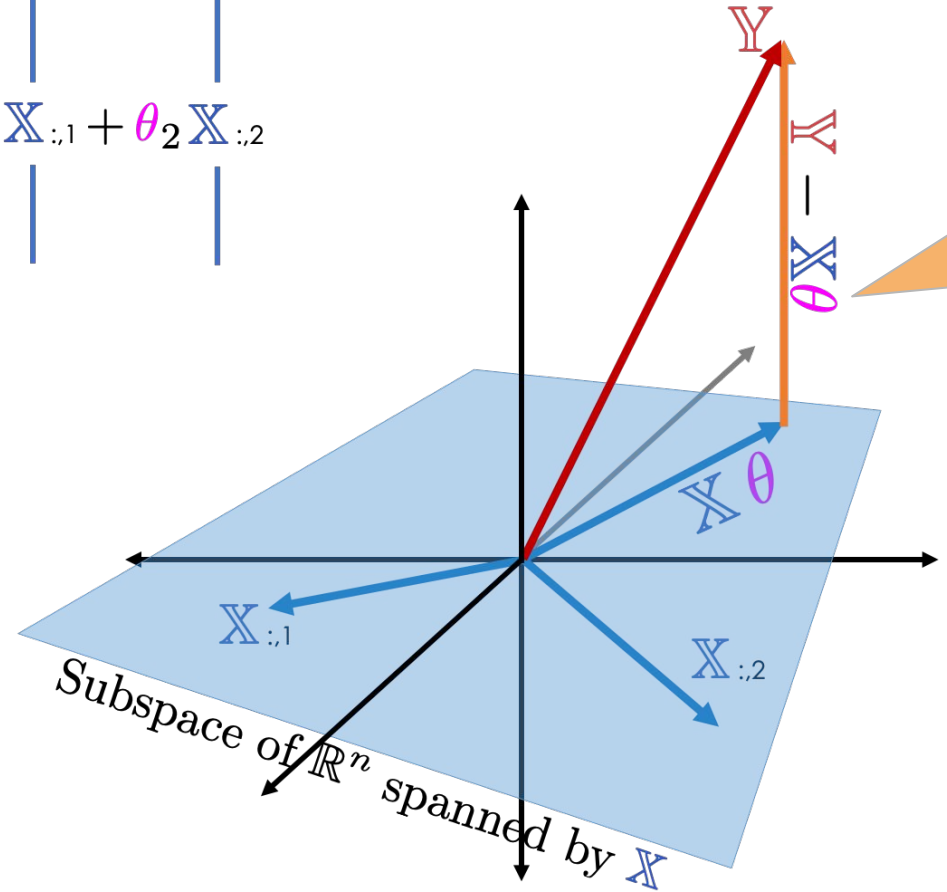


This is the residual vector, $e = \mathbb{Y} - \hat{\mathbb{Y}}$.

Goal:
Minimize the *L2* norm of the residual vector.
i.e., get the predictions $\hat{\mathbb{Y}}$ to be "as close" to our true $\mathbb{Y}$ values as possible.

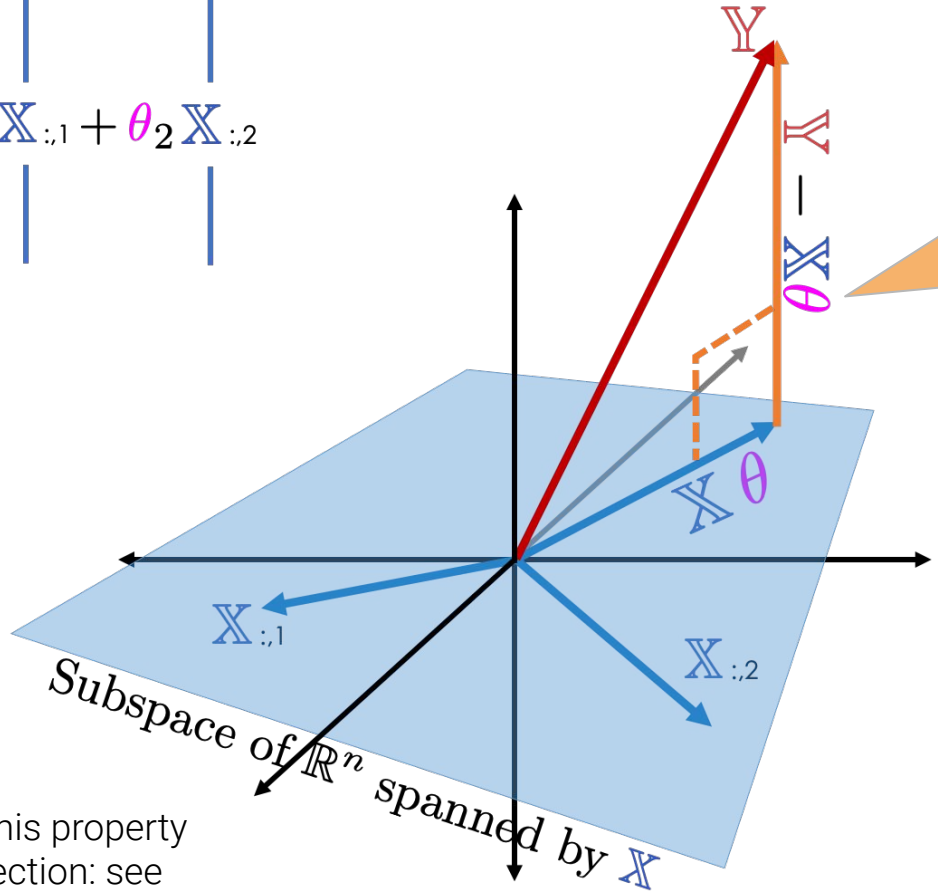
$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

$$\begin{bmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2}$$



How do we minimize this distance – the norm of the residual vector (squared)?

$$\begin{bmatrix} \\ \hat{\mathbb{Y}} \\ \\ \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2}$$

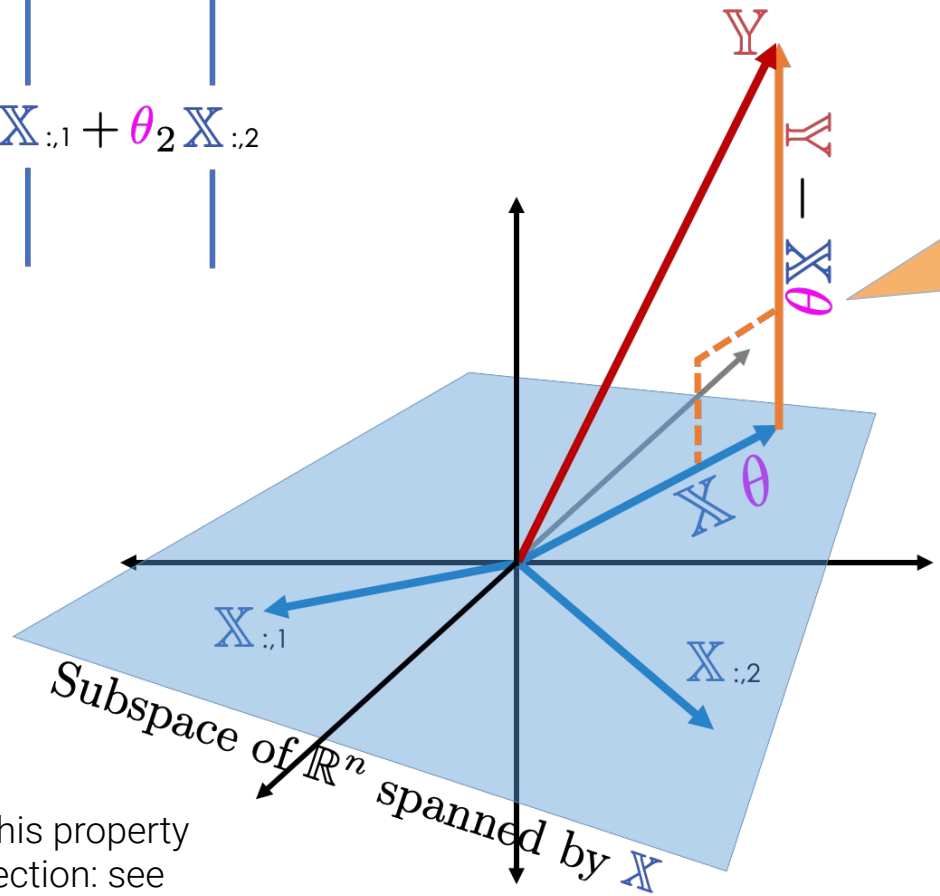


How do we minimize this distance – the norm of the residual vector (squared)?

The vector in $span(\mathbb{X})$ that is closest to $\mathbb{Y}$ is the **orthogonal projection** of $\mathbb{Y}$ onto $span(\mathbb{X})$.

We will not prove this property of orthogonal projection: see Khan Academy.

$$\begin{bmatrix} \\ \hat{\mathbb{Y}} \\ \\ \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2}$$



How do we minimize this distance – the norm of the residual vector (squared)?

The vector in $span(\mathbb{X})$ that is closest to $\mathbb{Y}$ is the **orthogonal projection** of $\mathbb{Y}$ onto $span(\mathbb{X})$.

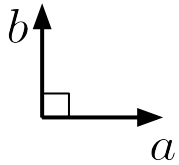
Thus, we should choose the θ that makes the residual vector **orthogonal** to $span(\mathbb{X})$.

We will not prove this property of orthogonal projection: see Khan Academy.

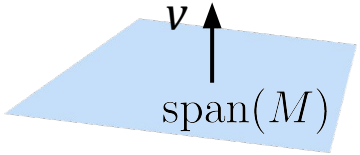
[Linear Algebra] Orthogonality

1. Vector a and Vector b are **orthogonal** if and only if their dot product is 0: $a \cdot b = 0$

This is a generalization of the notion of two vectors in 2D being perpendicular.



2. A vector v is **orthogonal** to $\text{span}(M)$, the span of the columns of a matrix M, if and only if v is orthogonal to **each column** in M.



Let's express **2** in matrix notation. Let $v \in \mathbb{R}^{n \times 1}$, $M \in \mathbb{R}^{n \times d}$ where $M = \begin{bmatrix} | & | & & | \\ m_1 & m_2 & \dots & m_d \\ | & | & & | \end{bmatrix}$:

$$\begin{aligned} m_1^T v &= 0 \\ m_2^T v &= 0 \\ &\vdots \\ m_d^T v &= 0 \end{aligned}$$

v is orthogonal to each column of M, $m_j \in \mathbb{R}^{n \times 1}$

$$\begin{bmatrix} m_1^T v \\ m_2^T v \\ \vdots \\ m_d^T v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\underbrace{M^T}_{M^T \in \mathbb{R}^{d \times n}} v = \underbrace{\vec{0}}_{\text{zero vector}}$$

zero vector (d-length vector full of 0s).

Ordinary Least Squares Proof

The **least squares estimate** $\hat{\theta}$ is the parameter θ that minimizes the objective function $R(\theta)$:

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

Design matrix

$$M^T v = \vec{0}$$

Residual vector

Equivalently, this is the $\hat{\theta}$ such that the residual vector $\mathbb{Y} - \mathbb{X}\hat{\theta}$ is orthogonal to $span(\mathbb{X})$.

Definition of orthogonality of $\mathbb{Y} - \mathbb{X}\hat{\theta}$ to $span(\mathbb{X})$ (0 is the $\vec{0}$ vector)

$$\mathbb{X}^T\left(\mathbb{Y} - \mathbb{X}\hat{\theta}\right) = 0$$

Rearrange terms

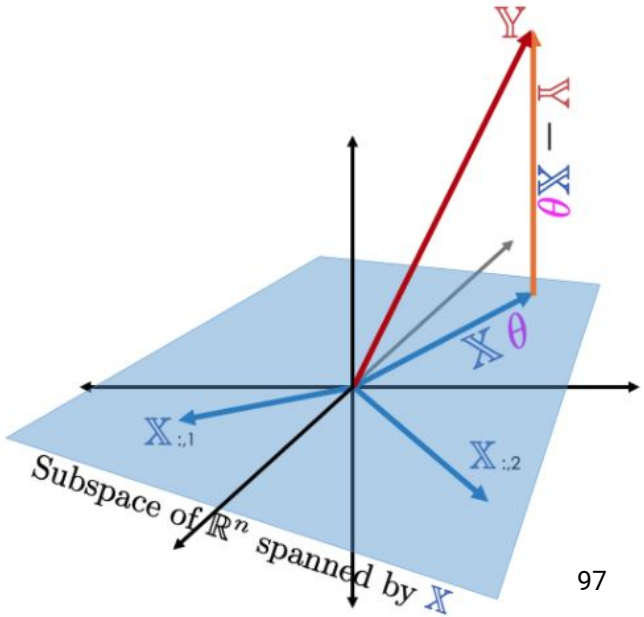
$$\mathbb{X}^T\mathbb{Y} - \mathbb{X}^T\mathbb{X}\hat{\theta} = 0$$

The **normal equation**

$$\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$$

If $\mathbb{X}^T\mathbb{X}$ is invertible

$$\hat{\theta} = \left(\mathbb{X}^T\mathbb{X}\right)^{-1}\mathbb{X}^T\mathbb{Y}$$



2327602

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$$

This result is so important that it deserves its own slide.

It is the **least squares estimate** and the solution to the normal equation $\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$.

2327602

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$$

This result is so important that it deserves its own slide.

It is the **least squares estimate** and the solution to the normal equation $\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$.

Least Squares Estimate

1. Choose a model

Multiple Linear Regression

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

2. Choose a loss function

L2 Loss

Mean Squared Error (MSE)

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

3. Fit the model ✅

Minimize average loss with ~~calculus~~ geometry

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$$

4. Evaluate model performance

Visualize,
~~Root MSE~~
Multiple R^2