LECTURE 14

Gradient Descent II, Feature Engineering

Finishing Optimization. Transforming Data to Improve Our Models.

Data 100/Data 200, Fall 2022 @ UC Berkeley

Narges Norouzi and Lisa Yan

slido

Join at slido.com
#3819148

ⓘ Start presenting to display the joining instructions on this slide.

Partial Derivatives: Review

Suppose we had a linear model
with two features and MSE loss, no intercept.

$$f_{\vec{\theta}}(\vec{x}) = \vec{x}^T\vec{\theta} = \theta_0 x_0 + \theta_1 x_1$$

If we just consider the **a single observation**,
then the **per-datapoint loss** function is:

$$L(\vec{\theta}, \vec{x}, y) = (y \overbrace{- \theta_0 x_0 - \theta_1 x_1}^{-\hat{y}})^2$$

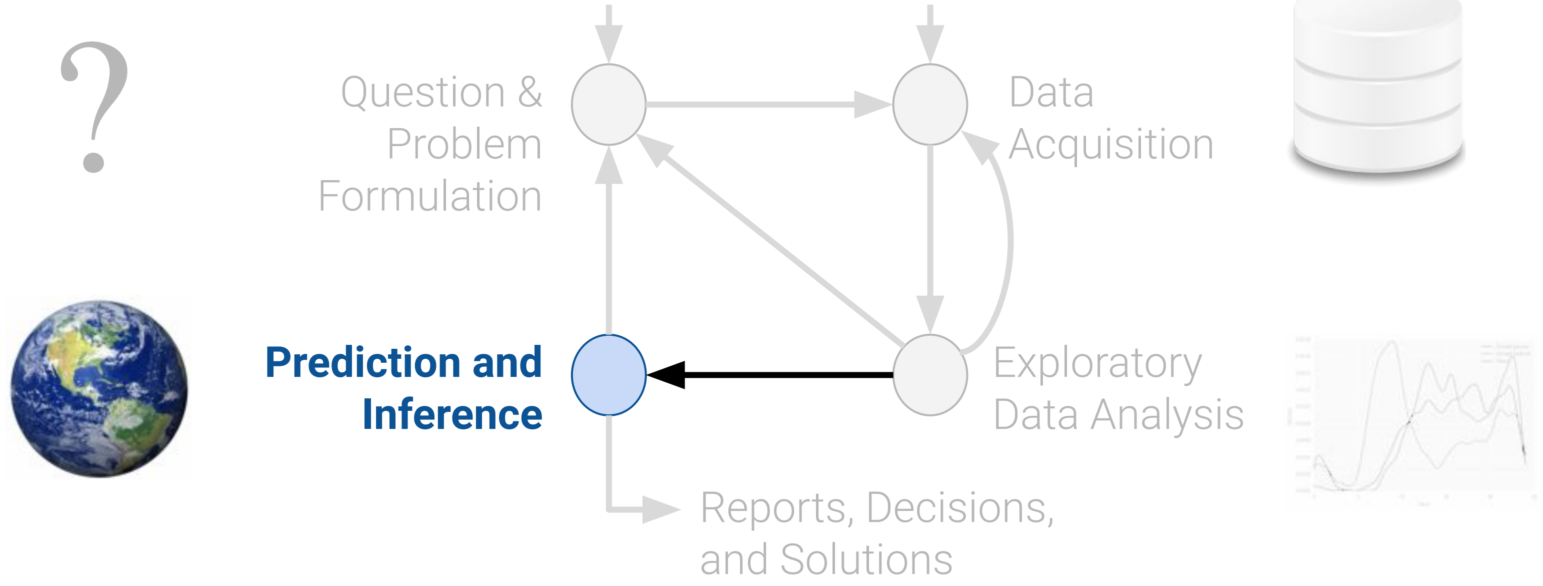
1. $$\frac{\partial}{\partial \theta_0} L(\vec{\theta}, \vec{x}, y) = \ ?$$

2. $$\frac{\partial}{\partial \theta_1} L(\vec{\theta}, \vec{x}, y) = \ ?$$

(no slido here)

Plan for Lectures 12 and 13: Model Implementation

3819148



(today)

Model Implementation I: sklearn Gradient Descent	**Model Implementation II:** Gradient Descent Feature Engineering

Today's Roadmap

Lecture 14, Data 100 Spring 2023

3819148

Gradient Descent in Higher Dimensions

Mini-Batch and Stochastic Gradient Descent

Feature Engineering

One-Hot Encoding

Higher-Order Polynomial Example

Overfitting

Variance and Training Error

[Extra] Convexity

[Extra] Deciding Overfitting

Partial Derivatives: Review

Suppose we had a linear model
with two features and MSE loss, no intercept.

$$f_{\vec{\theta}}(\vec{x}) = \vec{x}^T\vec{\theta} = \theta_0 x_0 + \theta_1 x_1$$

If we just consider the **a single observation**,
then the **per-datapoint loss** function is:

$$L(\vec{\theta}, \vec{x}, y) = (y - \theta_0 x_0 - \theta_1 x_1)^2$$

1. $$\frac{\partial}{\partial \theta_0} L(\vec{\theta}, \vec{x}, y) = 2(y - \theta_0 x_0 - \theta_1 x_1)(-x_0)$$
2. $$\frac{\partial}{\partial \theta_1} L(\vec{\theta}, \vec{x}, y) = 2(y - \theta_0 x_0 - \theta_1 x_1)(-x_1)$$

Partial Derivatives: Review

Suppose we had a linear model
with two features and MSE loss, no intercept.

$$f_{\vec{\theta}}(\vec{x}) = \vec{x}^T \vec{\theta} = \theta_0 x_0 + \theta_1 x_1$$

If we just consider the **a single observation**,
then the **per-datapoint loss** function is:

$$L(\vec{\theta}, \vec{x}, y) = (y - \theta_0 x_0 - \theta_1 x_1)^2$$

$$\nabla_{\vec{\theta}} L(\vec{\theta}, \vec{x}, y) = \begin{bmatrix} 2(y - \theta_0 x_0 - \theta_1 x_1)(-x_0) \\ \\ 2(y - \theta_0 x_0 - \theta_1 x_1)(-x_1) \end{bmatrix}$$

Congratulations! You just computed your first **gradient**!

3819148

Gradient Descent in Higher Dimensions

Lecture 14, Data 100 Spring 2023

[Review] Gradient Descent in 1-D

Gradient descent allows us to find a minima of functions.

The idea: nudge θ in negative slope direction until θ converges.

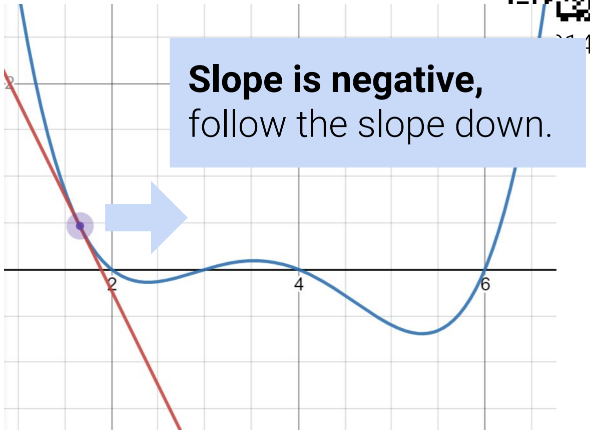
$$x^{(t+1)} = x^{(t)} - \alpha \underbrace{\frac{d}{dx} f(x^{(t)})}$$

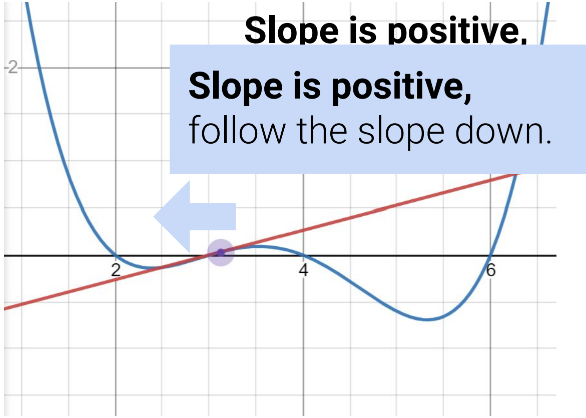
next solution

current solution

learning rate

Slope of the function we're minimizing @ **current solution**.





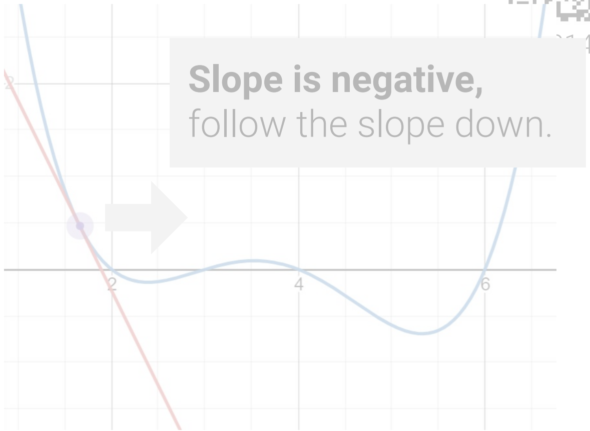
[Review] Gradient Descent in 1-D

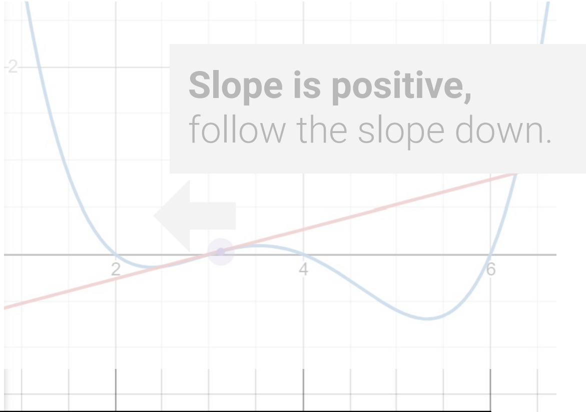
Gradient descent allows us to find the minima of functions.

The idea: nudge θ in negative slope direction until θ converges.

$$x^{(t+1)} = x^{(t)} - \alpha \underbrace{\frac{d}{dx} f(x^{(t)})}_{}$$

next solution | **current solution** | **learning rate** | **Slope** of the function we're minimizing @ **current solution**.





This process underlies sklearn's `LinearRegression()` to find the optimal theta that minimizes MSE loss:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{d}{d\theta} L(\theta^{(t)})$$

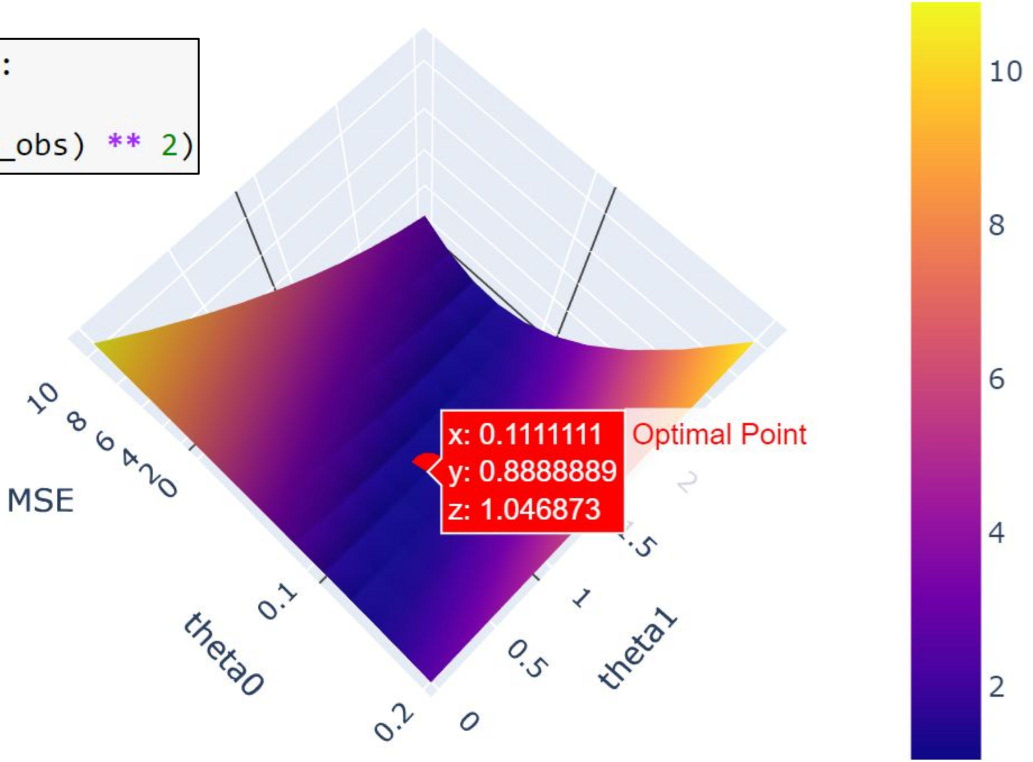
But what if θ is a ***vector***?

A 2D Loss Function

3819148

Here, we see the loss of our model as a function of our two parameters.

```python
def mse_loss(theta, X, y_obs):
    y_hat = X @ theta
    return np.mean((y_hat - y_obs) ** 2)
```



The Gradient is the Vector of First Derivatives

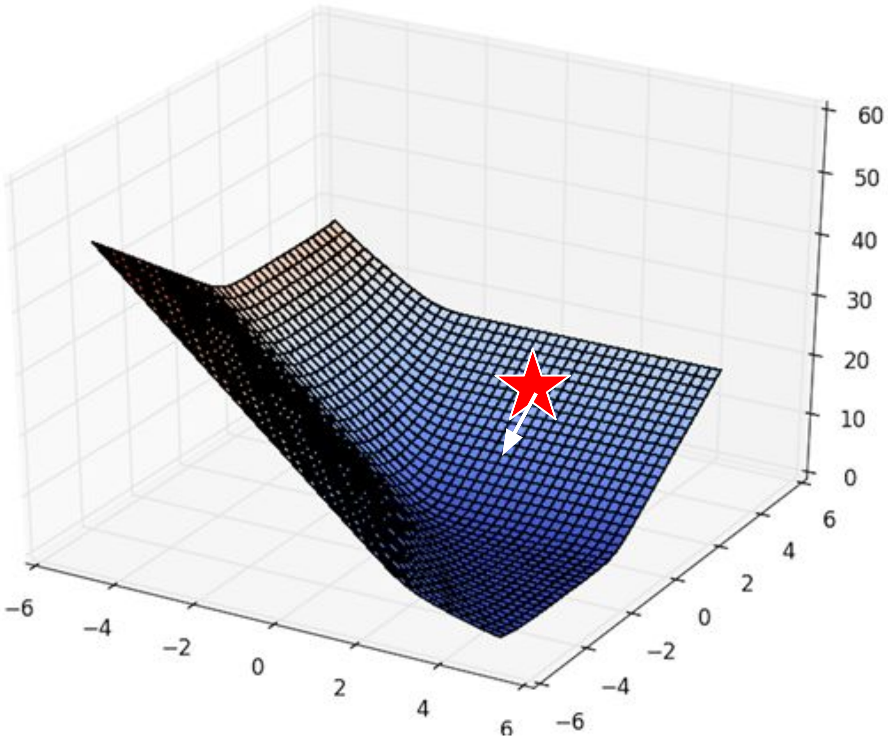
Just like earlier, we can follow the slope of our 2D function.

- On a 2D surface, the best way to go down (**gradient**) is described by a 2D vector.

The Gradient is the Vector of First Derivatives

3819148

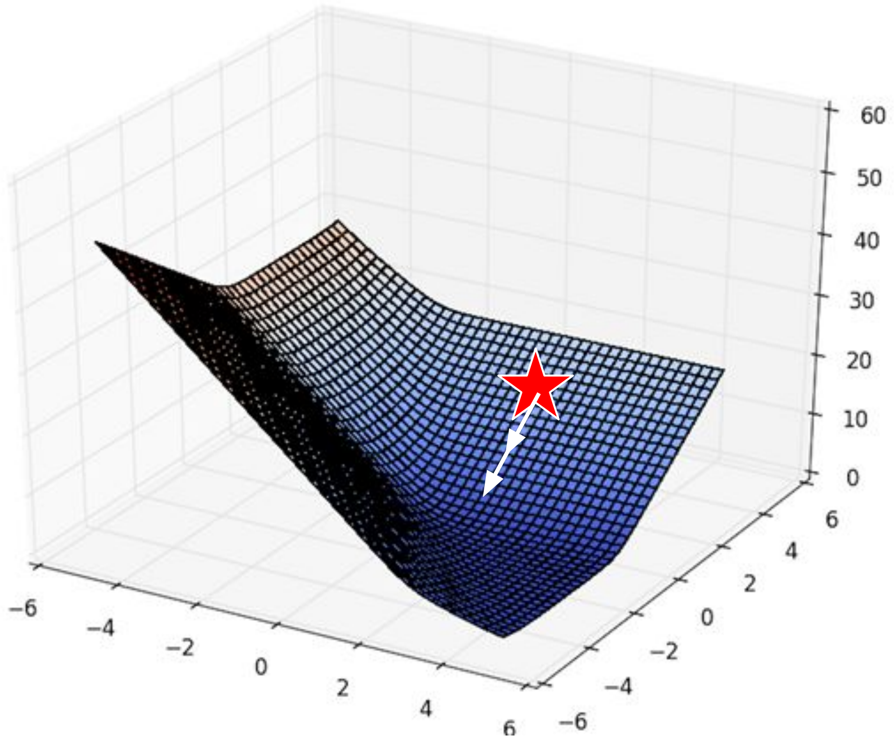
On a 2D surface, the best way to go down is described by a 2D vector.



The Gradient is the Vector of First Derivatives

3819148

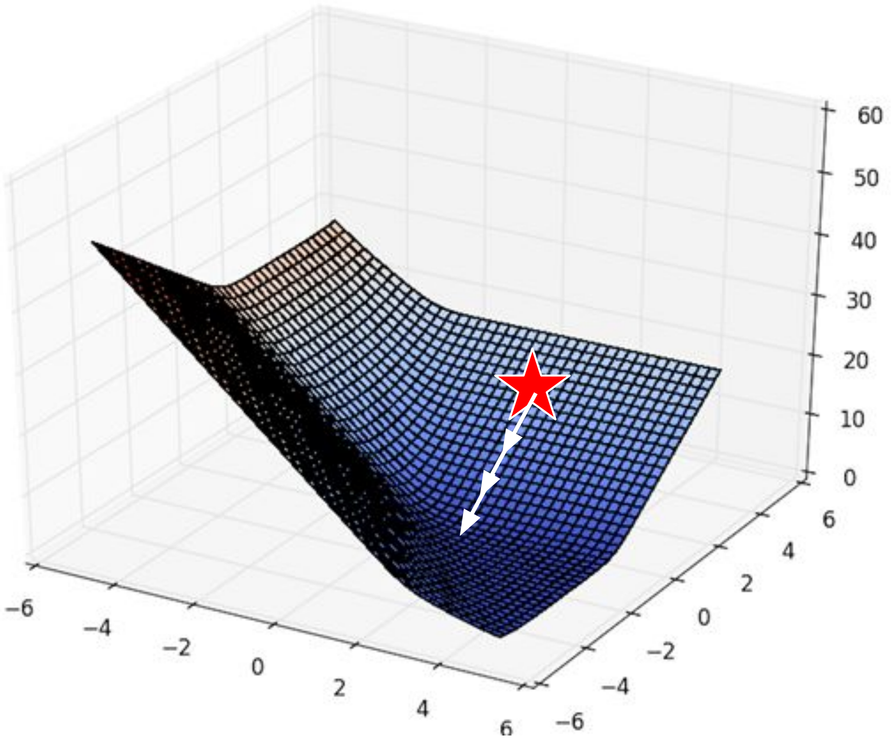
On a 2D surface, the best way to go down is described by a 2D vector.



The Gradient is the Vector of First Derivatives

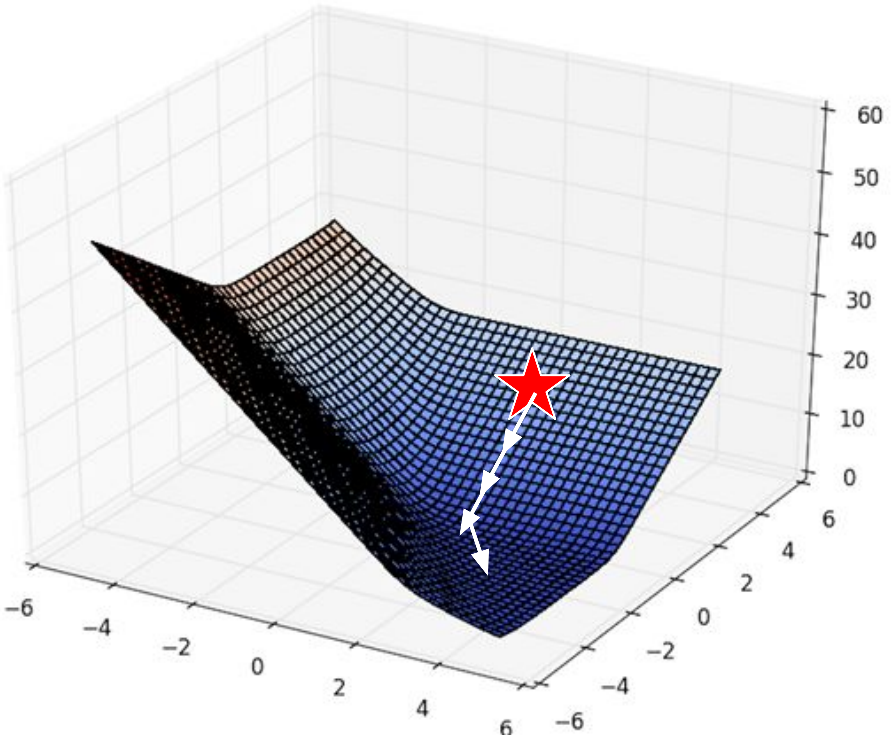
On a 2D surface, the best way to go down is described by a 2D vector.

3819148



The Gradient is the Vector of First Derivatives

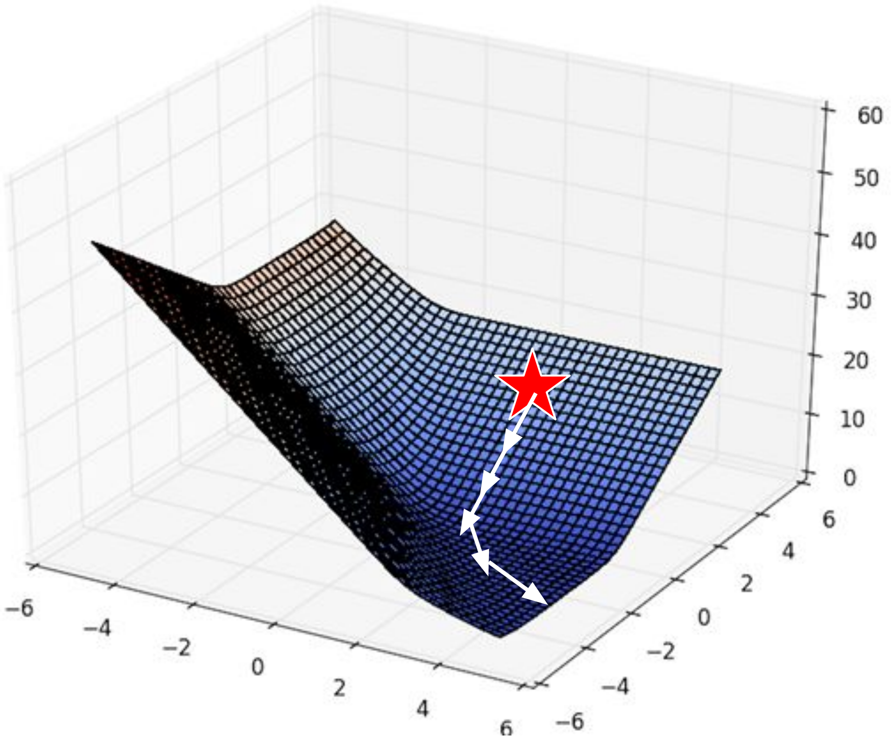
On a 2D surface, the best way to go down is described by a 2D vector.



The Gradient is the Vector of First Derivatives

3819148

On a 2D surface, the best way to go down is described by a 2D vector.



Example: Gradient of a 2D Function

3819148

Consider the 2D function: $$f(\theta_0, \theta_1) = 8\theta_0^2 + 3\theta_0\theta_1$$

$$\frac{\partial f}{\partial \theta_0} = 16\theta_0 + 3\theta_1$$

$$\frac{\partial f}{\partial \theta_1} = 3\theta_0$$

$$\underbrace{\nabla_{\vec{\theta}} f(\vec{\theta})} = \begin{bmatrix} 16\theta_0 + 3\theta_1 \\ 3\theta_0 \end{bmatrix}$$

Slope in θ_0 direction

Slope in θ_1 direction

Gradient w.r.t. (with respect to) elements of $\vec{\theta}$

Example: Gradient for L2 datapoint loss on a 2-feature model

Suppose we had a linear model
with two features and MSE loss, no intercept.

$$f_{\vec{\theta}}(\vec{x}) = \vec{x}^T\vec{\theta} = \theta_0 x_0 + \theta_1 x_1$$

If we just consider the **a single observation**,
then the **per-datapoint loss** function is:

$$L(\vec{\theta}, \vec{x}, y) = (y - \theta_0 x_0 - \theta_1 x_1)^2$$

$$\nabla_{\vec{\theta}} L(\vec{\theta}, \vec{x}, y) = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \end{bmatrix} = \begin{bmatrix} -2(y_i - \theta_0 x_0 - \theta_1 x_1)(x_0) \\ -2(y_i - \theta_0 x_0 - \theta_1 x_1)(x_1) \end{bmatrix}$$

Interpret gradient:

- If I nudge the 1st model parameter θ_0, what happens to loss?
- If I nudge the 2nd θ_1, what happens to loss?

Gradient of MSE in practice

Suppose we had a linear model
with two features and MSE loss, no intercept.

$$f_{\vec{\theta}}(\vec{x}) = \vec{x}^T\vec{\theta} = \theta_0 x_0 + \theta_1 x_1$$

Gradient descent uses the gradient with respect to θ of the **MSE loss function of the entire dataset**, not just per-datapoint loss.

That gradient derivation is left to you!

$$L(\vec{\theta}, \mathbb{X}, \mathbb{Y}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \vec{x}_i^T\vec{\theta}\right)^2$$

$$\nabla_{\vec{\theta}} L(\vec{\theta}, \vec{x}, y) = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \end{bmatrix} = \ ?$$

Interpret gradient:

- If I nudge the 1st model parameter θ_0, what happens to loss?
- If I nudge the 2nd θ_1, what happens to loss?

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L(\vec{\theta}^{(t)}, \mathbb{X}, \mathbb{Y})$$

Summary: Gradient Descent

3819148

Gradient descent algorithm: nudge θ in negative gradient direction until θ converges.

For a model with multiple parameters:

$$\underbrace{\vec{\theta}^{(t+1)}}_{\text{Next value for } \theta} = \vec{\theta}^{(t)} - \alpha \underbrace{\nabla_{\vec{\theta}} L(\vec{\theta}^{(t)}, \mathbb{X}, \mathbb{Y})}_{\text{gradient of the loss function evaluated at current } \theta}$$

Convexity property of MSE loss guarantees global minima. (See extra slides/recording: [Sp23](Sp23))

θ: Model weights **L**: loss function

α: Learning rate (ours is constant; other techniques have α decrease over time)

X: design matrix, **Y**: True values from training data

Mini-Batch and Stochastic Gradient Descent

Lecture 14, Data 100 Spring 2023

3819148

Computing the optimal $\hat{\theta}$ that minimizes loss

3819148

Analytical solution:

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

Impractical when # datapoints is huge.

- Inverting matrices is computationally slow!
- See CS61B, CS61C

"**Batch**" gradient descent:

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha\nabla_{\vec{\theta}}L(\vec{\theta}^{(t)}, \mathbb{X}, \mathbb{Y})$$

Impractical when # datapoints is huge.

- Computing gradient of loss is slow!
- Will converge very slowly due to time to compute gradient in each step.

Imagine you have *billions* of data points.

- Computing the analytical inverse would require matrix-multiplying a billion-datapoint design matrix multiple times.
- Computing the gradient would require computing the loss for a prediction for EVERY data point, then computing the mean loss across all several billion.

Approximating the optimal $\hat{\theta}$ that minimizes loss

3819148

Analytical solution:

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

Impractical when # datapoints is huge.

- Inverting matrices is computationally slow!
- See CS61B, CS61C

"**Batch**" gradient descent:

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha\nabla_{\vec{\theta}}L(\vec{\theta}^{(t)}, \mathbb{X}, \mathbb{Y})$$

Impractical when # datapoints is huge.

- Computing gradient of loss is slow!
- Will converge very slowly due to time to compute gradient in each step.

Minibatch gradient descent

Compute gradient of loss over a **fraction** of data (e.g., 32 datapoints.) to update of θ. Repeat a lot (e.g., 32 times # steps for batch grad. desc).

Stochastic gradient descent:

Compute gradient of loss over a **single**, randomly picked datapoint to update θ. Repeat ***a lot***.

Both algorithms are **approximations** that speed up gradient descent computation in practice.

Mini-Batch Gradient Descent

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L(\vec{\theta}^{(t)},$$

3819148

In **mini-batch gradient descent**, we only use a subset of the data to compute the gradient.

Example:

- Compute gradient on first 10% of the data. Adjust parameters θ.
- Then compute gradient on next 10% of the data. Adjust parameters θ.
- Then compute gradient on third 10% of the data. Adjust parameters θ.
- …
- Then compute gradient on last 10% of the data. Adjust parameters θ.

Are we done now?

Not unless we were lucky! We've only approximated computing the entire gradient one time.

So what should we do next?

Go through the data again.

Mini-Batch Gradient Descent, with Epochs

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L(\vec{\theta}^{(t)},$$

3819148

In **mini-batch gradient descent**, we only use a subset of the data to compute the gradient.

Example:

Repeat:

- Compute gradient on first 10% of the data. Adjust parameters θ.
- Then compute gradient on next 10% of the data. Adjust parameters θ.
- Then compute gradient on third 10% of the data. Adjust parameters θ.
- …
- Then compute gradient on last 10% of the data. Adjust parameters θ.

Training Epoch

Stop when we either:

- Hit some max number of iterations, or
- Our error is below some desired threshold,
  i.e., θ does not change significantly between iterations.

Mini-Batch Gradient Descent in Practice

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L(\vec{\theta}^{(t)},$$

3819148

Important: The gradient we compute using only 10% of our data is **not** the true gradient.

- It's merely an approximation; likely not the best way down the true loss surface.
- However, it works well in practice.

Batch size: number of datapoints to incorporate into gradient loss computation in a step.

- Batch size represents quality of gradient approximation:
  - All of the data: true gradient (**batch** gradient descent),
  - 10% of the data: approximation of gradient, etc.
- In real world practice, mini-batch size is a fixed number, e.g., 32 datapoints.
  - n datapoints $\rightarrow n/32$ mini-batch updates (steps) per epoch.
  - Why fixed, regardless of dataset size? See ML literature.

Other tips: **Shuffle** data in-between training epochs, instead of always traversing in order of our original dataset. Details beyond the scope of this course.

Stochastic Gradient Descent

3819148

In the most extreme case, we choose a **batch size of 1**.

This is called **stochastic gradient descent**.

- Gradient is computed using only a single (random) data point!
- n datapoints $\rightarrow$ n steps per epoch.

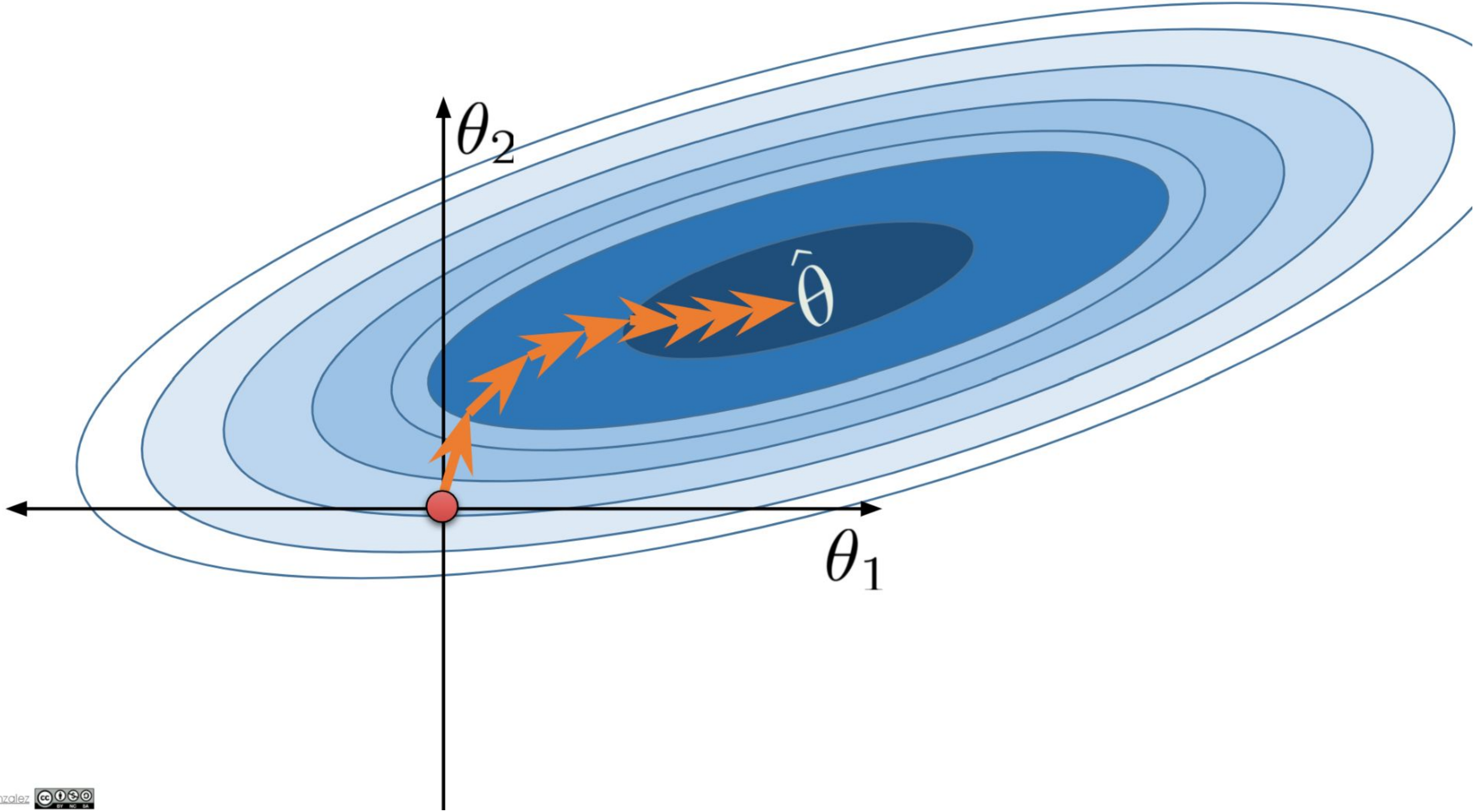
It may surprise you, but this actually works on real world datasets. :-)

- Imagine training an algorithm that recognizes pictures of dogs. Training based on only one dog image at a time means updating potentially millions of parameters based on a single image.
- Intuition: If we **average across many epochs** across the entire dataset, the effect is similar to if we simply compute the true gradient based on the entire dataset.

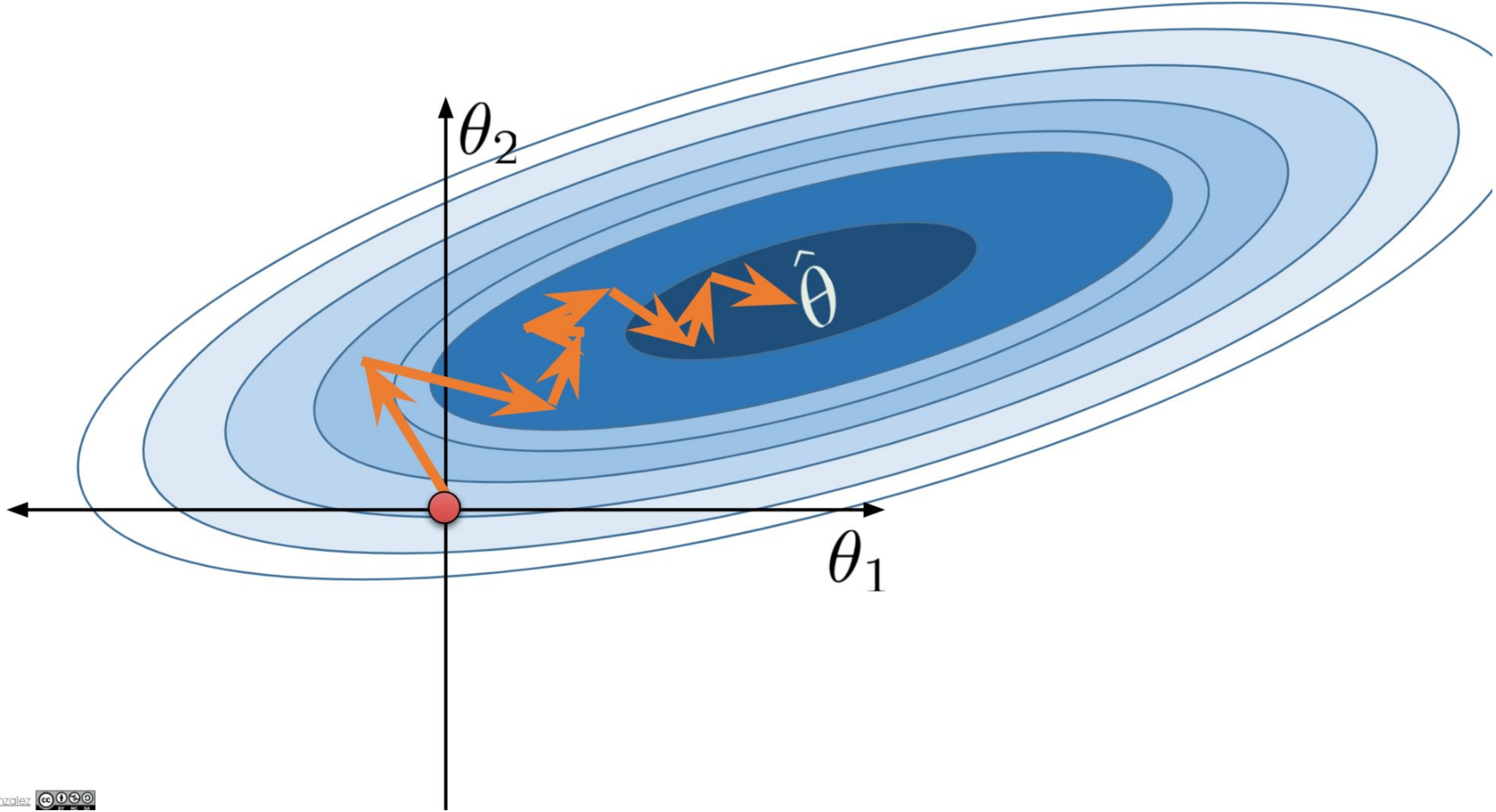
Note: some practitioners use the terms “stochastic gradient descent” and “mini-batch gradient descent” interchangeably, but we will avoid this in this class.

Gradient Descent

3819148



Stochastic Gradient Descent



slido

Suppose we fit an Ordinary Least Squares model to a 8192 datapoint dataset using **stochastic** gradient descent. How many updates are there per epoch?

ⓘ Start presenting to display the poll results on this slide.

Interlude

The loss functions we discuss in Data 100 are convex. See extra slides/recording: [Sp23]

Feature Engineering

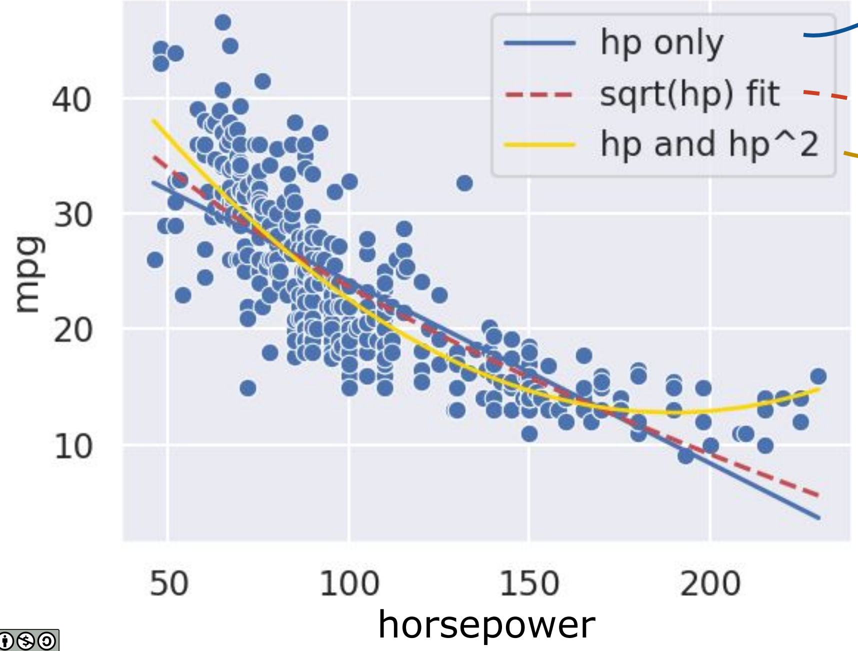
Lecture 14, Data 100 Spring 2023

In lab this week, you saw the “Linear” Parabolic relationship

3819148

Fuel efficiency vs. engine power of different car models.

- y: Fuel efficiency in **miles per gallon** (similar to liters / kilometer).
- x: Total engine power in **horsepower** (1 horsepower = 745.7 watts).



$$\hat{y} = \theta_0 + \theta_1 x$$

$$\hat{y} = \theta_0 + \theta_1 \sqrt{x}$$

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$$

We used `LinearRegression()` to fit all these models, despite polynomial/root terms!

How? By **engineering new features**.

Feature Engineering

Feature Engineering is the process of **transforming** the raw features **into more informative features** that can be used in modeling or EDA tasks.

Feature engineering allows you to:

- Capture domain knowledge (e.g. periodicity or relationships between features).
- Express non-linear relationships using simple linear models.
- Encode non-numeric features to be used as inputs to models.
  - Example: Using the country of origin of a car as an input to modeling its efficiency.

Why doesn't sklearn doesn't have `SquareRegression/PolynomialRegression`.

- We can translate these into linear models with features that are polynomials of x.
- Feature engineering saves sklearn a lot of redundancy in their library.
- As you saw in homework, linear models have really nice properties.

Feature Function

3819148

A **Feature Function** takes

our original **d dimensional input** X and **transforms** it into a **d' dimensional input** Φ.

$$\mathbb{X} \in \mathbb{R}^{n \times d} \quad \Rightarrow \quad \Phi \in \mathbb{R}^{n \times d'}$$

Our feature function took d=1 dimensional input [hp] and transformed it into d'=2 dimensional input [hp, hp^2].

Often, d' >> d.

- As number of features grows, we can capture arbitrarily complex relationships.

	hp	mpg
0	130.00	18.00
1	165.00	15.00
2	150.00	18.00
...	...	...
395	84.00	32.00
396	79.00	28.00
397	82.00	31.00

392 rows × 2 columns

	hp	hp^2	mpg
0	130.00	16900.00	18.00
1	165.00	27225.00	15.00
2	150.00	22500.00	18.00
...	...	...	...
395	84.00	7056.00	32.00
396	79.00	6241.00	28.00
397	82.00	6724.00	31.00

392 rows × 3 columns

Feature Function

3819148

A **Feature Function** takes our original **d dimensional input** X and **transforms** it into a **d' dimensional input** Φ.

$$\mathbb{X} \in \mathbb{R}^{n \times d} \quad \Rightarrow \quad \Phi \in \mathbb{R}^{n \times d'}$$

Linear models trained on transformed data are sometimes written using the symbol Φ instead of X:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 \quad \Rightarrow \quad \hat{y} = \theta_0 + \theta_1 \phi_1 + \theta_2 \phi_2$$

$$\hat{\mathbb{Y}} = \mathbb{X}\theta \quad \Rightarrow \quad \hat{\mathbb{Y}} = \Phi\theta$$

Check out [this video] where Professor Joey Gonzalez transforms 2-D input into 15-D input…

Feature Function

3819148

A **Feature Function** takes
our original **d dimensional input** X and **transforms** it into a **d' dimensional input** Φ.

$$\mathbb{X} \in \mathbb{R}^{n \times d} \Rightarrow \Phi \in \mathbb{R}^{n \times d'}$$

Designing feature functions is a **major part** of data science and machine learning.

- You'll have a chance to do lots of feature function design on project 1.
- Fun fact: Much of the success of modern deep learning is because some models have the ability to automatically learn feature functions. See CS W182/282A (Deep Learning) for more.

Check out [this video] where Professor Joey Gonzalez transforms 2-D input into 15-D input…

One Hot Encoding

Lecture 14, Data 100 Spring 2023

3819148

Regression Using Non-Numeric Features

3819148

We can also perform regression on non-numeric features. For example, for the tips dataset from last lecture, we might want to use the day of the week.

- One problem: Our linear model is always a linear combination of our features. Unclear at first how you'd do this.

$$\hat{y} = \theta_1 \times bill + \theta_2 \times size + \theta_3 \times day$$

??

total_bill	tip	sex	smoker	day	time	size
28.97	3.00	Male	Yes	Fri	Dinner	2
17.81	2.34	Male	No	Sat	Dinner	4
13.37	2.00	Male	No	Sat	Dinner	2
15.69	1.50	Male	Yes	Sun	Dinner	2
15.48	2.02	Male	Yes	Thur	Lunch	2

Non-Numeric Features: One-Hot Encoding

3819148

One approach is to use what is known as **one-hot encoding**.

- Each category of a categorical variable gets its own feature:
  - value = 1 if that category applies to that row
  - value = 0 otherwise.

	total_bill	size	day
193	15.48	2	Thur
90	28.97	2	Fri
25	17.81	4	Sat
26	13.37	2	Sat
190	15.69	2	Sun

	day_Thur	day_Fri	day_Sat	day_Sun
193	1.0	0.0	0.0	0.0
90	0.0	1.0	0.0	0.0
25	0.0	0.0	1.0	0.0
26	0.0	0.0	1.0	0.0
190	0.0	0.0	0.0	1.0

Use sklearn's `OneHotEncoder`!
(documentation)

Fitting a Model

3819148

If we fit a linear model, the result is a 6 dimensional model.

- θ_1 = 0.093: How much to weight the total bill.
- θ_2 = 0.187: How much to weight the party size.
- θ_3 = 0.746
- θ_4 = 0.621
- θ_5 = 0.732
- θ_6 = 0.668

How much to weight the fact that it is Friday, Saturday, Sunday, or Thursday, respectively.

	total_bill	size	day	day_Fri	day_Sat	day_Sun	day_Thur
193	15.48	2	Thur	0.0	0.0	0.0	1.0
90	28.97	2	Fri	1.0	0.0	0.0	0.0
25	17.81	4	Sat	0.0	1.0	0.0	0.0
26	13.37	2	Sat	0.0	1.0	0.0	0.0
190	15.69	2	Sun	0.0	0.0	1.0	0.0

Resulting prediction is: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

Test your understanding

3819148

If we fit a linear model, the result is a 6 dimensional model.

- θ_1 = 0.093: How much to weight the total bill.
- θ_2 = 0.187: How much to weight the party size.
- θ_3 = 0.746
- θ_4 = 0.621
- θ_5 = 0.732
- θ_6 = 0.668

How much to weight the fact that it is Friday, Saturday, Sunday, or Thursday, respectively.

	total_bill	size	day	day_Fri	day_Sat	day_Sun	day_Thur
193	15.48	2	Thur	0.0	0.0	0.0	1.0
90	28.97	2	Fri	1.0	0.0	0.0	0.0
25	17.81	4	Sat	0.0	1.0	0.0	0.0
26	13.37	2	Sat	0.0	1.0	0.0	0.0
190	15.69	2	Sun	0.0	0.0	1.0	0.0

Resulting prediction is: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

What tip would the model predict for:

- A party of 3
- With a $50 check
- Eating on a Thursday?

What tip would the model predict for a party of 3 with a $50 check eating on a Thursday?

ⓘ Start presenting to display the poll results on this slide.

Test your understanding

If we fit a linear model, the result is a 6 dimensional model.

- θ_1 = 0.093: How much to weight the total bill.
- θ_2 = 0.187: How much to weight the party size.
- θ_3 = 0.746
- θ_4 = 0.621
- θ_5 = 0.732
- θ_6 = 0.668

How much to weight the fact that it is Friday, Saturday, Sunday, or Thursday, respectively.

	total_bill	size	day	day_Fri	day_Sat	day_Sun	day_Thur
193	15.48	2	Thur	0.0	0.0	0.0	1.0
90	28.97	2	Fri	1.0	0.0	0.0	0.0
25	17.81	4	Sat	0.0	1.0	0.0	0.0
26	13.37	2	Sat	0.0	1.0	0.0	0.0
190	15.69	2	Sun	0.0	0.0	1.0	0.0

Resulting prediction is: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

What tip would the model predict for:

- A party of 3
- With a $50 check
- Eating on a Thursday?

$$\hat{y} = 0.093 \times 50 + 0.187 \times 3 + 0.668 = \$5.88$$

```
# total_bill, size, day_Fri, day_Sat, day_Sun, day_Thur
f_with_day.predict([[50, 3, 1, 0, 0, 0]])
```

Interpreting the 6 Dimensional Model?

3819148

It turns out the MSE for this one-hot-encoded, 6-dimensional model is **1.01**.

- A model trained on only the bill and the table size (2-dimensional) has an MSE of **1.06**.

This model makes slightly better predictions on this training set,
but it likely **does not represent** the true nature of the data generating process.

- Bizarre to imagine that humans have a base tip that they start with for every day of the week.
- My guess: This model will not generalize well to newly collected data.

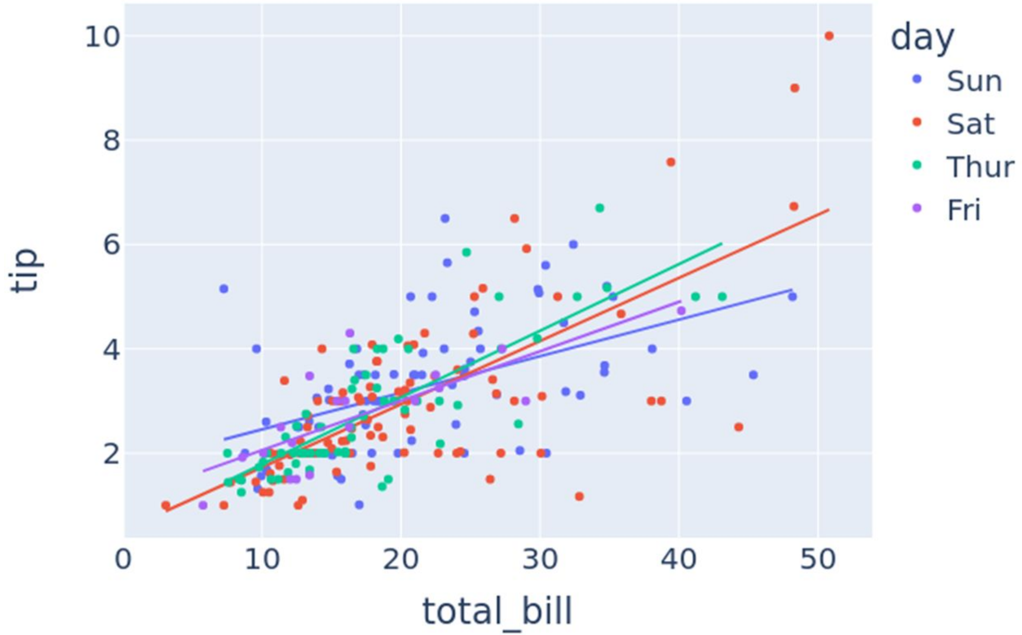
An Alternate Approach

3819148

Another approach is to fit a separate model to each condition.

- Reasonable for a small number of conditions.

```
px.scatter(data, x="total_bill", y="tip", color = "day", trendline = "ols")
```



Higher-Order Polynomial Example

Lecture 14, Data 100 Spring 2023

3819148

Cubic Fit

$x = hp$ $$\hat{y} = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_p x^p$$

3819148

Let's return to where we started today: Creating **higher-order polynomial features** for the mpg dataset.

What happens if we add a feature corresponding to the horsepower cubed?

- Will we get better results?
- What will the model look like?

Let's try it out:

hp	hp2	hp3	mpg
130.0	16900.0	2197000.0	18.0
165.0	27225.0	4492125.0	15.0
150.0	22500.0	3375000.0	18.0
150.0	22500.0	3375000.0	16.0
140.0	19600.0	2744000.0	17.0

```
vehicle_data["hp3"] = vehicle_data["hp"]**3
```

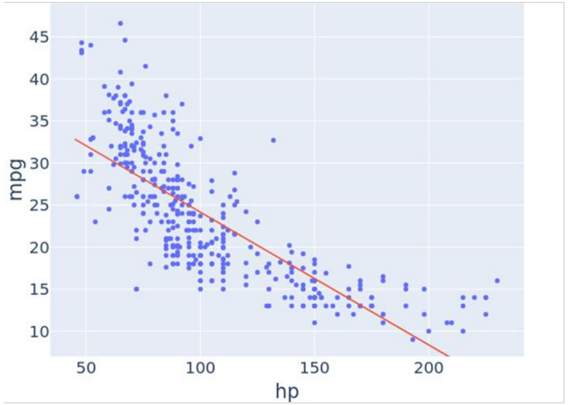
```
cu_model = LinearRegression()
cu_model.fit(vehicle_data[['hp', 'hp2', 'hp3']], vehicle_data['mpg'])
```

Cubic Fit Results

$$\hat{y} = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_p x^p$$

3819148

MSE: 21

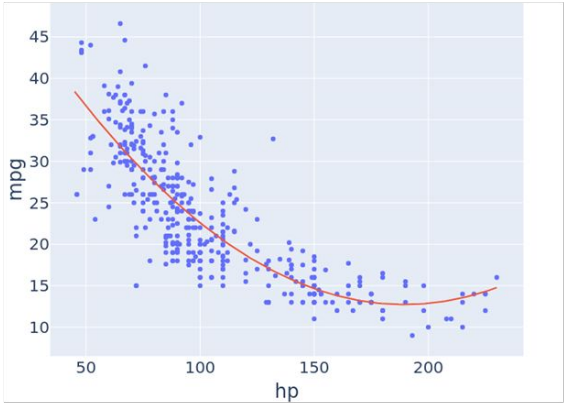


Degree 1 Features

```
fit(vehicle_data[['hp']]
```

$p = 1$

MSE: 18.98

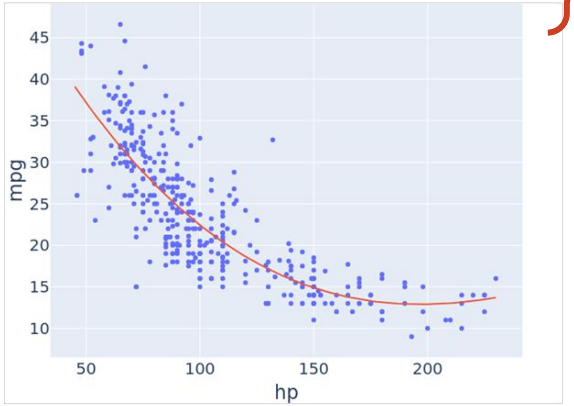


Degree 2 Features

```
fit(vehicle_data[['hp', 'hp2']]
```

$p = 2$

MSE: 18.94



Degree 3 Features

```
fit(vehicle_data[['hp', 'hp2', 'hp3']]
```

$p = 3$

We observe a **small improvement** in **MSE**.

Qualitatively, the curve looks quite similar. Only ***slightly better*** prediction power.

…but what happens if we add even higher order features??

slido

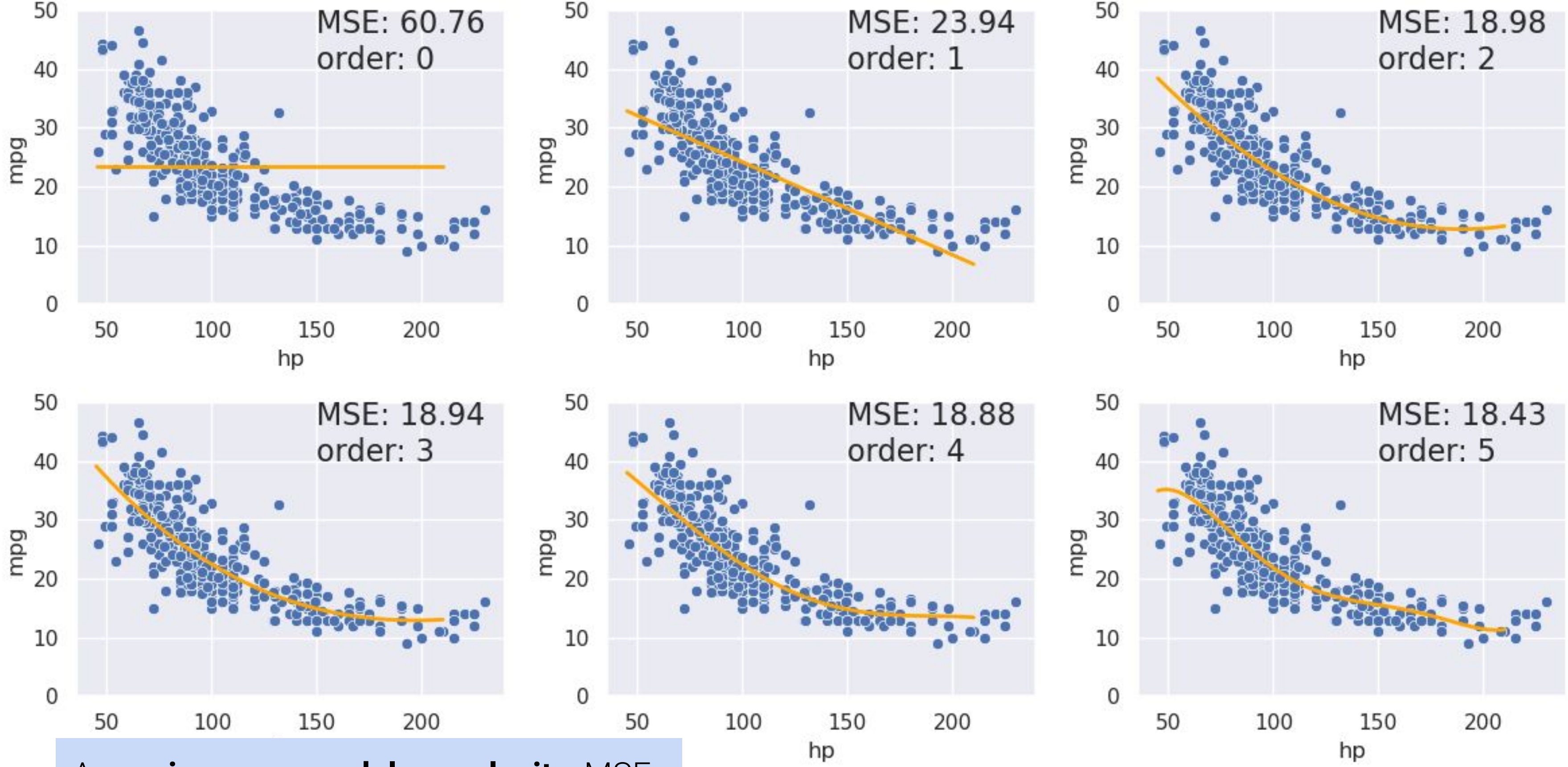
Which higher-order polynomial model do you think fits best?

ⓘ Start presenting to display the poll results on this slide.

Going Even Higher Order

$x = hp$ $\qquad \hat{y} = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_p x^p$

9148



As we **increase model complexity**, MSE drops from 60.76 to 23.94 to … 18.43.

Overfitting

Lecture 14, Data 100 Spring 2023

3819148

Four Parameter Model with Four Data Points

3819148

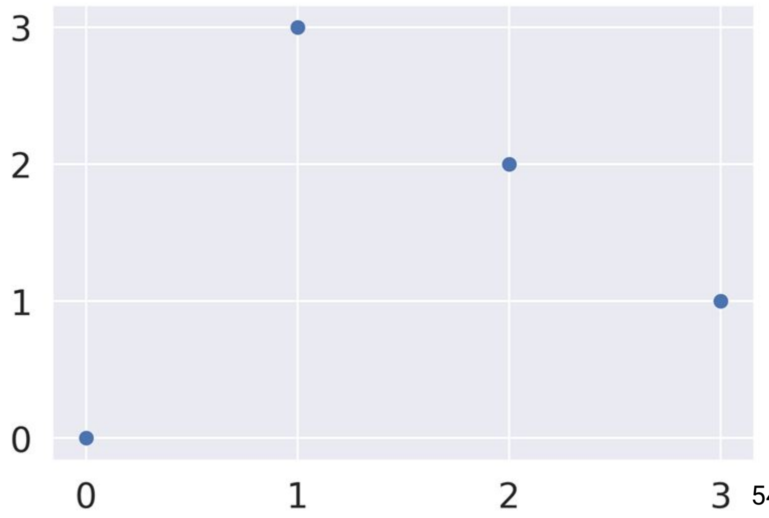
Algebra fact: Given **N** non-overlapping data points, we can always find a polynomial of degree **N-1** that goes through all those points.

Example: $x_1, y_1 = (0,0), x_2, y_2 = (1,3), x_3, y_3 = (2,2), x_4, y_4 = (3,1)$

There exist $\theta_0, \theta_1, \theta_2, \theta_3$ such that $\boxed{\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$ goes through all of these points.

Just solve the system of equations:

$$\begin{gathered}\theta_0 = 0 \\ \theta_0 + \theta_1 + \theta_2 + \theta_3 = 3 \\ \theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 = 2 \\ \theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 = 1\end{gathered}$$



Four Parameter Model with Four Data Points

3819148

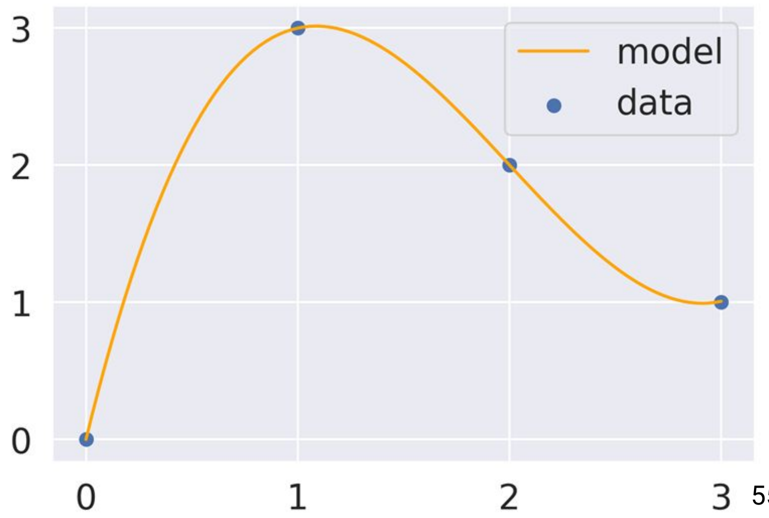
Algebra fact: Given N non-overlapping data points, we can always find a polynomial of degree N-1 that goes through all those points.

Example: $x_1, y_1 = (0,0), x_2, y_2 = (1,3), x_3, y_3 = (2,2), x_4, y_4 = (3,1)$

There exist $\theta_0, \theta_1, \theta_2, \theta_3$ such that $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ goes through all of these points, meaning **MSE = 0**.

Just solve the system of equations:

$$\begin{aligned} \theta_0 &= 0 \\ \theta_0 + \theta_1 + \theta_2 + \theta_3 &= 3 \\ \theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 &= 2 \\ \theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 &= 1 \end{aligned} \quad \Rightarrow \quad \begin{aligned} \theta_0 &= 0 \\ \theta_1 &= 19/3 \\ \theta_2 &= -4 \\ \theta_3 &= 2/3 \end{aligned}$$



Reminder: Solving a System of Linear Equations is Equivalent to Matrix Inversion

Solving our linear equations is equivalent to a matrix inversion.

$$\hat{y} = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \theta_3 x^3$$

$$x_1, y_1 = (0, 0), x_2, y_2 = (1, 3),$$
$$x_3, y_3 = (2, 2), x_4, y_4 = (3, 1)$$

$$\Leftrightarrow$$

$$\begin{aligned} \theta_0 &= 0 \\ \theta_0 + \theta_1 + \theta_2 + \theta_3 &= 3 \\ \theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 &= 2 \\ \theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 &= 1 \end{aligned}$$

Specifically, we're solving $\hat{Y} = \Phi\theta$,where $\hat{Y}$ is predictions, Φ s features, and θ is parameters.

$$\begin{bmatrix} 0 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

In sklearn

3819148

Can also do this in sklearn:

```
model = Pipeline([
    ('josh_transform', PolynomialFeatures(degree = 3, include_bias = False)),
    ('josh_regression', LinearRegression())
])
model.fit(arbitrary_data[["x"]], arbitrary_data["y"])
```

```
model.named_steps["josh_regression"].coef_
```

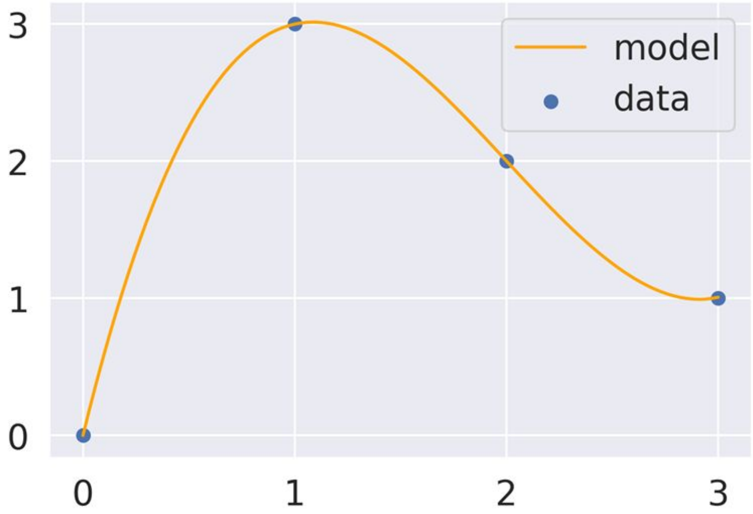
```
array([ 6.33333333, -4.        ,  0.66666667])
```

θ_1 θ_2 θ_3

```
model.named_steps["josh_regression"].intercept_
```

```
5.329070518200751e-15
```

θ_0



The Danger of Overfitting

3819148

This principle generalizes. If we have **100 data points** with only **a single feature**, we can always generate 99 polynomial features from the original feature, then fit a **100 parameter model** Φ that perfectly fits our data.

- MSE is always zero.
- Model is totally useless!!!

The problem we're facing here is **overfitting**: Our model is effectively just memorizing existing data and cannot handle new situations at all.

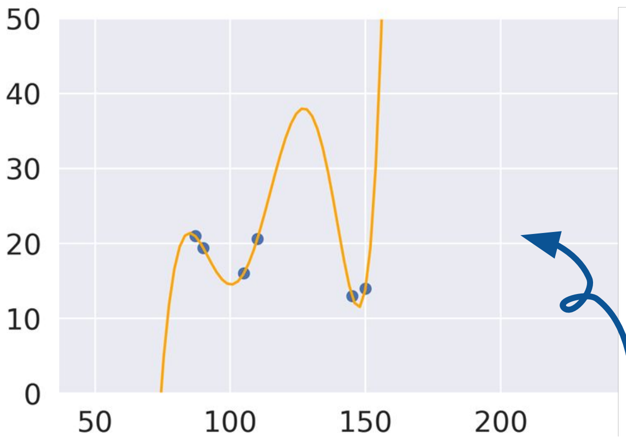
Model Sensitivity in Action

Let's build an **order-5** model that perfectly fits **6 randomly chosen vehicles** from our fuel efficiency dataset.

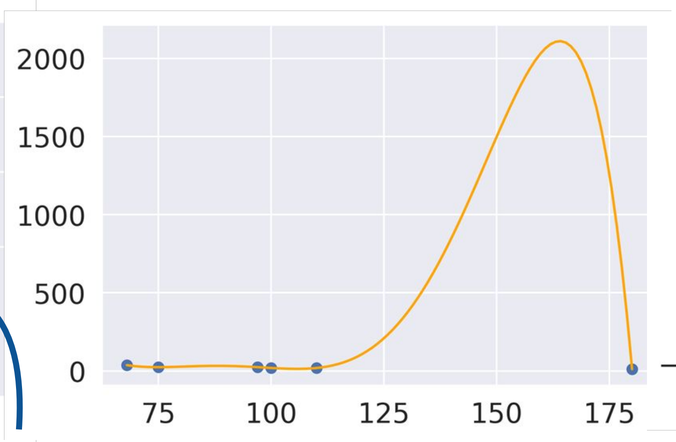
No matter which vehicles we pick, we'll almost always get an essentially* perfect fit.

(*With the caveat that real computers do not have infinite precision, and thus for even higher order models, this will break due to rounding errors.)

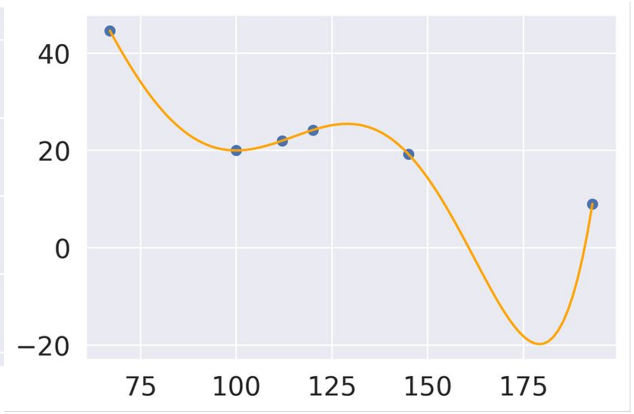
Order-5 model perfectly fit to Dataset 1



Order-5 model perfectly fit to Dataset 2



Order-5 model perfectly fit to Dataset 3



How well does one of these order-5 models **generalize** to the rest of the data?

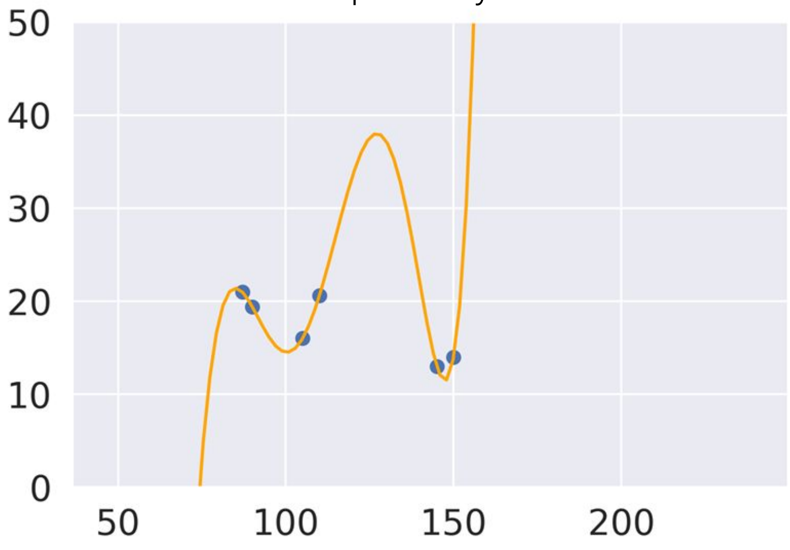
Comparing a Fit On Our Six Data Points with the Full Data Set

3819148

When overlaid on our full data set, we see that our predictions are terrible.

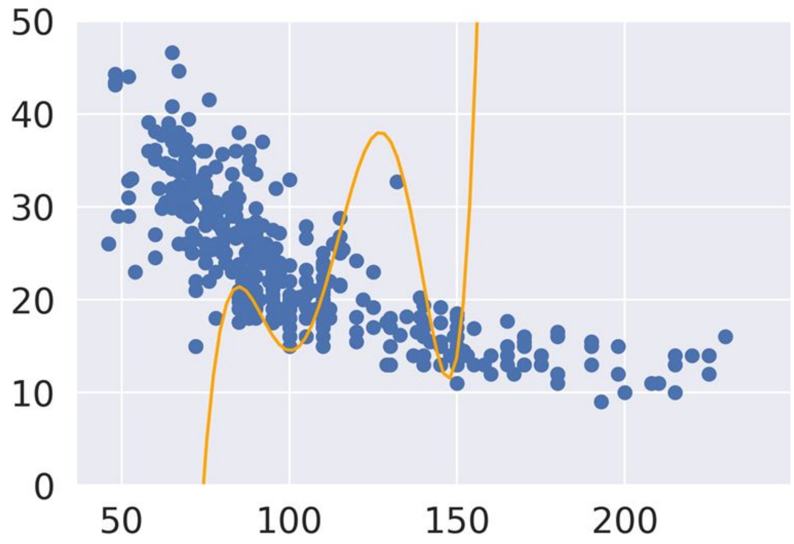
- Zero error on the **training set** (i.e. the set of data we used to train our model, Dataset 1).
- … but enormous error on a bigger sample of **real world data**.
- Since most data that we work with are just samples of some larger population, this is bad!

Order-5 model perfectly fit to **Dataset 1**



…

Model fit is terrible compared to **full dataset**



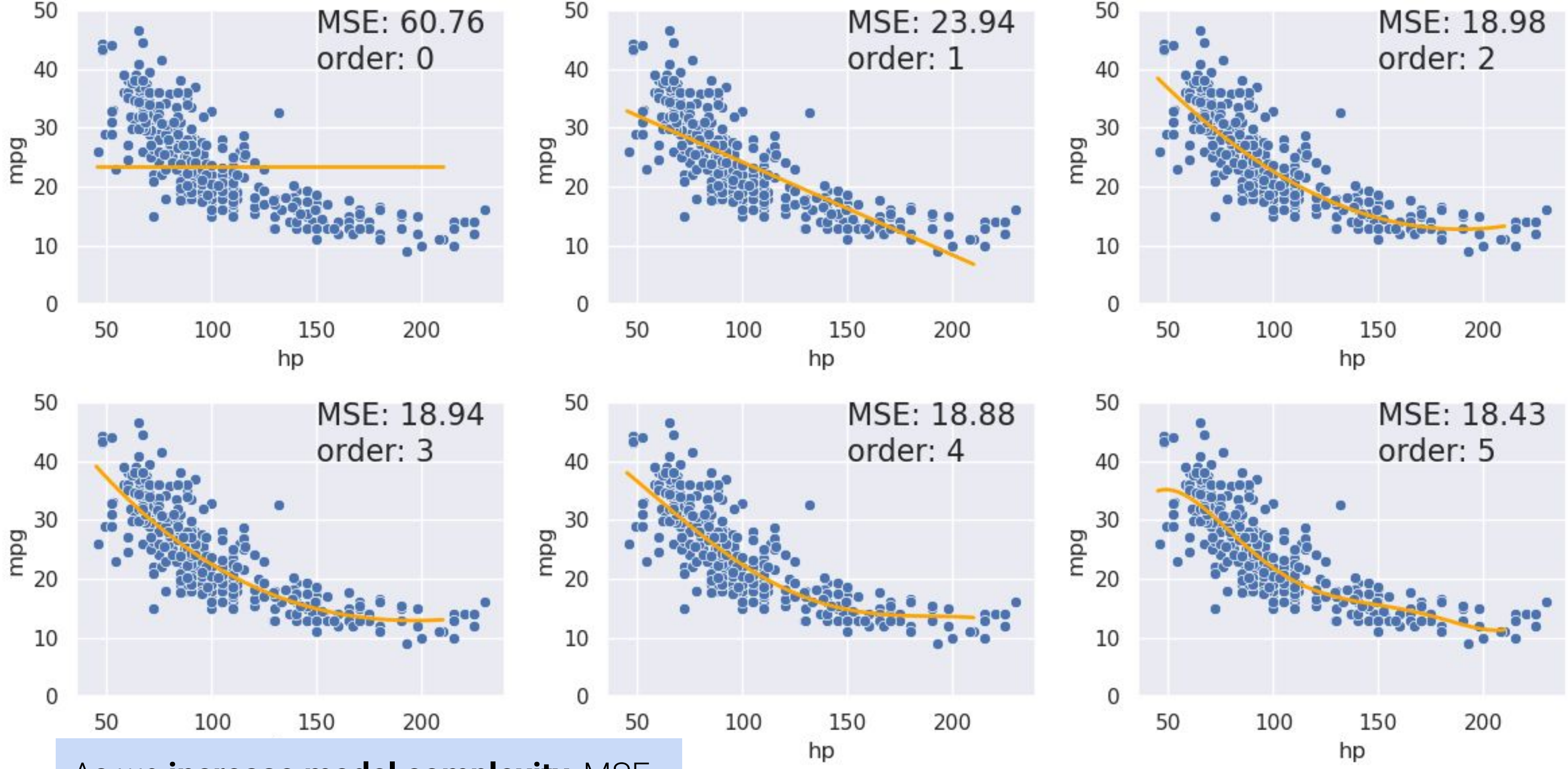
3819148

Variance and Training Error

Lecture 14, Data 100 Spring 2023

Going Even Higher Order

9148



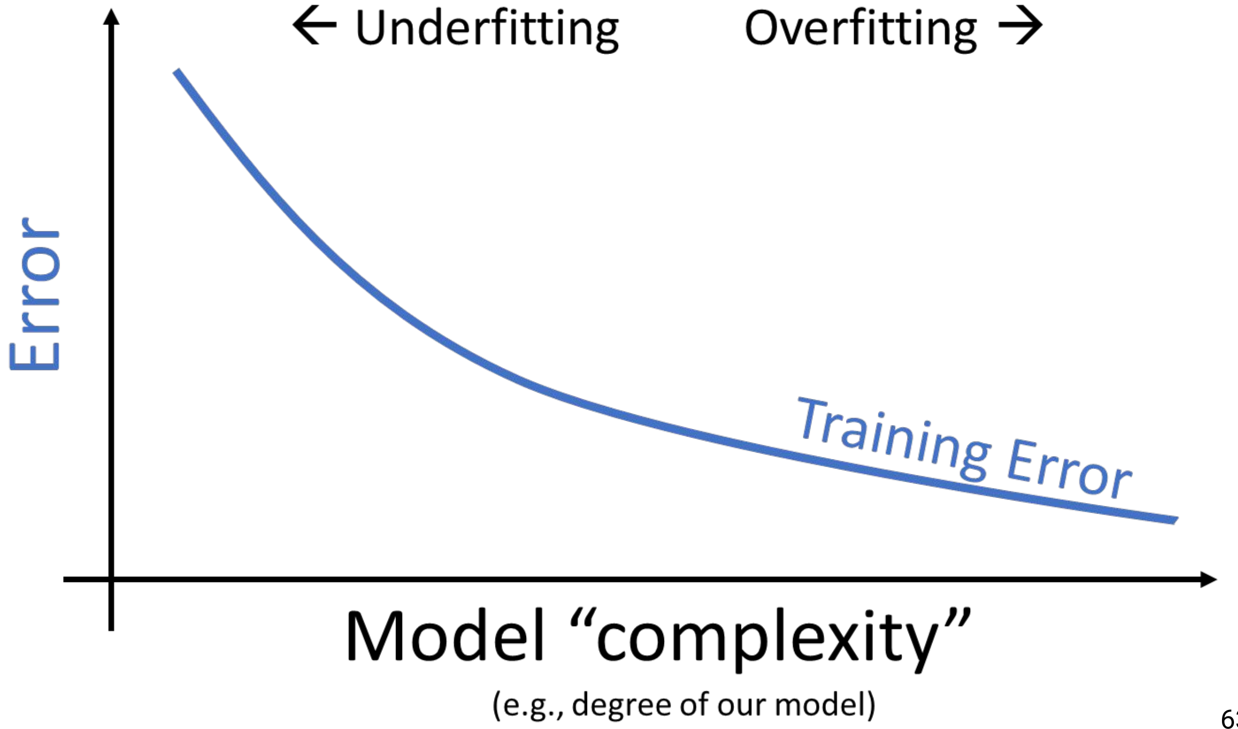
As we **increase model complexity,** MSE drops from 60.76 to 23.94 to … 18.43.

Error vs. Complexity

3819148

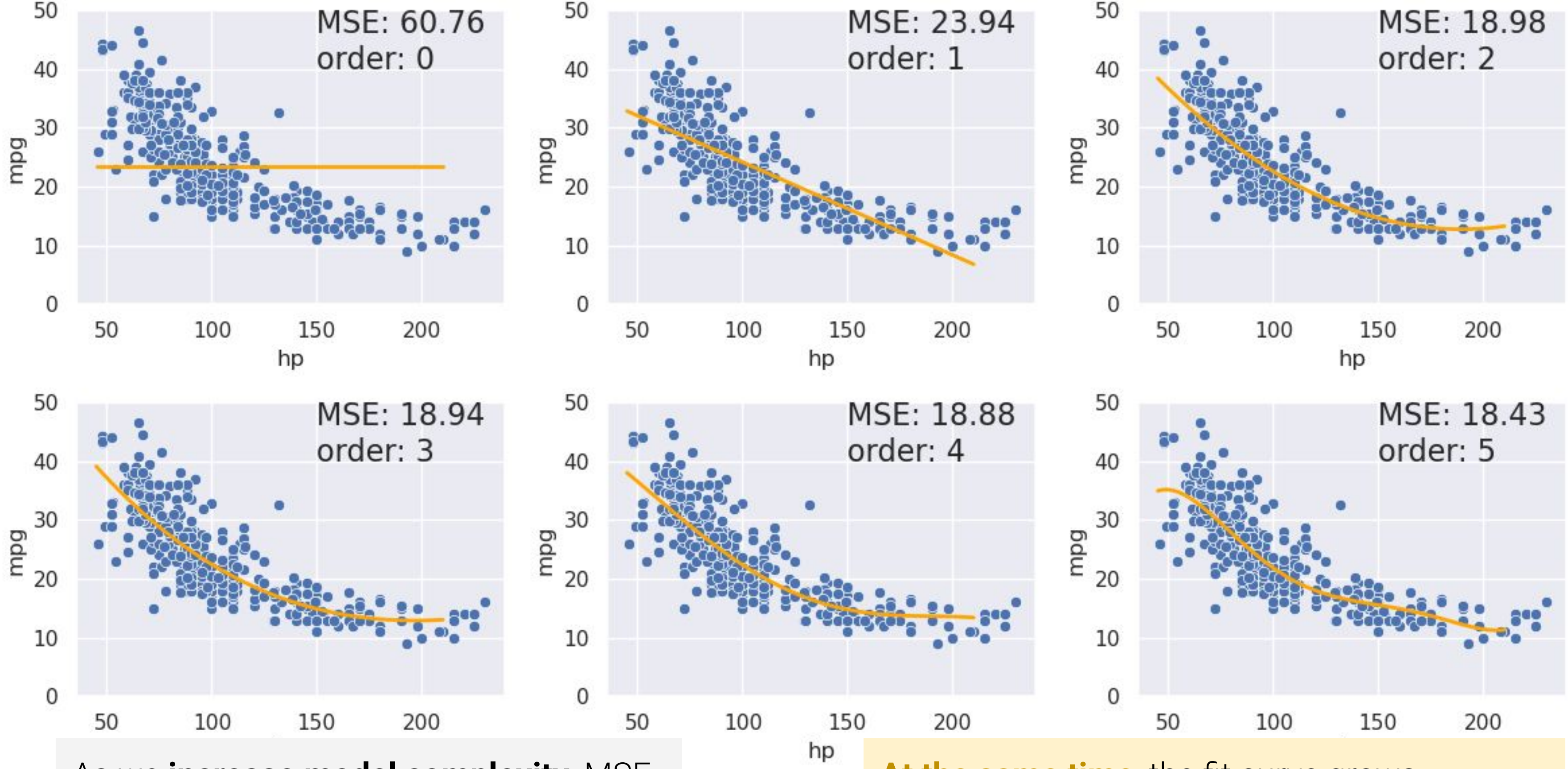
As we increase the complexity of our model:

- We see that the error on our training data (also called the **Training Error**) decreases.



Going Even Higher Order

9148



As we **increase model complexity**, MSE drops from 60.76 to 23.94 to … 18.43.

At the same time, the fit curve grows increasingly erratic and **sensitive** to the data.

Example on a Subset of the Data

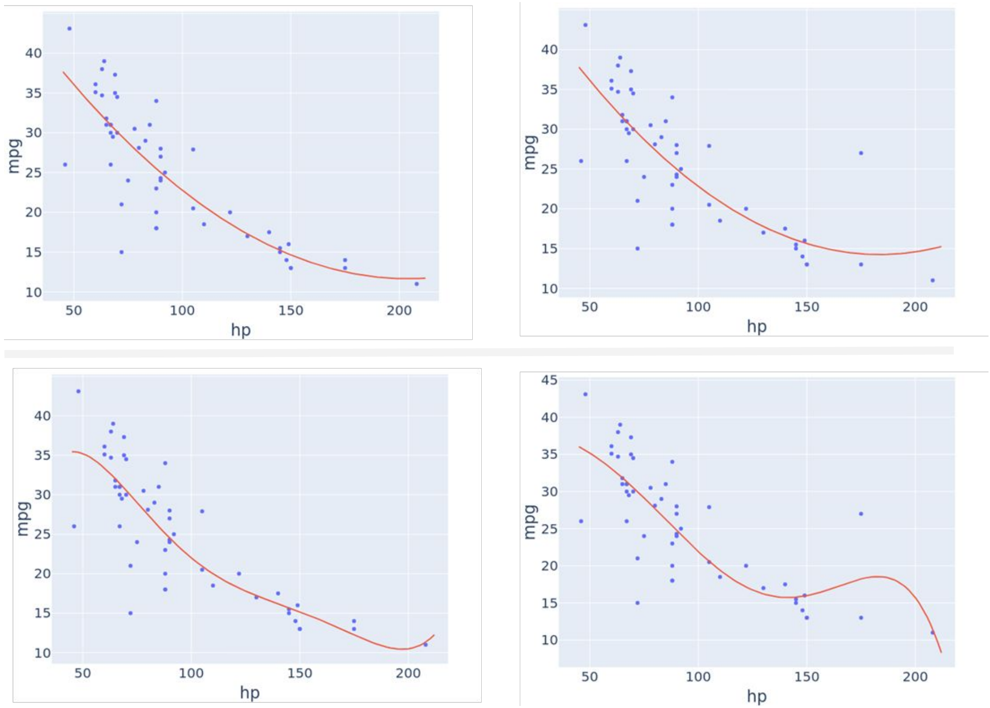
3819148

On top, we see the results of fitting two very similar datasets using an **order 2** model ($\theta_1 + \theta_2\, x + \theta_3\, x^2$). The resulting fit (model parameters) is close.

On bottom, we see the results of fitting the same datasets using an **order 6** model ($\theta_1 + \ldots + \theta_7\, x^6$). We see *very different predictions*, especially for higher hp.

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

Dataset 1 Dataset 2

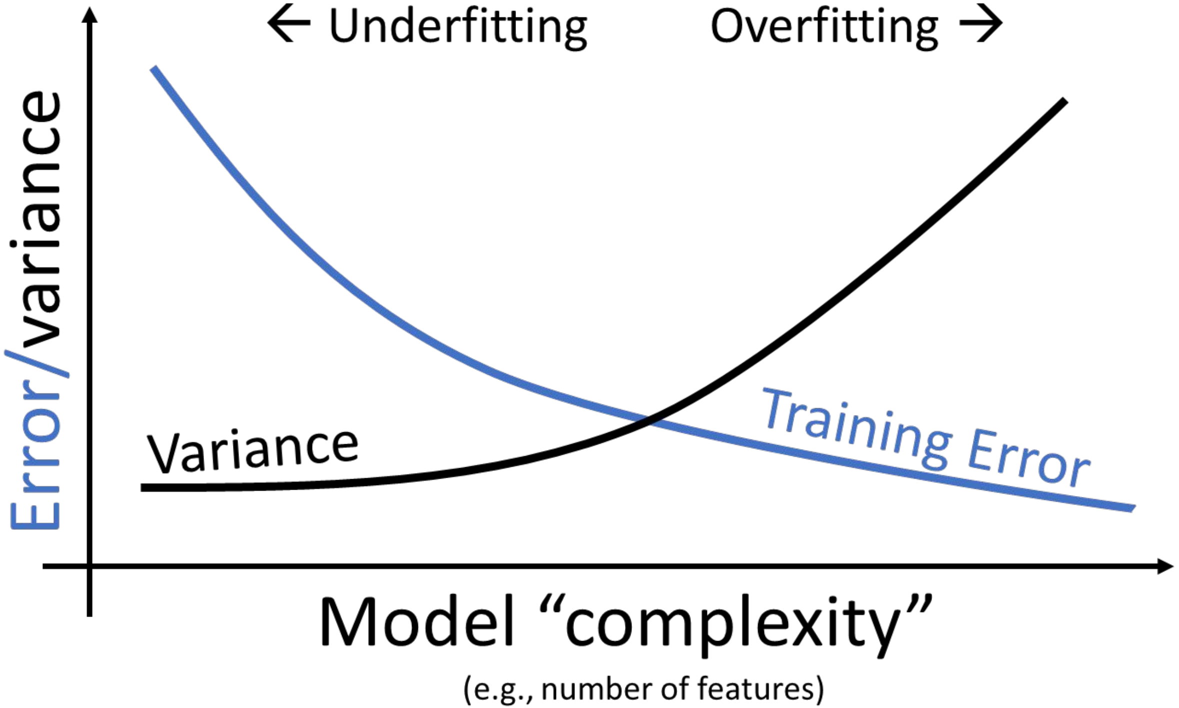


Error vs. Complexity

3819148

As we increase the complexity of our model:

- **Training error** decreases.
- **Variance** increases.



How do we find a model that hits the "sweet spot"?

Stay tuned for next time!!

Sp22 recording:
https://youtu.be/jIVrnaPZID8

3819148

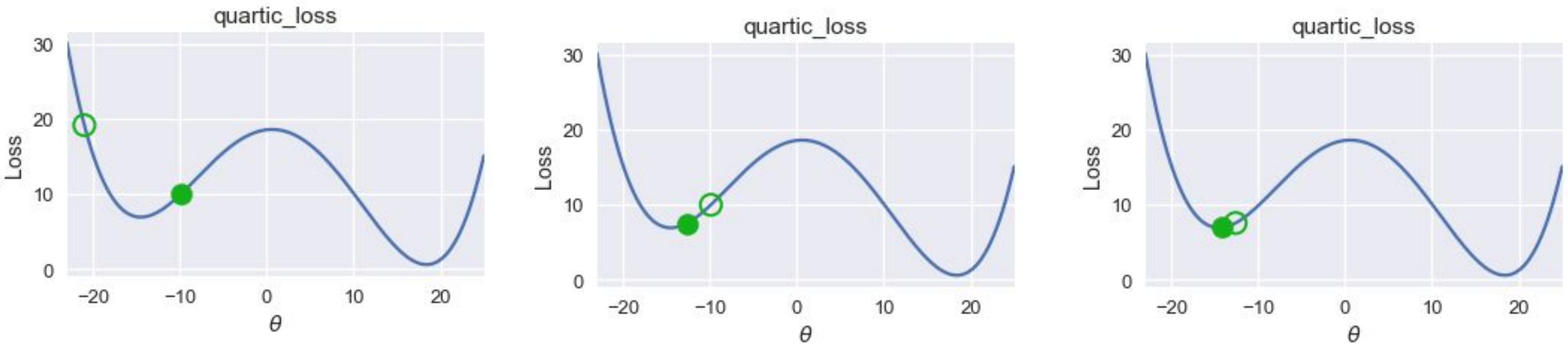
[Extra] Convexity

Lecture 14, Data 100 Spring 2023

Gradient Descent Only Finds Local Minima

3819148

As we saw, the gradient descent procedure can get stuck in a local minimum.



If a function has a special property called “convexity”, then gradient descent is guaranteed to find the global minimum.

Convexity

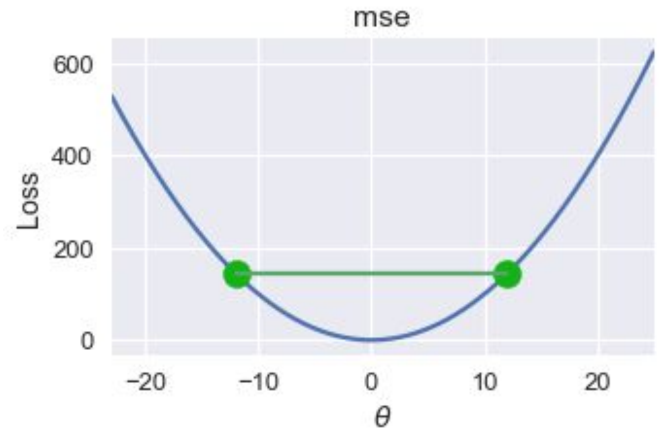
3819148

Formally, f is convex iff:

$$tf(a) + (1-t)f(b) \geq f(ta + (1-t)b)$$

$$\text{For all } a, b \text{ in domain of } f \text{ and } t \in [0,1]$$

- Or in plain English: If I draw a line between two points on the curve, all values on the curve must be on or below the line.
- Good news, MSE loss is convex (not proven)! So gradient descent is always going to do a good job minimizing the MSE, and will always find the global minimum.



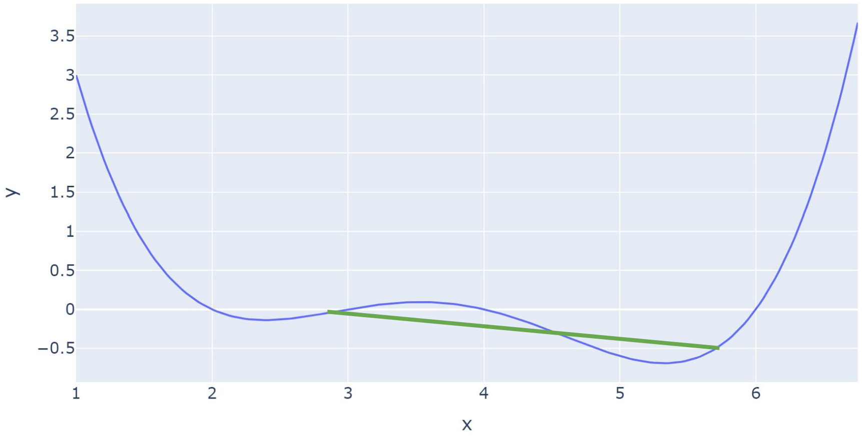
Convexity and Avoidance of Local Minima

3819148

For a **convex** function f, any local minimum is also a global minimum.

- If loss function convex, gradient descent will always find the globally optimal parameters.

Our arbitrary curve from before is not convex:

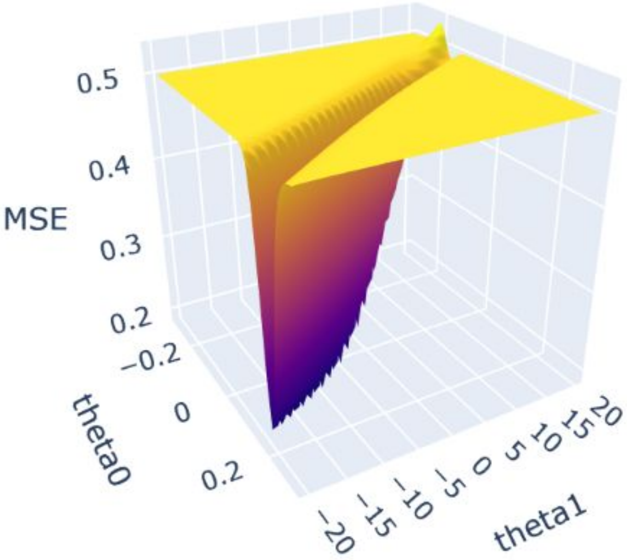


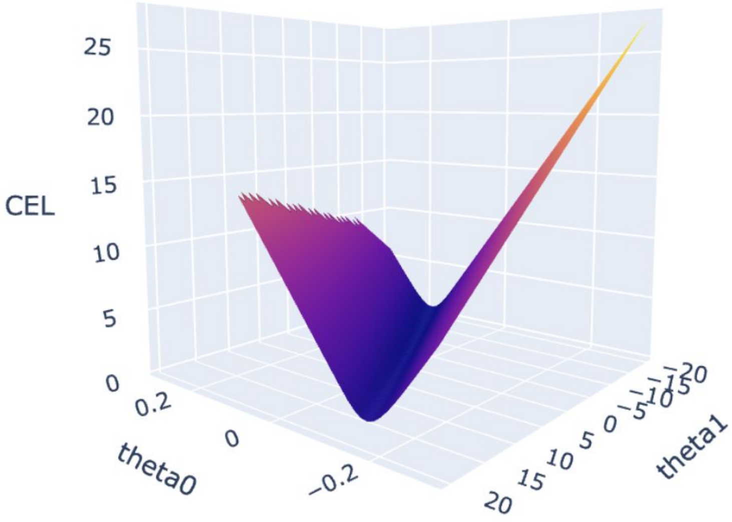
$$tf(a) + (1-t)f(b) \geq f(ta + (1-t)b)$$

For all a, b in domain of f and $t \in [0, 1]$

Not all point are below the line!

Convexity and Optimization Difficulty





Sp22 recording:
https://youtu.be/1UkINNEaA5A

3819148

[Extra] Detecting Overfitting

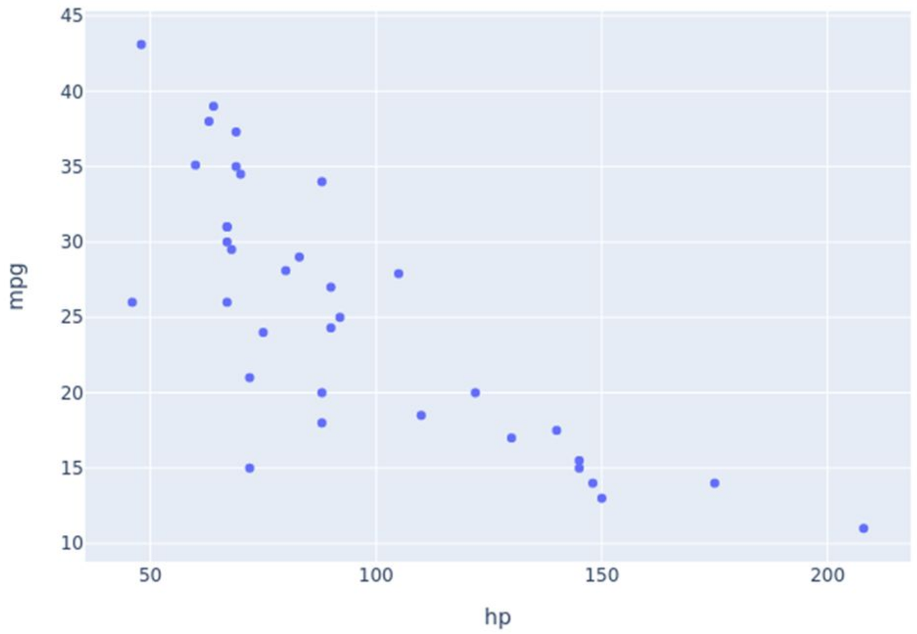
Lecture 14, Data 100 Spring 2023

Our 35 Samples

3819148

Consider a model fit on only the 35 data points.

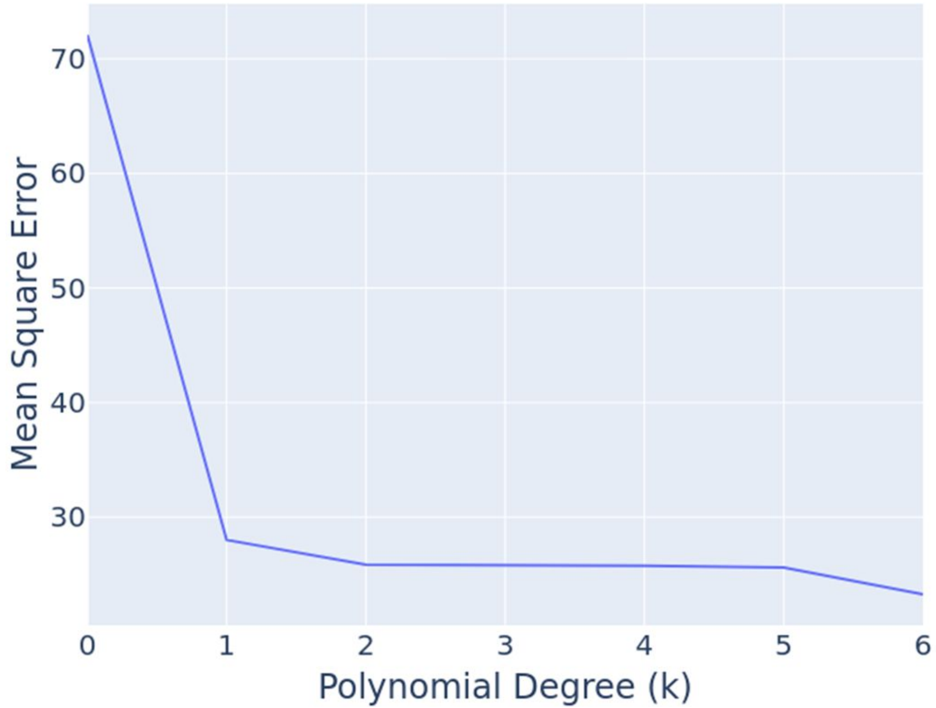
- We'll try various degrees and try to find the one we like best.



Fitting Various Degree Models

3819148

If we fit models of degree 0 through 7 of this model. The MSE is as shown below.

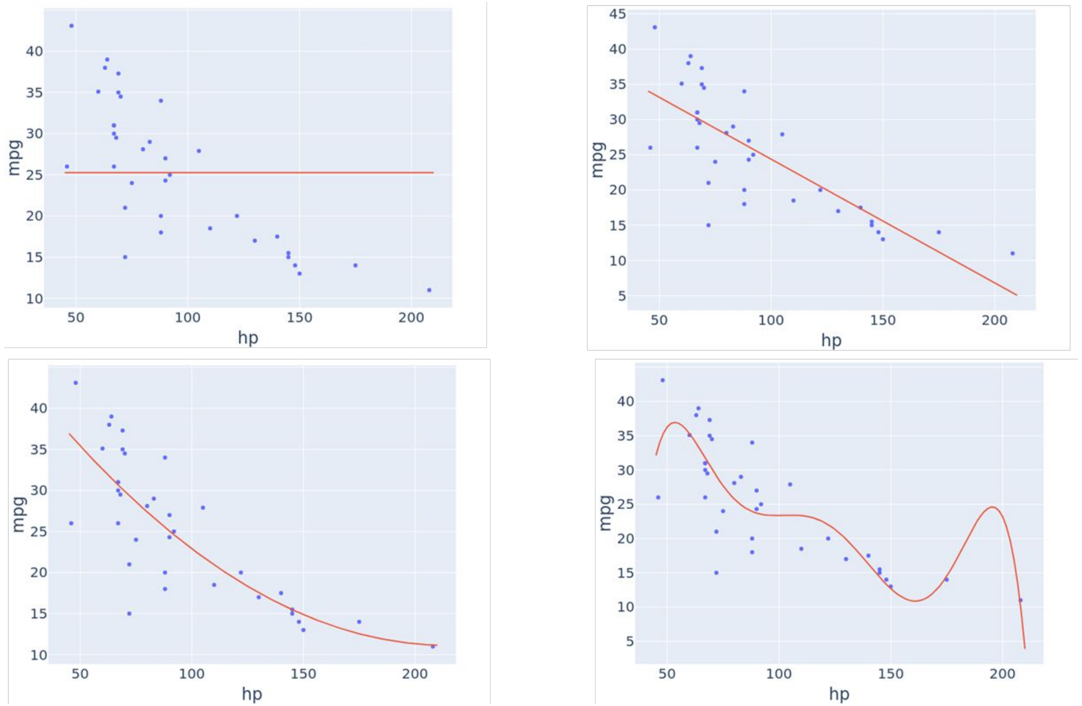


k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001

Visualizing the Models

3819148

Below we show the order 0, 1, 2, and 6 models.



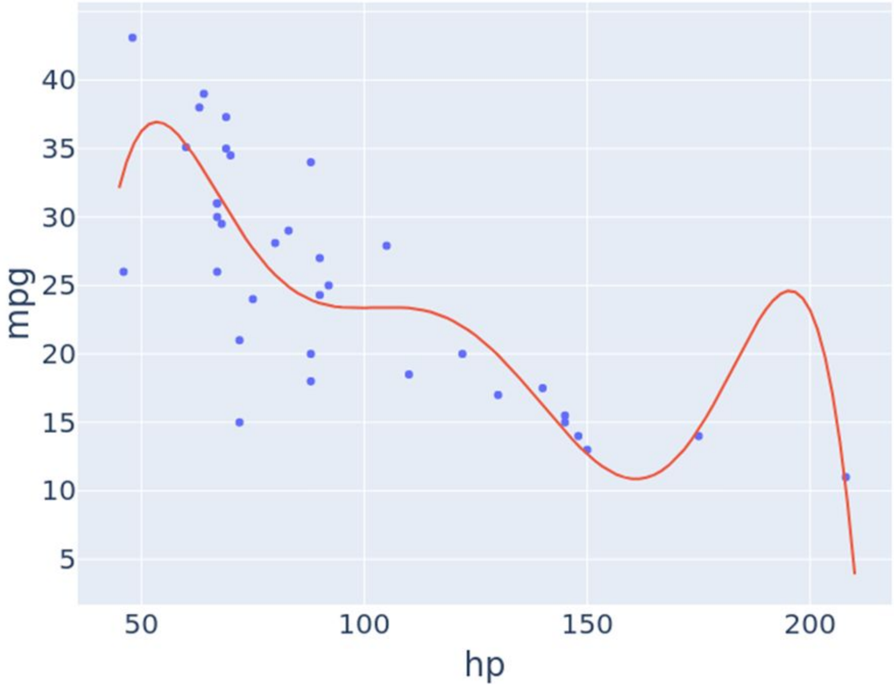
k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001

An Intuitively Overfit Model

3819148

Intuitively, the degree 6 model below feels like it is overfit.

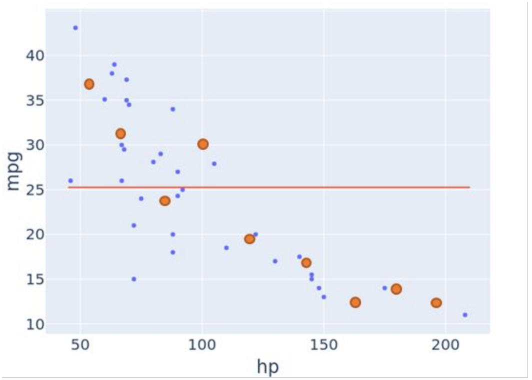
- More specifically: It seems that if we collect more data, i.e. draw more samples from the same distribution, we are worried this model will make poor predictions.

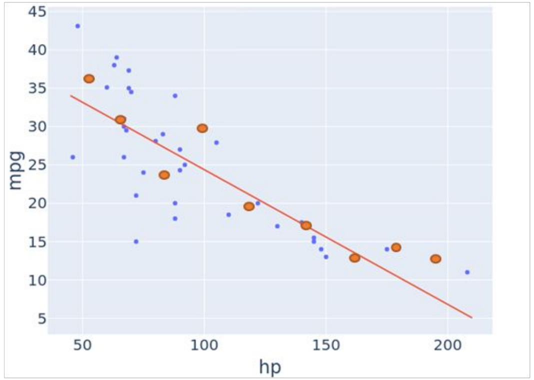


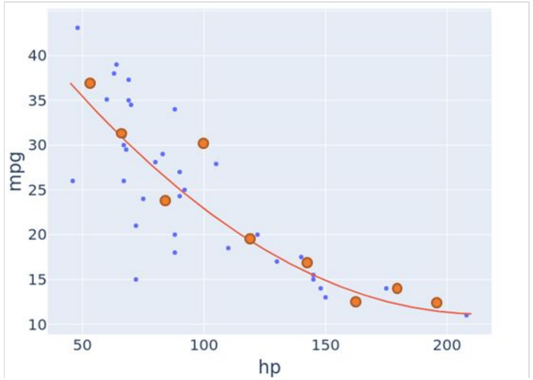
Collecting More Data to Prove a Model is Overfit

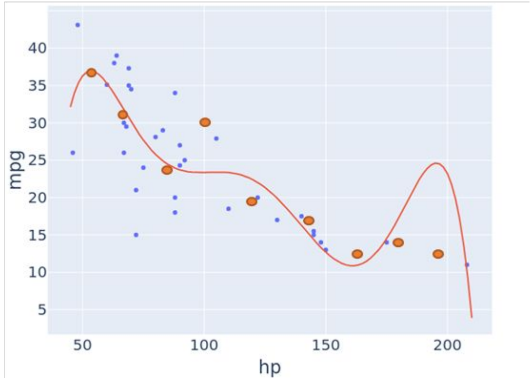
3819148

Suppose we collect the 9 new orange data points. Can compute MSE for our original models **without refitting using the new orange data points**.









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

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

Collecting More Data to Prove a Model is Overfit

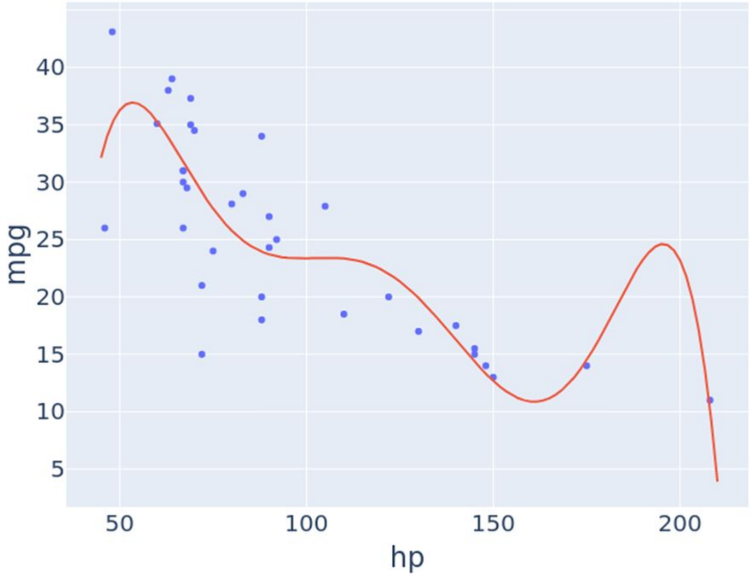
3819148

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.



LECTURE 14

Gradient Descent, Feature Engineering

Content credit: Acknowledgments