

Lecture 13

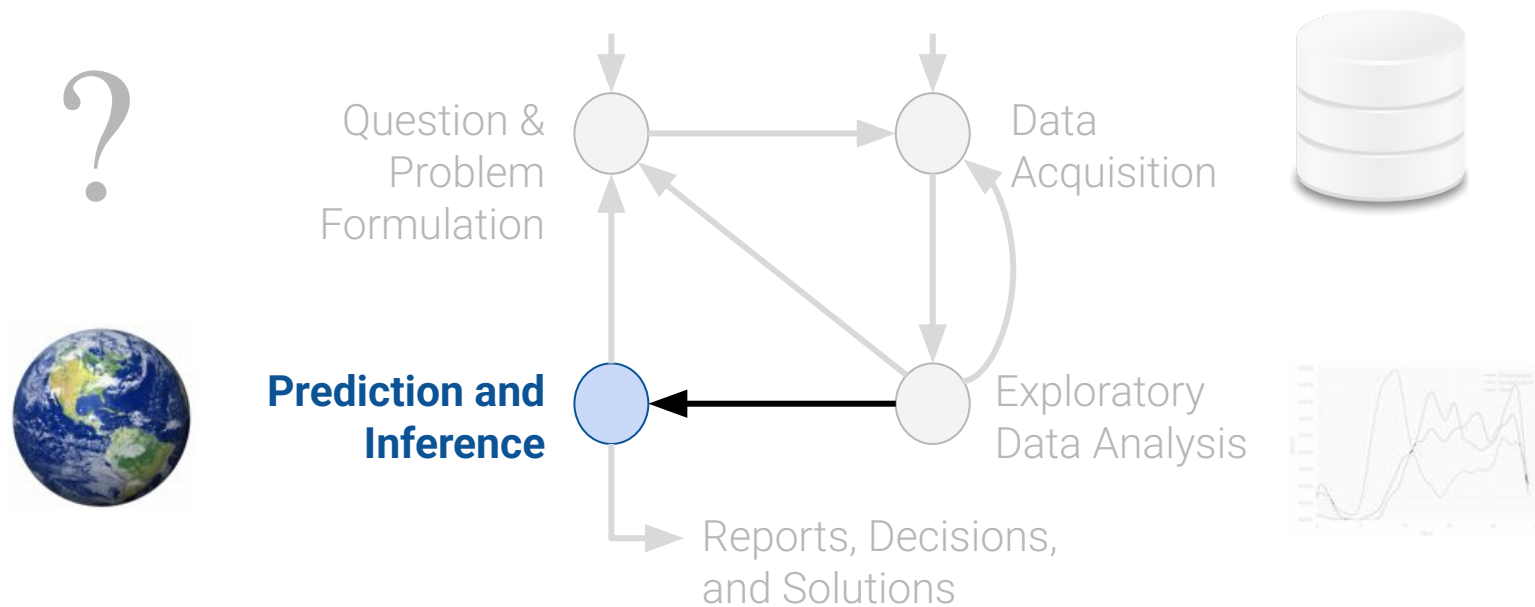
Gradient Descent, Feature Engineering

Finishing Optimization. Transforming Data to Improve Our Models.

Data 100/Data 200, Spring 2022 @ UC Berkeley

Josh Hug and Lisa Yan

Plan for Lectures 12 and 13: Model Implementation



(today)

Model Implementation I:

sklearn
Gradient Descent



Model Implementation II:

Gradient Descent
Feature Engineering

Today's Roadmap

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap Up:

- Stochastic Gradient Descent
- Convexity

Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

Stochastic Gradient Descent

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- **Stochastic Gradient Descent**
- Convexity

Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

Review: Gradient Descent

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

For a model with one parameter (equivalently: input data is one dimensional):

gradient of the loss function
evaluated at current θ

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

Next value for θ

θ : Model weights L : loss function

α : Learning rate (ours was constant, but other techniques have α decrease over time)

y : True values from training data

Review: Gradient Descent

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

For a model with multiple parameters:

gradient of the loss function
evaluated at current θ

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

Next value for θ

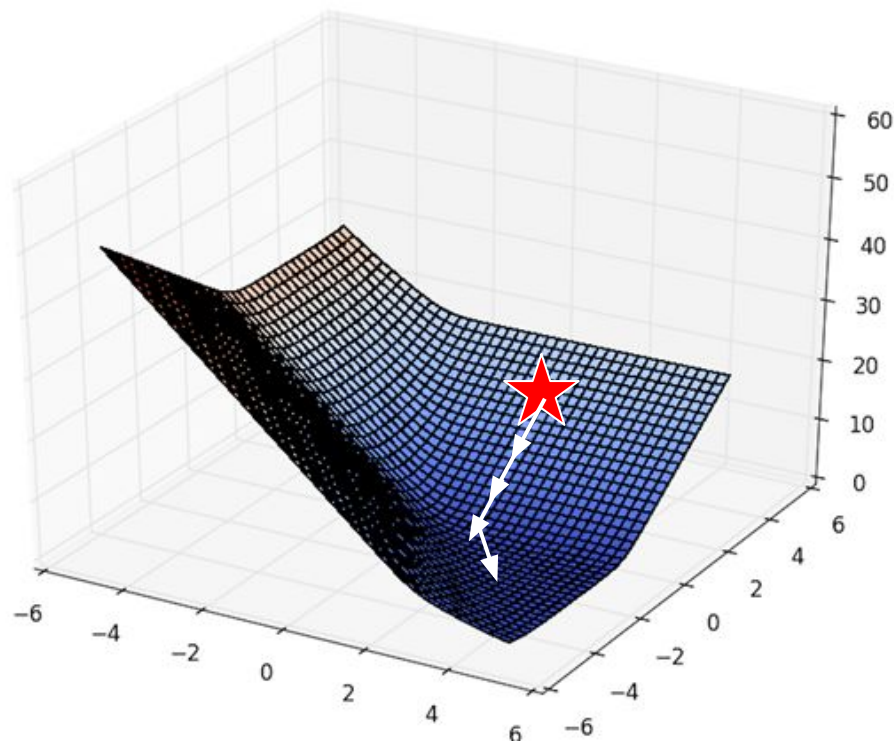
θ : Model weights L : loss function

α : Learning rate (ours was constant, but other techniques have α decrease over time)

y : True values from training data

Gradient Descent

By repeating this process over and over, you can find a local minimum of the function being optimized.



The algorithm we derived in the last class is more verbosely known as “batch gradient descent”.

- Uses the entire batch of available data to compute the gradient.

gradient of the loss function
evaluated at current θ

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

Next value for θ

Impractical in some circumstances. Imagine you have a billions of data points.

- Computing the gradient would require computing the loss for a prediction for EVERY data point, then computing the mean loss across all several billion.

Mini-Batch Gradient Descent

In mini-batch gradient descent, we only use a subset of the data when computing the gradient.

Example:

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

gradient of the loss function
evaluated at current θ

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

Next value for θ

Mini-Batch Gradient Descent

In mini-batch gradient descent, we only use a subset of the data when computing the gradient.

Example:

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

Question: Are we done now?

gradient of the loss function
evaluated at current θ

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

Next value for θ

Mini-Batch Gradient Descent

In mini-batch gradient descent, we only use a subset of the data when computing the gradient.

Example:

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

Question: Are we done now? **Not unless we were lucky!**

gradient of the loss function
evaluated at current θ

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

Next value for θ

Mini-Batch Gradient Descent

In mini-batch gradient descent, we only use a subset of the data when computing the gradient.

Example:

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

Question: So what should we do next?

gradient of the loss function
evaluated at current θ

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

Next value for θ

Mini-Batch Gradient Descent

In mini-batch gradient descent, we only use a subset of the data when computing the gradient.

Example:

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

Question: So what should we do next? **Go through data again.**

gradient of the loss function
evaluated at current θ

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

Next value for θ

Mini-Batch Gradient Descent

In mini-batch gradient descent, we only use a subset of the data when computing the gradient.
Example:

- Compute gradient on first 10% of the data. Adjust parameters. Then compute gradient on next 10% of the data ... Then compute gradient on final 10% of the data. Adjust parameters.
- Repeat the process above until we either hit some max number of iterations or our error is below some desired threshold.

gradient of the loss function
evaluated at current θ

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

Next value for θ

Mini-batch Gradient Descent

In mini-batch gradient descent, we only use a subset of the data when computing the gradient.
Example:

- **Compute gradient on first 10% of the data. Adjust parameters. Then compute gradient on next 10% of the data ... Then compute gradient on final 10% of the data. Adjust parameters.**
- Repeat the process above until we either hit some max number of iterations or our error is below some desired threshold.

Each pass in bold is called a **training epoch**.

gradient of the loss function
evaluated at current θ

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

Next value for θ

Of note: The gradient we compute using only 10% of our data is not the true gradient!

- It's merely an approximation. May not be the absolutely best way down the true loss surface.
- Works well in practice.

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

In our example, we used 10% as the size of each mini-batch.

- Interestingly, in real world practice, the size of a mini-batch is usually just a fixed number that is independent of the size of the data set.
- Size of the batch represents the quality of the gradient approximation.
 - All of the data (batch gradient descent): True gradient.
 - 10% of the data: Approximation of the gradient (may not be fastest way down)
- Typical choice for mini-batch size: 32 points.
 - Used regardless of how many billions of data points you may have.
 - See ML literature for more on why.
 - Results in a total of $N/32$ passes per epoch.

Additionally, rather than going in the order of our original dataset, we typically shuffle the data in between training epochs.

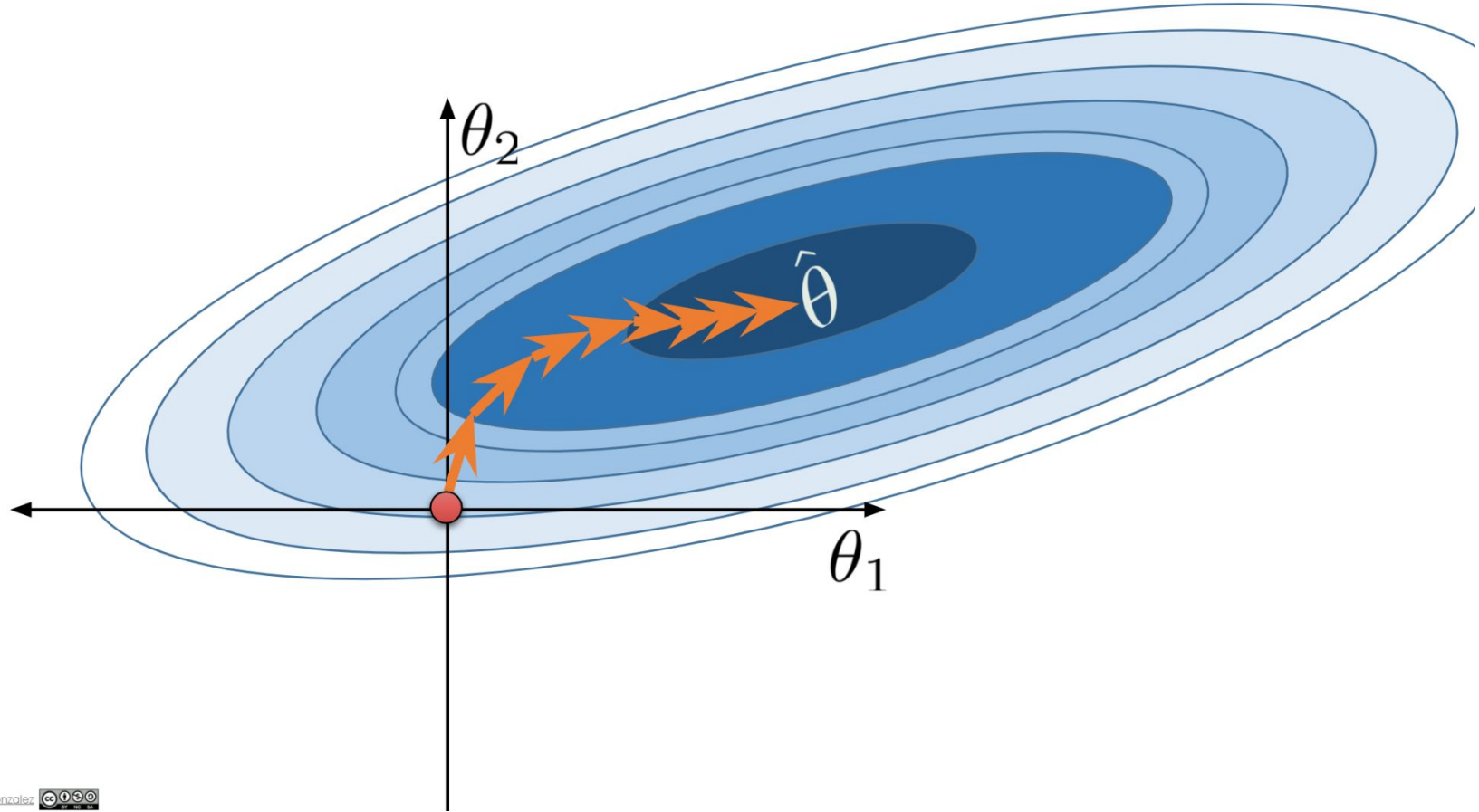
- Details are beyond the scope of our class.

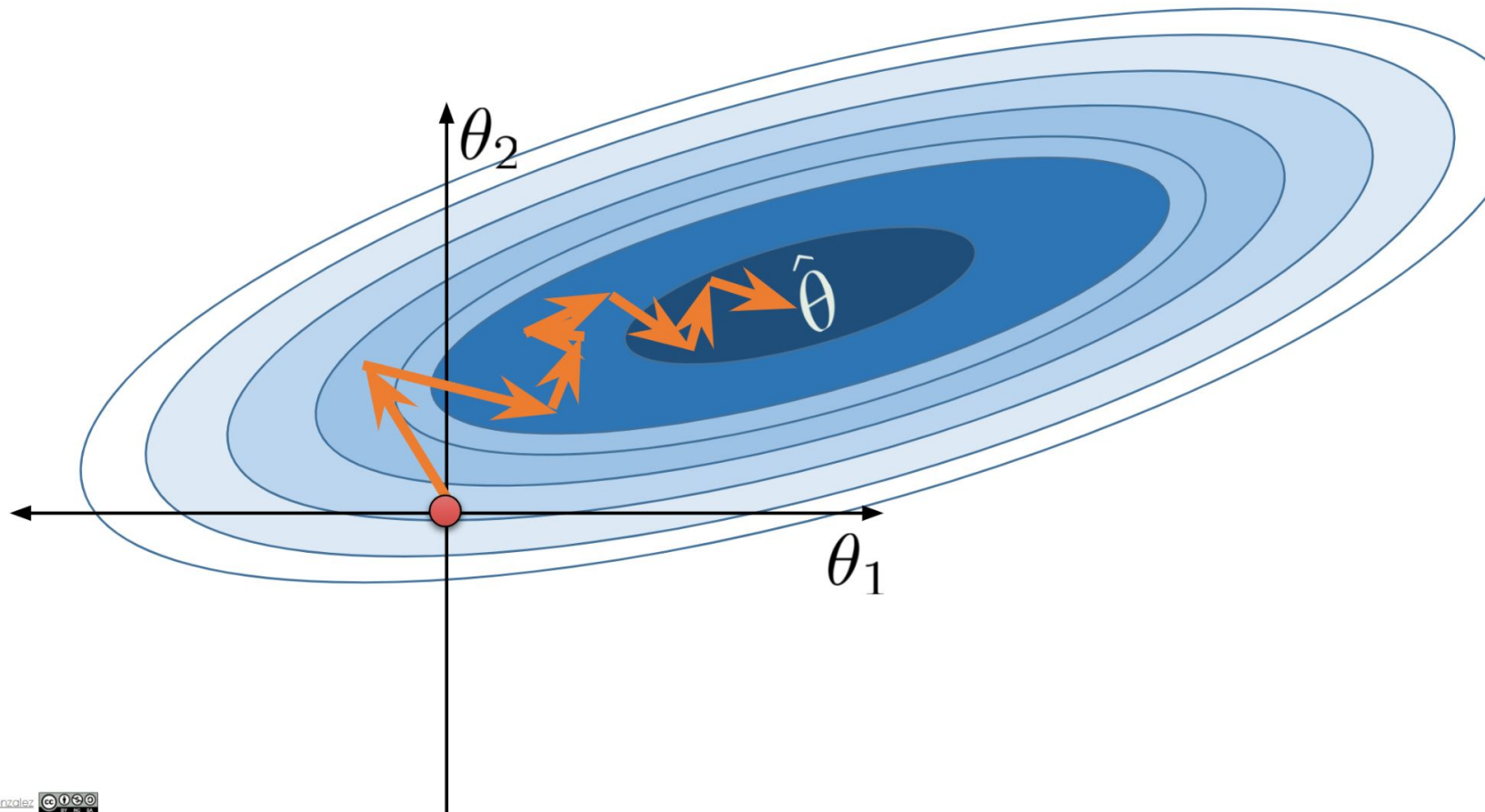
In the most extreme case, we choose a batch size of 1.

- Gradient is computed using only a single data point!
- Number of passes per epoch is therefore the number of data points.
- It may surprise you but this actually works on real world datasets.
 - Why it's surprising: 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.
 - Why it works (intuitively): If we average across many epochs across the entire dataset, effect is similar to if we simply compute the true gradient based on the entire dataset.

A batch size of 1 is called “stochastic gradient descent”.

- Some practitioners use the terms “stochastic gradient descent” and “mini-batch gradient descent” interchangeably.





Convexity

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

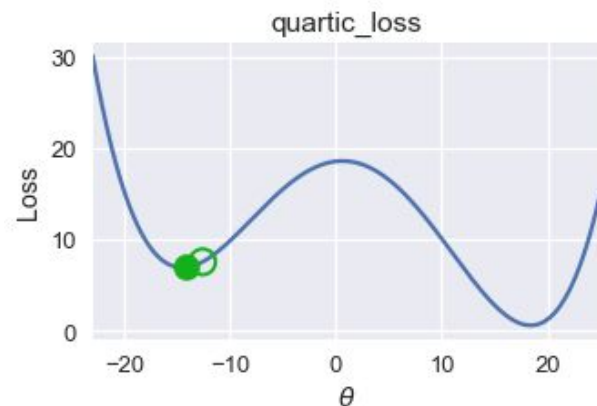
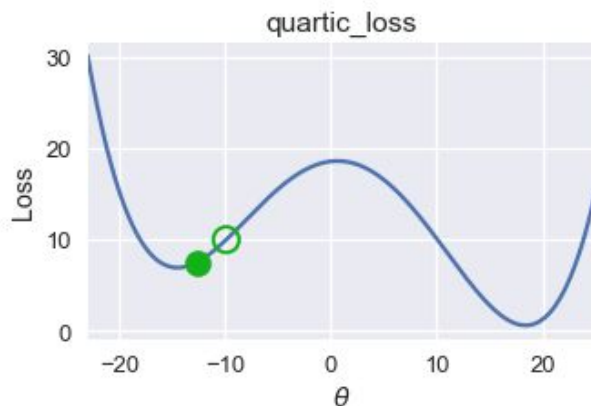
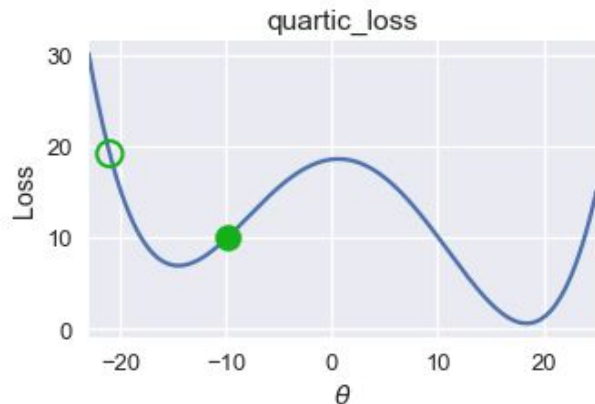
- Stochastic Gradient Descent
- **Convexity**

Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

Gradient Descent Only Finds Local Minima

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



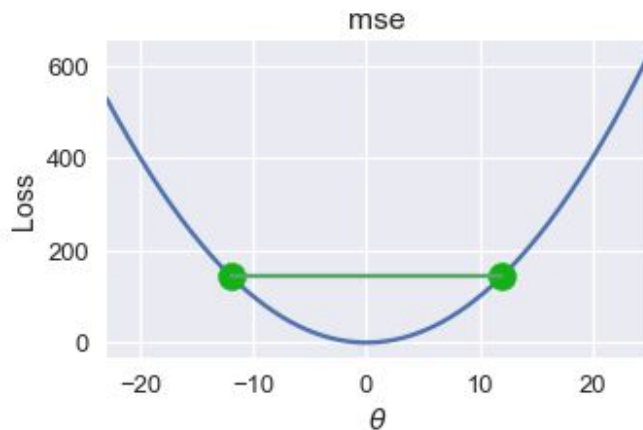
If a function has a special property called “convexity”, then gradient descent is guaranteed to find the global minimum.

Formally, f is convex iff:

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

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

- 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.

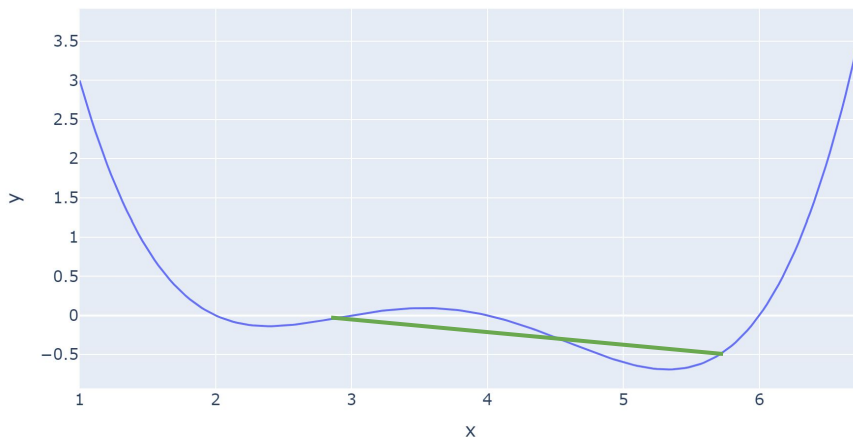


Convexity and Avoidance of Local Minima

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:

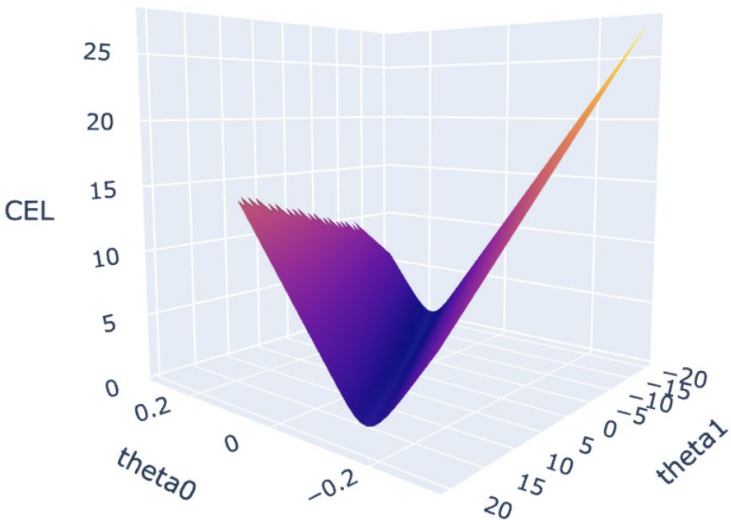
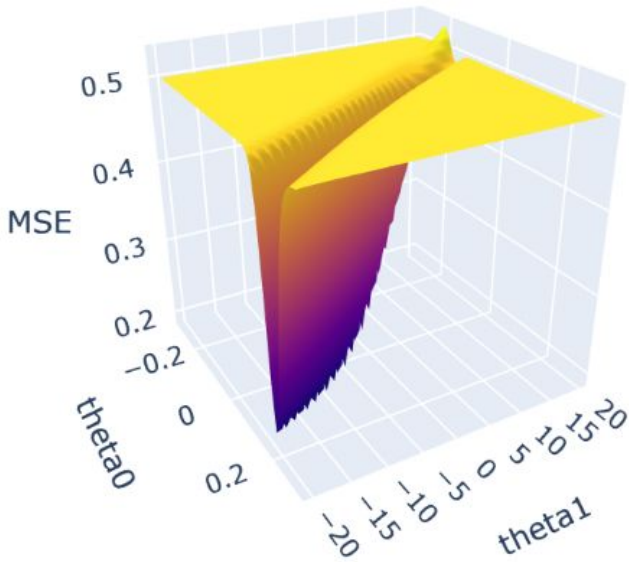


$$tf(a) + (1 - t)f(b) \geq f(ta + (1 - t)b)$$

Not all points are below the line!

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

Convexity and Optimization Difficulty



Fitting a Linear Parabolic Model

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

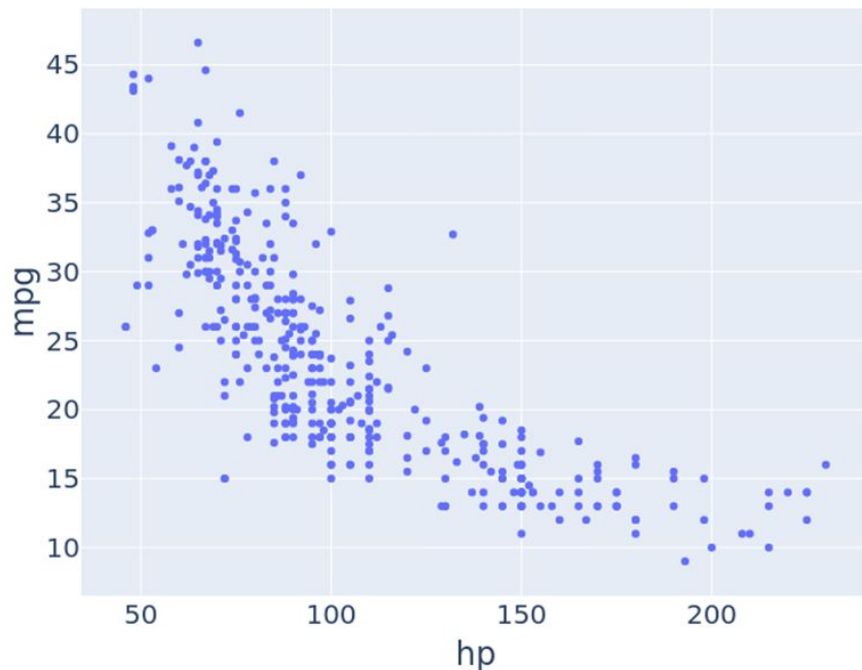
Feature Engineering:

- **Fitting a Linear Parabolic Model**
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

A Challenge for Linear Models

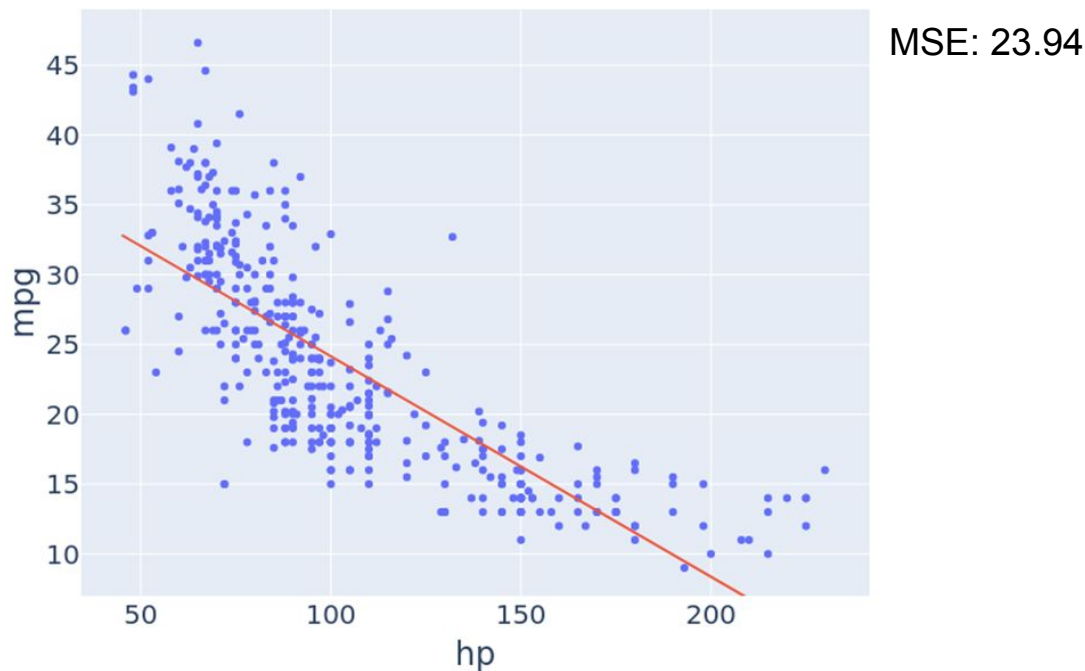
The plot below shows fuel efficiency vs. engine power of many different models of car.

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



Simple Linear Regression on MPG Data

If we create a simple linear regression model with hp as our only feature, we obviously can't capture the nonlinear relationship between mpg and hp.



Fitting a... Parabola?

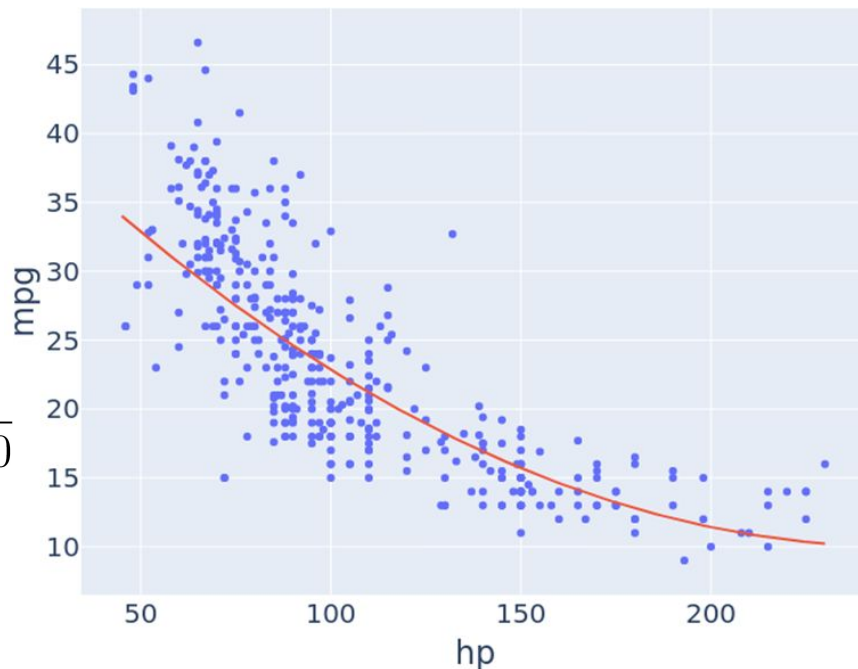
Just eyeballing this data, it seems that a quadratic model might do a better job on the range of data given.

- Bottom of the parabola somewhere around $x = 250, y = 10$.
- Should intersect somewhere near $x = 75, y = 27.5$.

$$y = \frac{(x - 250)^2}{1750} + 10$$

$$y = 45.7 - \frac{2}{7}x + \frac{x^2}{1750}$$

Same equation
written in two
different ways.



MSE: 21

With these two visual
observations we can
compute the equation for
the parabola.

Details not shown!

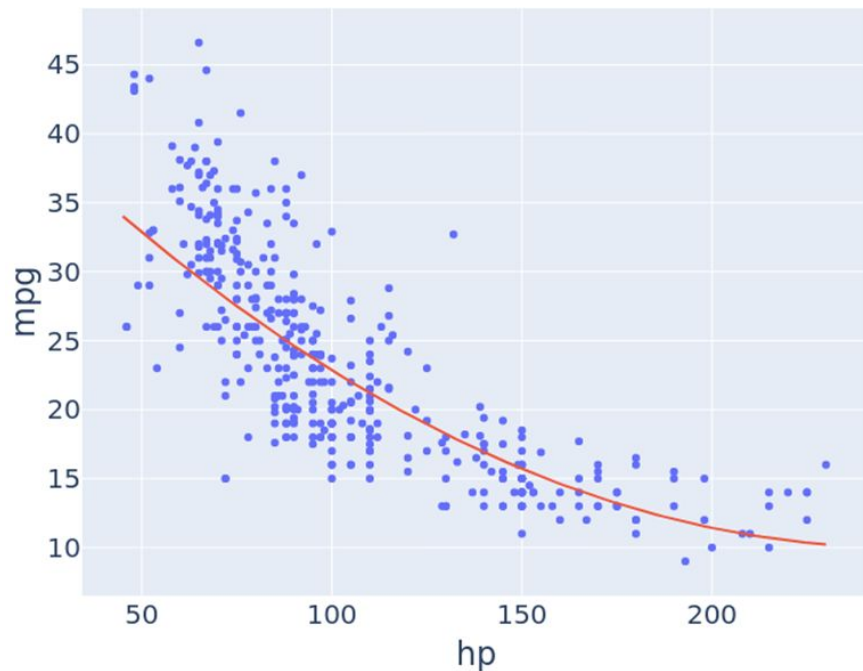
Our Model Is Nonlinear in X

Here, we observe that our model is of the form $y = \theta_0 + \theta_1 x + \theta_2 x^2$

- Our model appears to be nonlinear.
- In other words, doesn't seem to obey our definition of linear model:

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

$$y = 45.7 - \frac{2}{7}x + \frac{x^2}{1750}$$



The Wrong Approach

The wrong approach would be to entirely abandon our definition of a linear model and try to invent new fitting techniques and libraries for squared models.

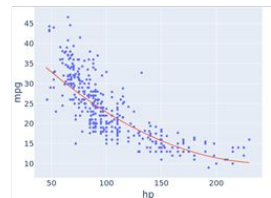
- Imaginary universe where “**SquaredRegression**” exists is depicted below! This **is not real**.
- Why is this bad? Lose all of the very nice linear algebra properties. There is a better way.

	hp	mpg
0	130.0	18.0
1	165.0	15.0
2	150.0	18.0
3	150.0	16.0
4	140.0	17.0
...
393	86.0	27.0
394	52.0	44.0
395	84.0	32.0
396	79.0	28.0
397	82.0	31.0

392 rows × 2 columns



```
from sklearn.nonlinear_model import SquaredRegression
model = SquaredRegression()
model.fit(vehicle_data[['hp']], vehicle_data['mpg'])
```



Staying Linear with Nonlinear Transformations

Rather than having to create an entirely new conceptual framework, a better solution is simply to add a new squared feature to our model.

- $x_1 = hp$
- $x_2 = hp^2$

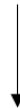
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_p x_p$$

If we do this, we can just use the same linear model framework from before!

	hp	hp2	mpg
0	130.0	16900.0	18.0
1	165.0	27225.0	15.0
2	150.0	22500.0	18.0
3	150.0	22500.0	16.0
4	140.0	19600.0	17.0
...
393	86.0	7396.0	27.0
394	52.0	2704.0	44.0
395	84.0	7056.0	32.0
396	79.0	6241.0	28.0
397	82.0	6724.0	31.0



```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(vehicle_data[['hp', 'hp2']], vehicle_data['mpg'])
```



Result on next slide

392 rows × 3 columns

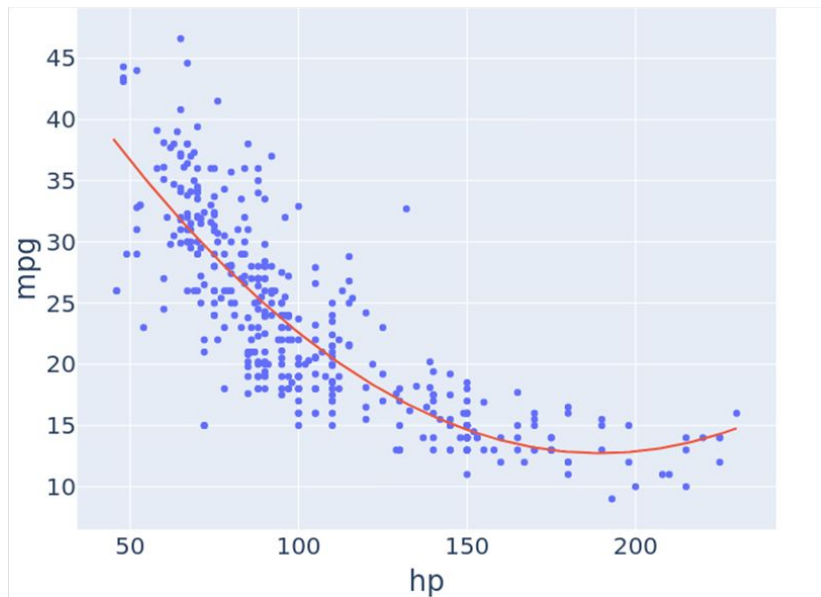
```
vehicle_data["hp2"] = vehicle_data["hp"]**2
```


Results of Linear Regression on Our Nonlinear Features

	hp	hp2	mpg
0	130.0	16900.0	18.0
1	165.0	27225.0	15.0
2	150.0	22500.0	18.0
3	150.0	22500.0	16.0
4	140.0	19600.0	17.0
...
393	86.0	7396.0	27.0
394	52.0	2704.0	44.0
395	84.0	7056.0	32.0
396	79.0	6241.0	28.0
397	82.0	6724.0	31.0

392 rows × 3 columns

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(vehicle_data[['hp', 'hp2']], vehicle_data['mpg'])
```



Comparing Our Models

My eyeballed parabolic model:

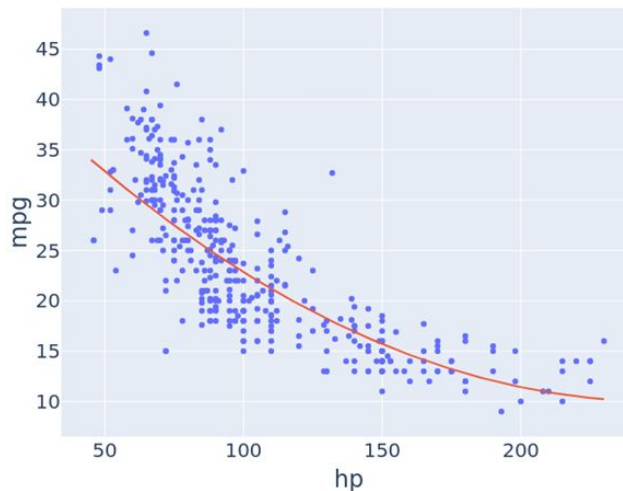
$$y = \frac{(x - 250)^2}{1750} + 10$$

$$y = 45.7 - \frac{2}{7}x + \frac{x^2}{1750}$$

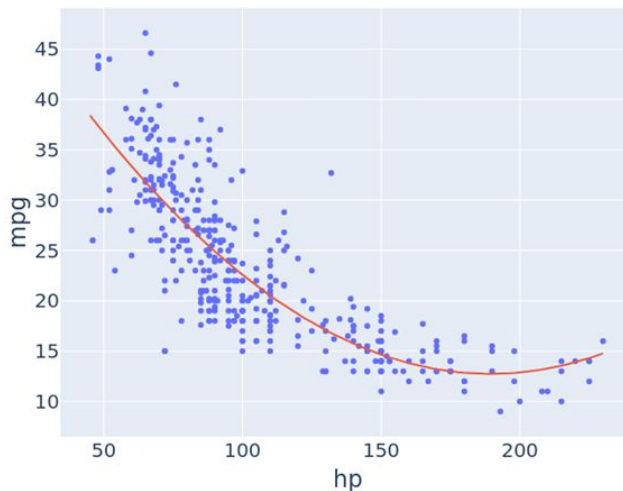
Our linear regression model using hp and hp².

$$y = \frac{(x - 189.424)^2}{812.68} + 12.746$$

$$y = 56.9 - 0.466x + \frac{x^2}{812.68}$$



MSE: 21



MSE: 18.98

Fitting a Linear Parabolic Model

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

Feature Engineering:

- Fitting a Linear Parabolic Model
- **Feature Engineering Overview**
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

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.

Feature Function

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

$$X \in \mathbb{R}^{n \times d} \longrightarrow \Phi \in \mathbb{R}^{n \times p}$$

Example: Our feature function earlier took our 1 dimensional input and transformed it into a 2 dimensional input.

p is often much greater than d.

	hp	mpg
0	130.0	18.0
1	165.0	15.0
2	150.0	18.0
3	150.0	16.0
4	140.0	17.0
...
393	86.0	27.0
394	52.0	44.0
395	84.0	32.0
396	79.0	28.0
397	82.0	31.0

392 rows × 2 columns

	hp	hp2	mpg
0	130.0	16900.0	18.0
1	165.0	27225.0	15.0
2	150.0	22500.0	18.0
3	150.0	22500.0	16.0
4	140.0	19600.0	17.0
...
393	86.0	7396.0	27.0
394	52.0	2704.0	44.0
395	84.0	7056.0	32.0
396	79.0	6241.0	28.0
397	82.0	6724.0	31.0

392 rows × 3 columns

Transformed Data and Linear Models

As we saw in our example earlier, adding a squared feature allowed us to capture a parabolic relationship.

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

Note that the equation for a linear model that is trained on transformed data is sometimes written using the symbol phi instead of x:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 \quad \longrightarrow \quad y = \theta_0 + \theta_1 \phi_1 + \theta_2 \phi_2$$

$$Y = X\theta \quad \longrightarrow \quad Y = \Phi\theta$$

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 of its ability to automatically learn feature functions. See a course in deep learning for more.

Let's see an example video where Professor Joey Gonzales takes a 2 dimensional input and transforms it into a 15 dimensional input, allowing him to fit a rather complex surface using a linear model.

$$X \in \mathbb{R}^{n \times d} \longrightarrow \Phi \in \mathbb{R}^{n \times p}$$

High Dimensional Feature Engineering Example (Joey Gonzales)

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- **High Dimensional Feature Engineering Example (Joey Gonzales)**
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

One Hot Encoding

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- **One Hot Encoding**
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

Regression Using Non-Numeric Features

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.

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

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


Using Non-Numeric Features: One Hot Encoding

One approach is to use what is known as a “one hot encoding.”

- Give every category its own feature, with value = 1 if that category applies to that row.
- Can do this using the `get_dummies` function.

	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

	Thur	Fri	Sat	Sun
193	1	0	0	0
90	0	1	0	0
25	0	0	1	0
26	0	0	1	0
190	0	0	0	1



```
dummies = pd.get_dummies(data['day'])
```

Using Non-Numeric Features: One Hot Encoding

One approach is to use what is known as a “one hot encoding.”

- Give every category its own feature, with value = 1 if that category applies to that row.
- Can do this using the `get_dummies` function. Then join with the original table with `pd.concat`.

	total_bill	size	day	Thur	Fri	Sat	Sun
193	15.48	2	Thur	1	0	0	0
90	28.97	2	Fri	0	1	0	0
25	17.81	4	Sat	0	0	1	0
26	13.37	2	Sat	0	0	1	0
190	15.69	2	Sun	0	0	0	1

```
data_w_dummies = pd.concat([three_feature_data, dummies], axis=1)
```

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

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

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

```
from sklearn.linear_model import LinearRegression
f_with_day = LinearRegression(fit_intercept=False)
f_with_day.fit(data_w_dummies[["total_bill", "size", "Thur",
                               "Fri", "Sat", "Sun"]], data["tip"])
```

Test Your Understanding

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

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

Resulting prediction is:

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

To test your understanding, what tip would the model predict for a party of 3 with a \$50 check eating on a Thursday?

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

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

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

To test your understanding, what tip would the model predict for a party of 3 with a \$50 check eating on a Thursday?

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

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

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

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

To test your understanding, what tip would the model predict for a party of 3 with a \$50 check eating on a Thursday?

```
f_with_day.predict([[50, 3, 1, 0, 0, 0]])
```


Interpreting the 6 Dimensional Model

It turns out the MSE for this 6 dimensional model is 1.01.

- A model trained on only the bill and the table size 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.

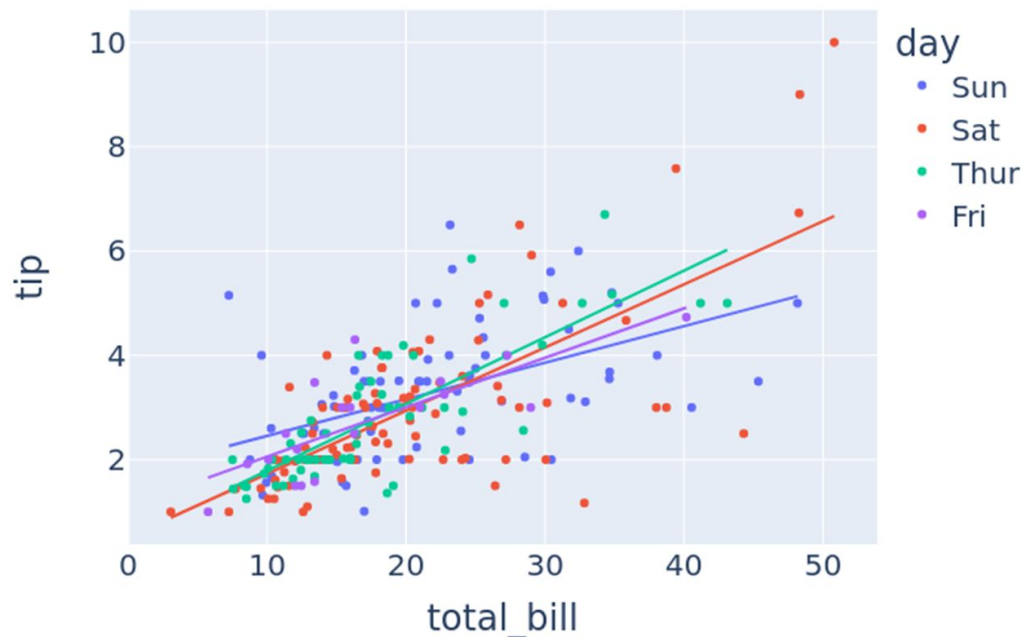
- Bizarre to imagine that humans have a base tip that they start with for every day of the week.
- My guess: This model will not generalize well to newly collected data.

An Alternate Approach

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")
```



High Order Polynomial Example

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- **High Order Polynomial Example**
- Variance and Training Error
- Overfitting
- Detecting Overfitting

Let's return to where we started today: Creating higher order features for the mpg dataset. An interesting question arises: 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:

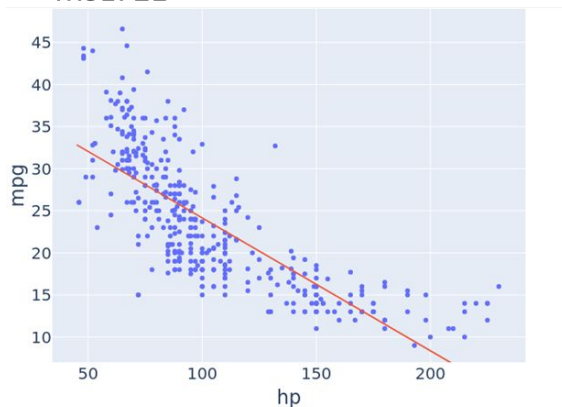
```
vehicle_data["hp3"] = vehicle_data["hp"]**3
```

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

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

Cubic Fit Results

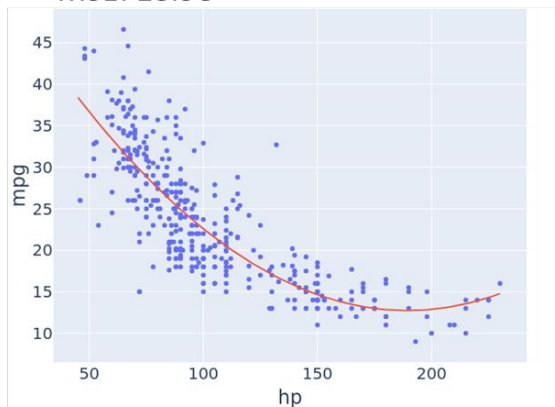
MSE: 21



Degree 1 Features

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

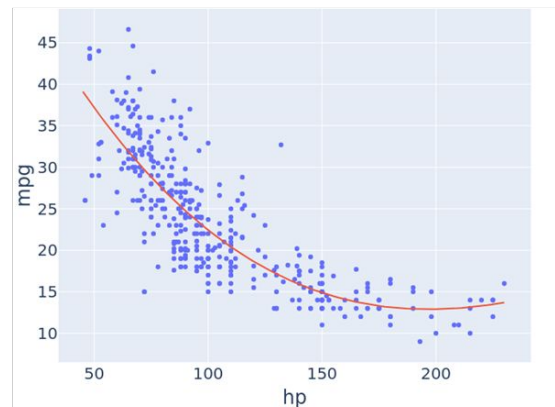
MSE: 18.98



Degree 2 Features

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

MSE: 18.94



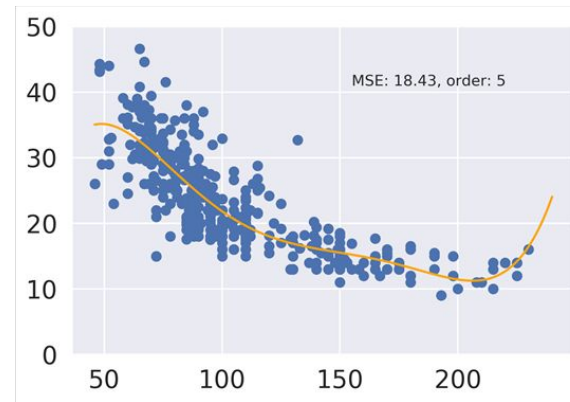
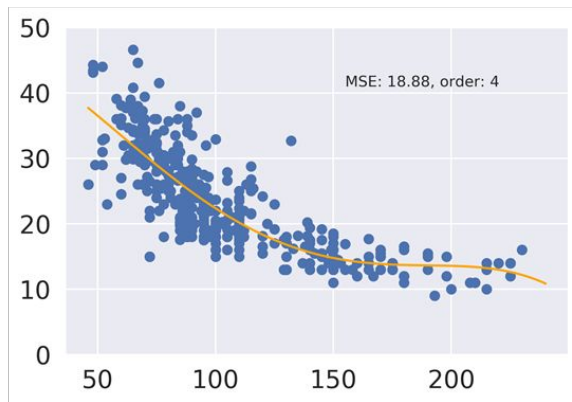
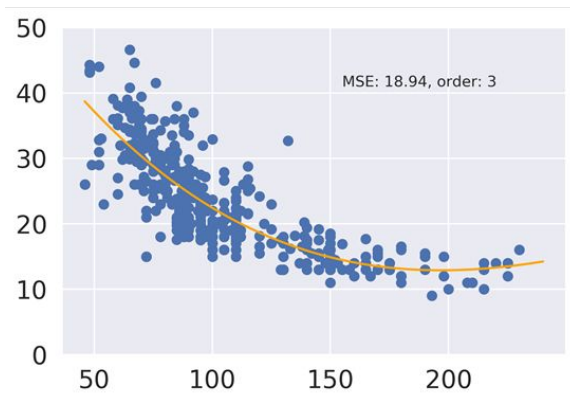
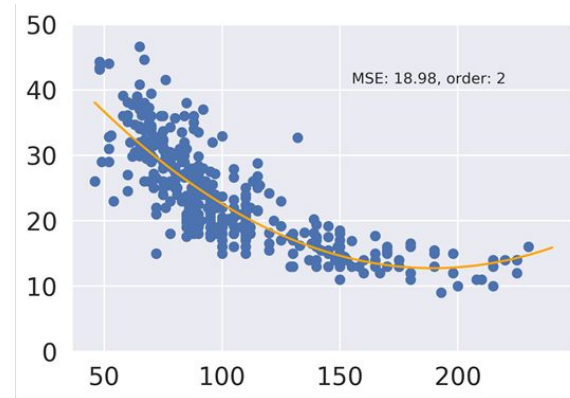
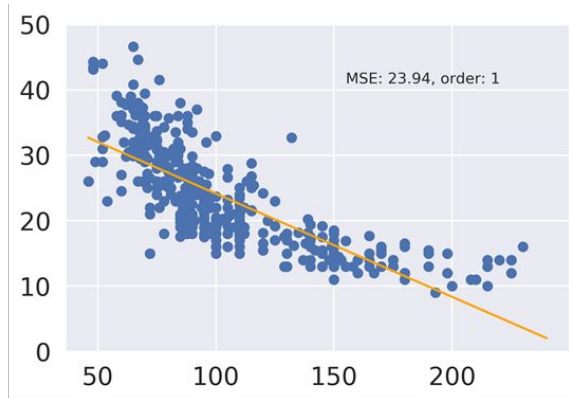
Degree 3 Features

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

We observe a small improvement in MSE.

- Qualitatively, the curve looks quite similar. Only slightly better predict power.
- ... but what happens if we add even higher order features?

Going Even Higher Order



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

The code that I used to generate these models is given below. Uses two out of scope syntax concepts:

- The sklearn Pipeline class.
- The sklearn PolynomialFeatures transformer.

See notebook for today if you're curious.

```
pipelined_model = Pipeline([
    ('josh_transform', PolynomialFeatures(degree = k)),
    ('josh_regression', LinearRegression(fit_intercept = True))
])
pipelined_model.fit(vehicle_data[["hp"]], vehicle_data["mpg"])
```

Variance and Training Error

Lecture 13, Data 100 Spring 2022

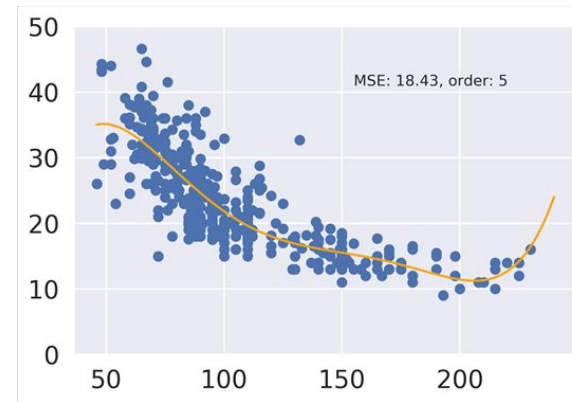
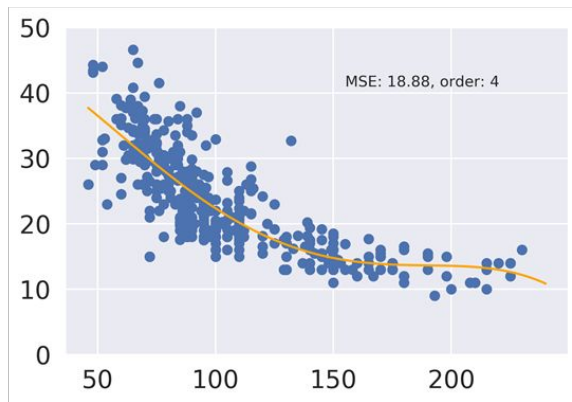
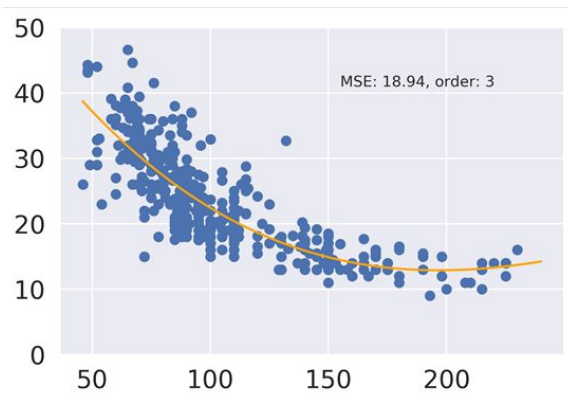
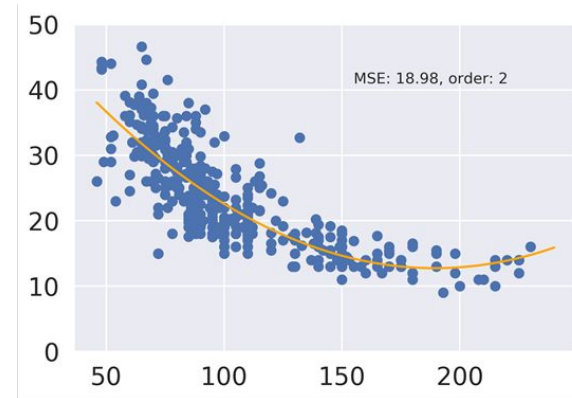
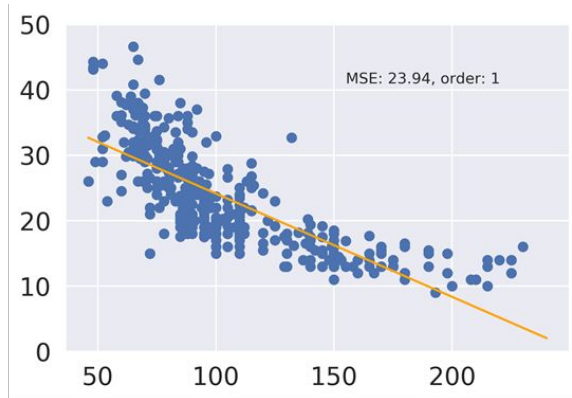
Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- **Variance and Training Error**
- Overfitting
- Detecting Overfitting

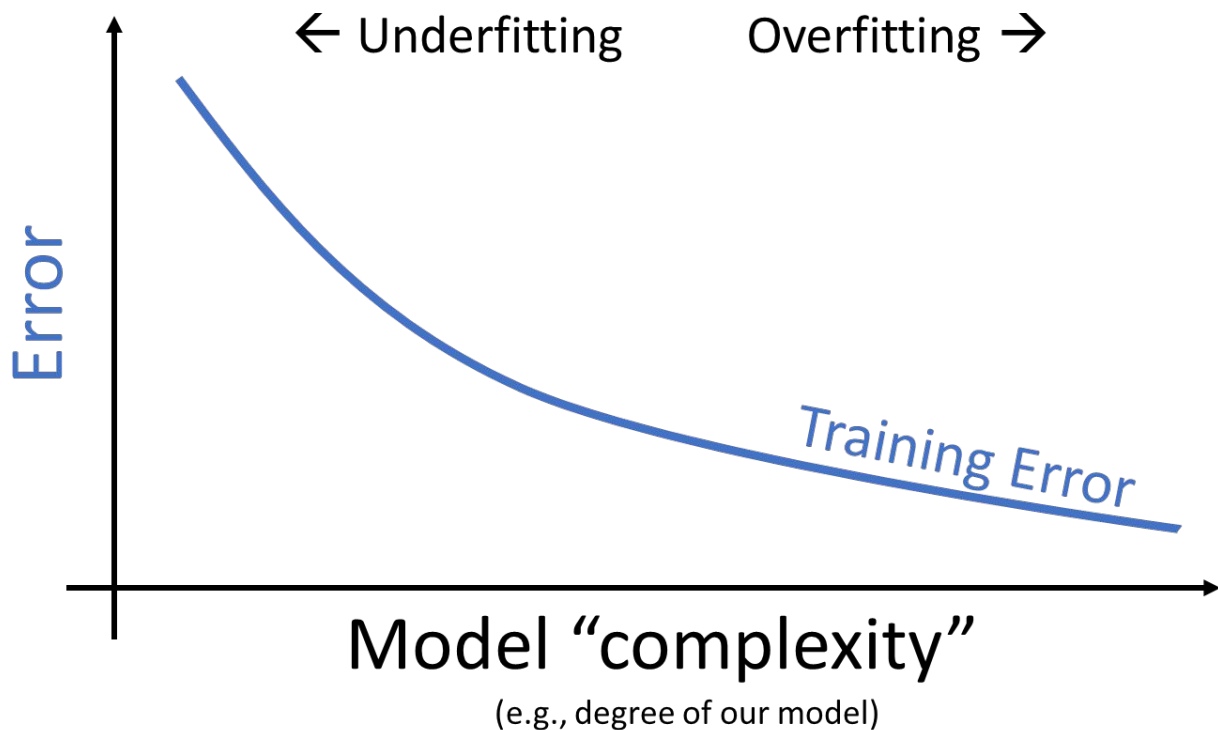
Going Even Higher Order



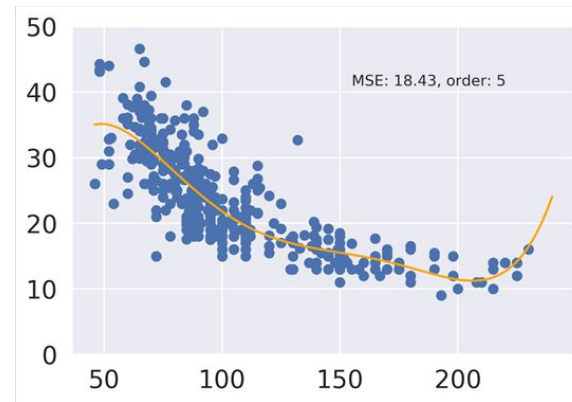
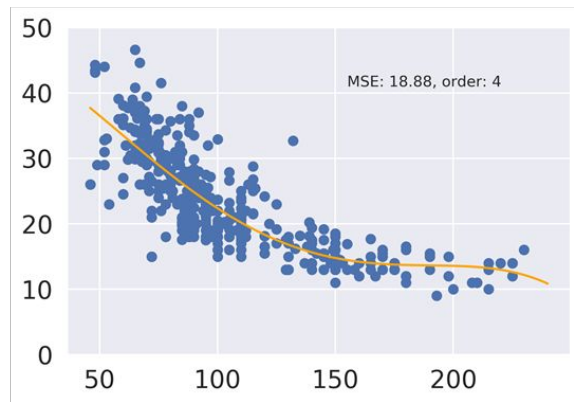
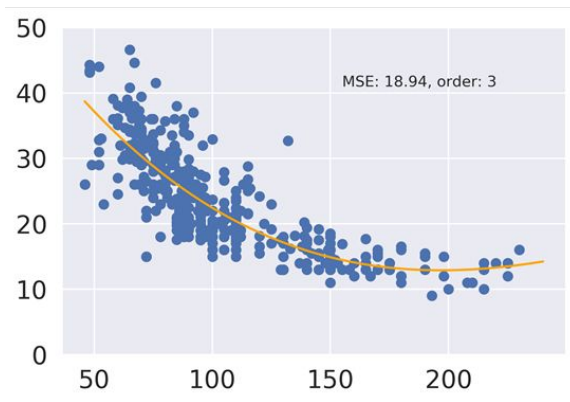
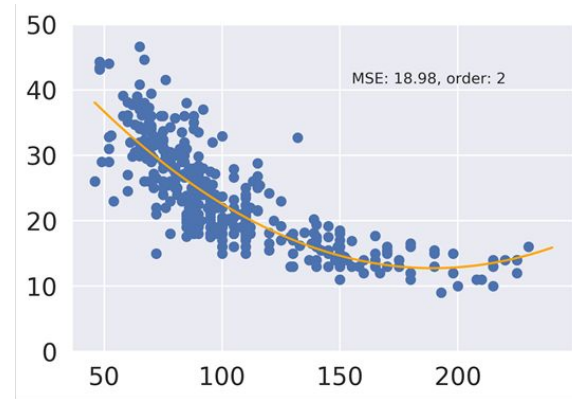
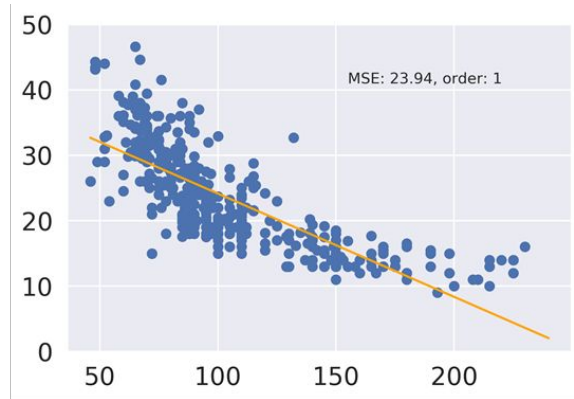
As we increase model complexity, MSE drops from 60.76 to 23.94 to ... 18.43.

Error vs. Complexity

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



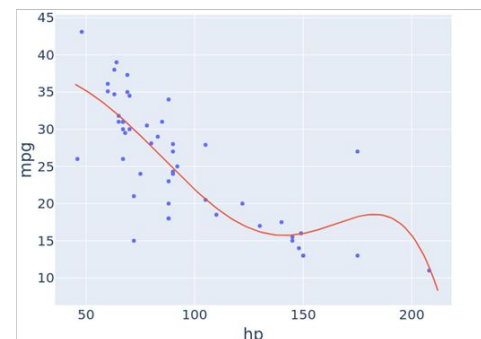
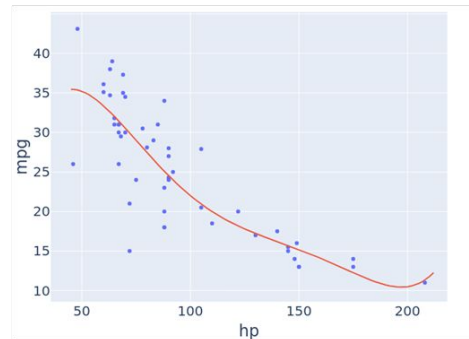
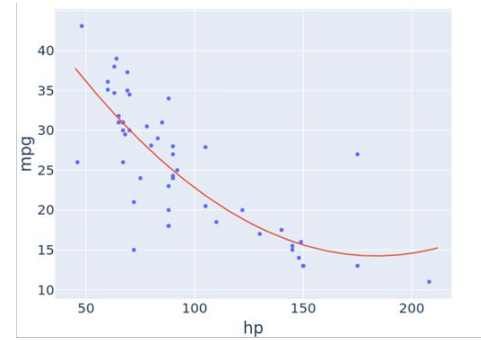
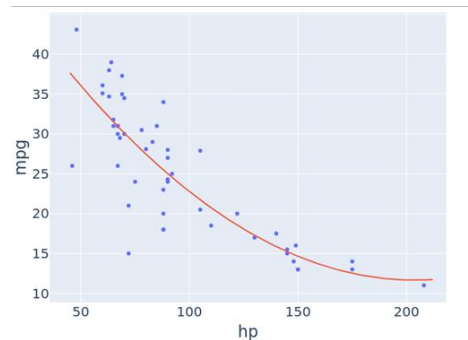
As we **increase model complexity**, **MSE drops** from 60.76 to 23.94 to ... 18.43. **At the same time**, the fit curve grows increasingly erratic and **sensitive** to the data.

Example on a Subset of the Data

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 ($\theta_1 + \dots + \theta_7 x^6$). We see very different predictions, especially for hp around 170.

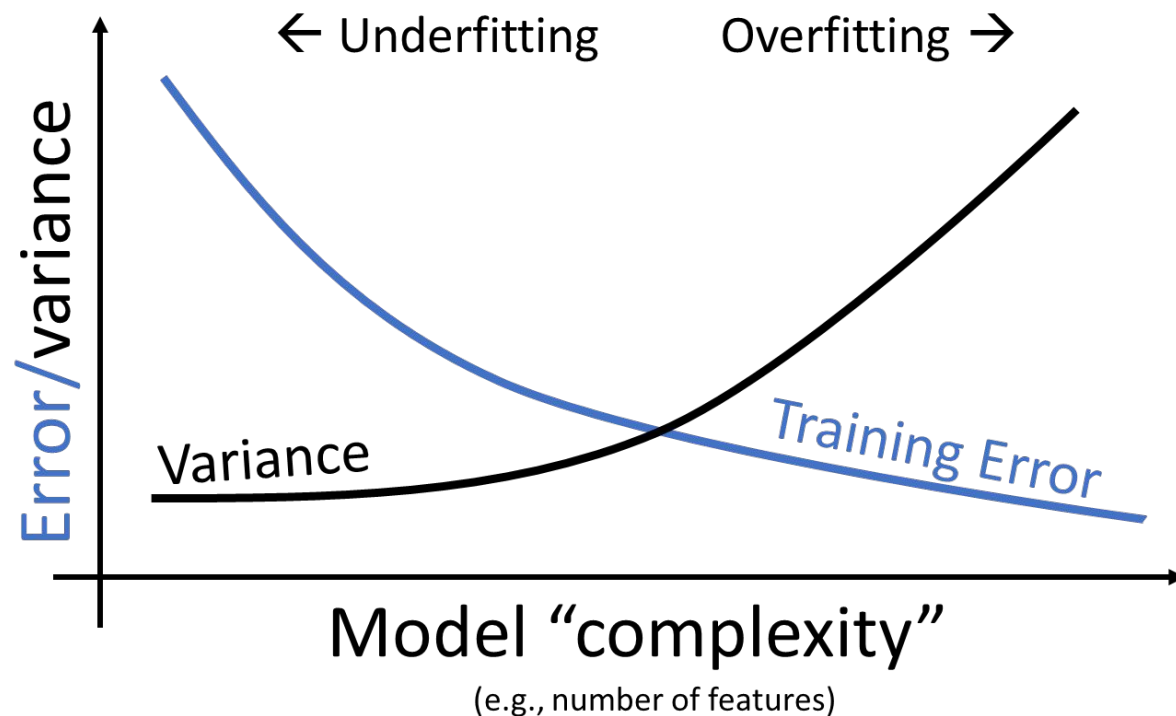
In ML, this **sensitivity** to data is known as “**variance**”.



Error vs. Complexity

As we increase the complexity of our model:

- Training error decreases.
- Variance increases.



Overfitting

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

Feature Engineering:

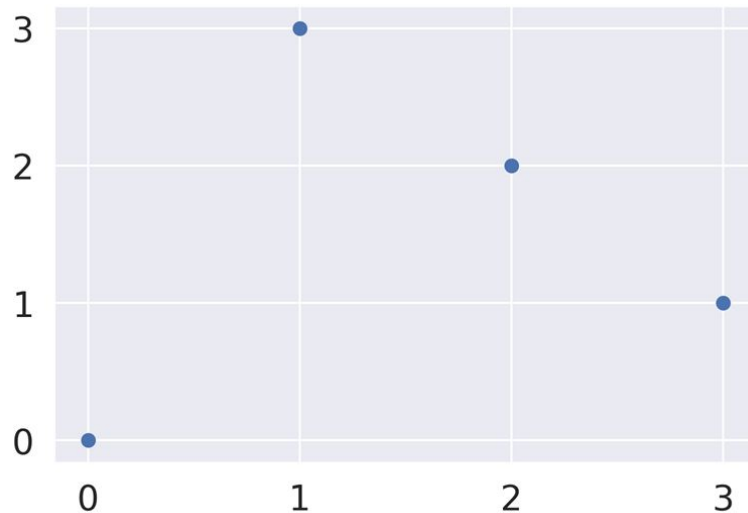
- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- **Overfitting**
- Detecting Overfitting

Four Parameter Model with Four Data Points

Interesting fact: Given N data points, we can always find a polynomial of degree $N-1$ that goes through all those points (as long as no point is directly above any other).

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.



Four Parameter Model with Four Data Points

Interesting fact: Given N data points, we can always find a polynomial of degree $N-1$ that goes through all those points (as long as no point is directly above any other).

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.

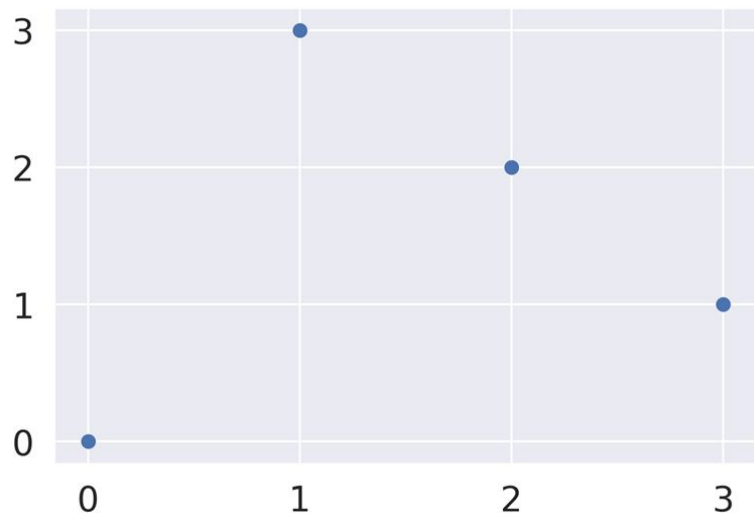
Just solve the system of equations below:

$$\theta_0 = 0$$

$$\theta_0 + \theta_1 + \theta_2 + \theta_3 = 3$$

$$\theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 = 2$$

$$\theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 = 1$$



Four Parameter Model with Four Data Points

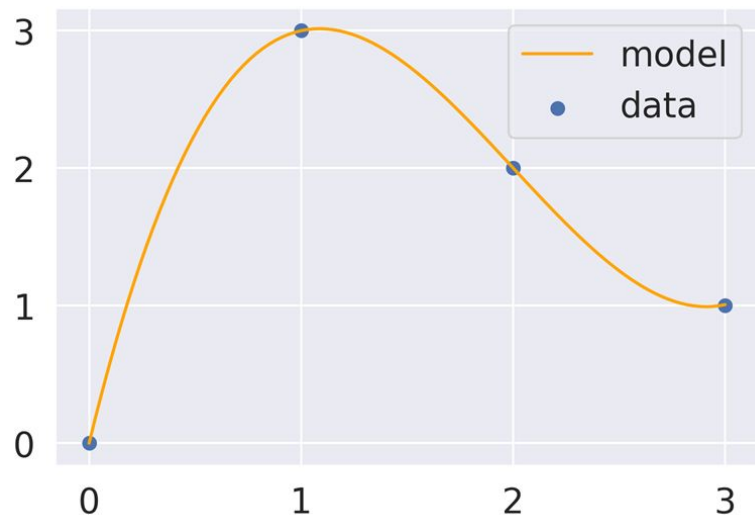
Interesting observation: Given N data points, we can always find a polynomial of degree $N-1$ that goes through all those points.

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.

Just solve the system of equations below:

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



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

Solving our linear equations is equivalent to a matrix inversion.

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

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

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

Can also do this in sklearn:

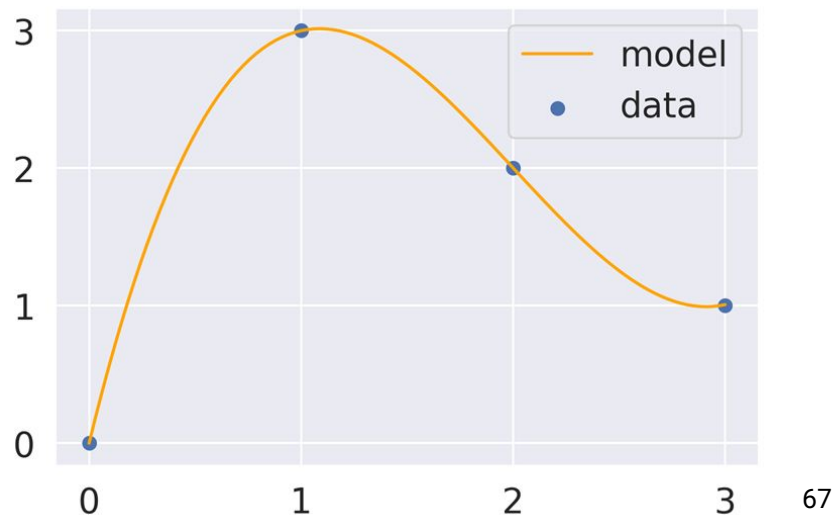
```
model = Pipeline([
    ('josh_transform', PolynomialFeatures(degree = 3, include_bias = False)),
    ('josh_regression', LinearRegression())
])
model.fit(arbitrary_data[["x"]], arbitrary_data["y"])
```

Diagram illustrating the mapping of model parameters to θ values:

- θ_1 points to the first coefficient in the array: 6.33333333
- θ_2 points to the second coefficient in the array: -4.
- θ_3 points to the third coefficient in the array: 0.66666667
- θ_0 points to the intercept value: 5.329070518200751e-15

The parameters are accessed via:

- `model.named_steps["josh_regression"].coef_` for the coefficients.
- `model.named_steps["josh_regression"].intercept_` for the intercept.



The Danger of Overfitting

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

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

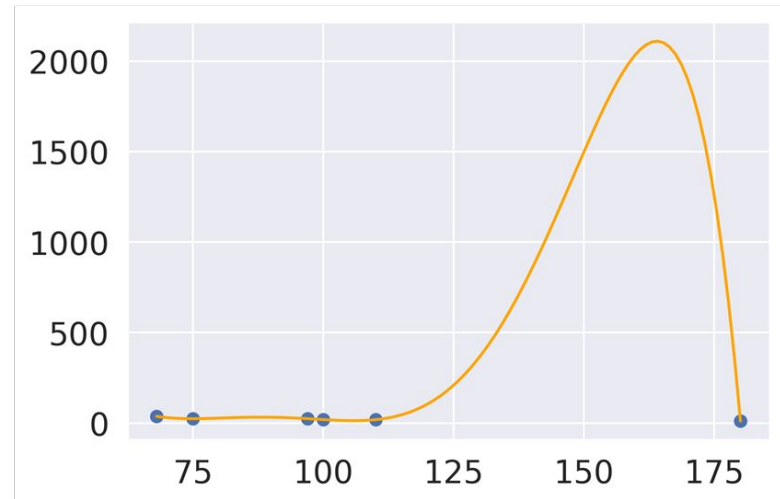
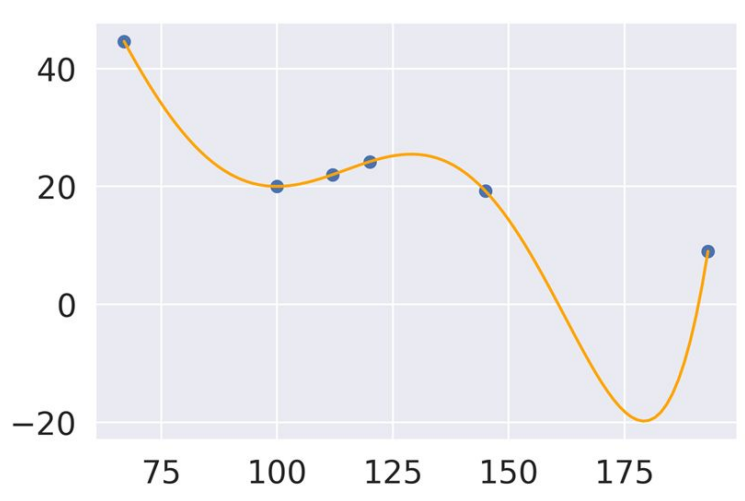
The problem we're facing here is "overfitting". Our model is effectively just memorizing existing data and cannot handle new situations at all.

To get a better handle on this problem, let's build a model that perfectly fits 6 randomly chosen vehicles from our fuel efficiency dataset.

Model Sensitivity in Action

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

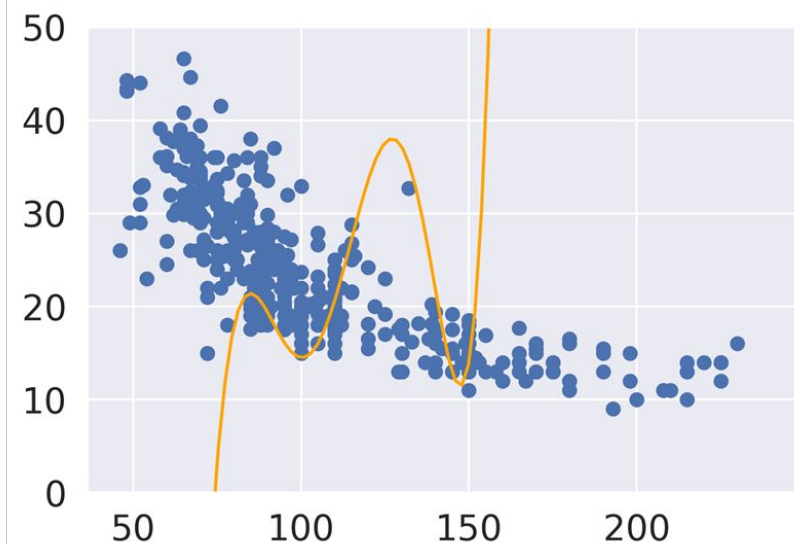
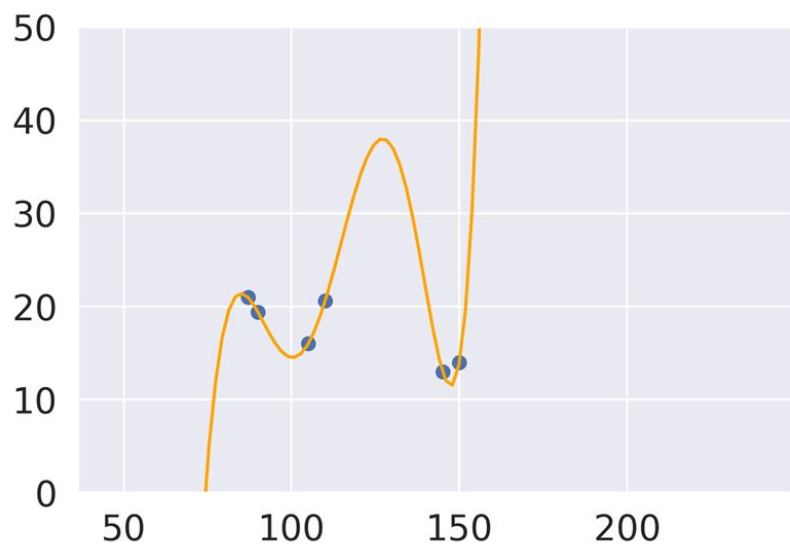
- (*With the caveat that real computers do not have infinite precision, and thus for even higher order models, this will break due to rounding errors. If you start generating features like x^{20} , it will break down because 100^{20} is too big to store.)



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

Consider the model on the left, generated from a sample of six data points. 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).
- ... 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!



Detecting Overfitting

Lecture 13, Data 100 Spring 2022

Gradient Descent Wrap up:

- Stochastic Gradient Descent
- Convexity

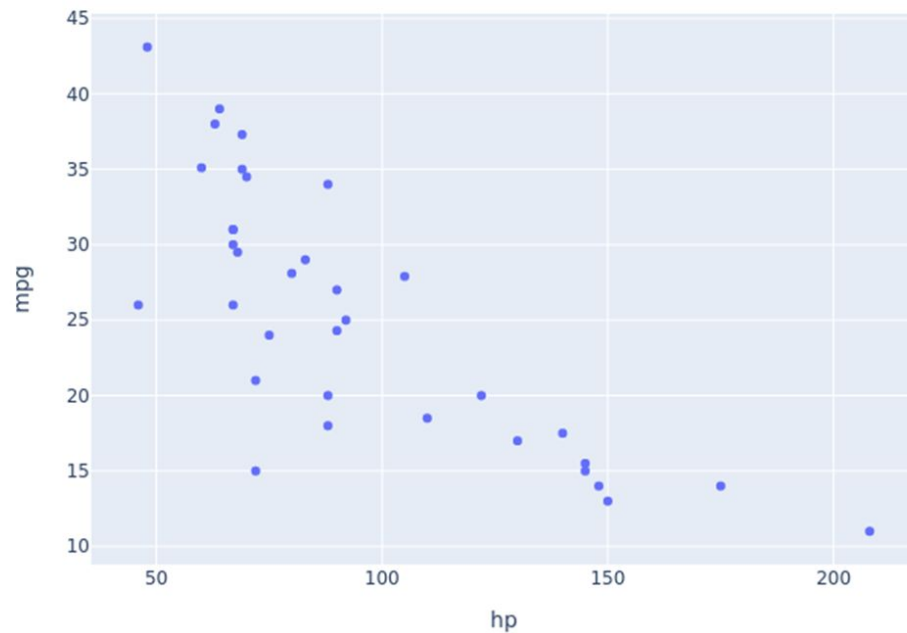
Feature Engineering:

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example (Joey Gonzales)
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- **Detecting Overfitting**

Our 35 Samples

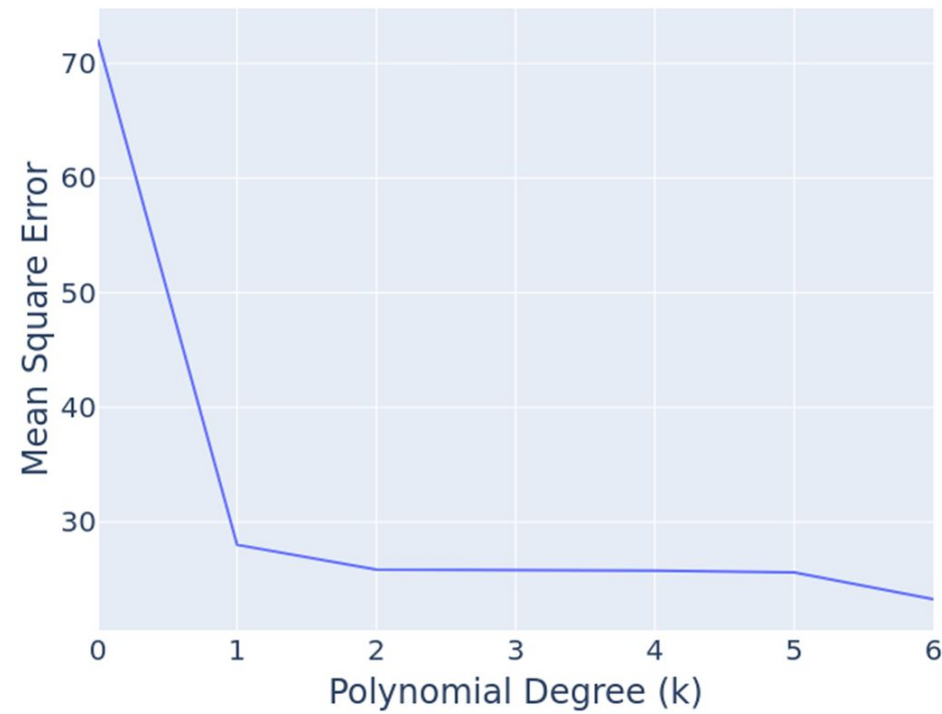
Consider a model fit on only the 35 data points.

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



Fitting Various Degree Models

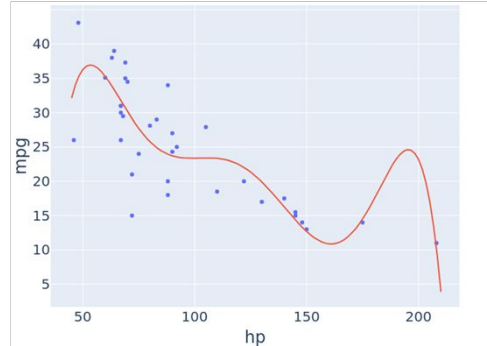
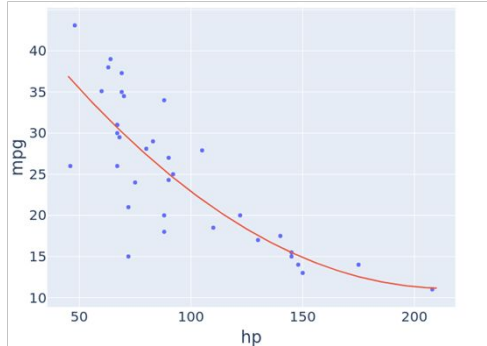
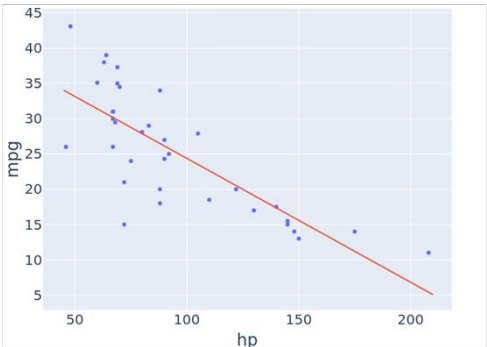
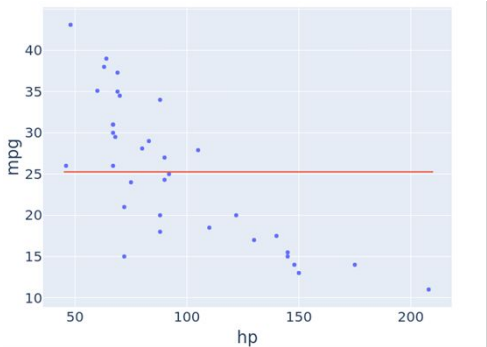
If we fit models of degree 0 through 7 of this model. The MSE is as shown below.



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

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



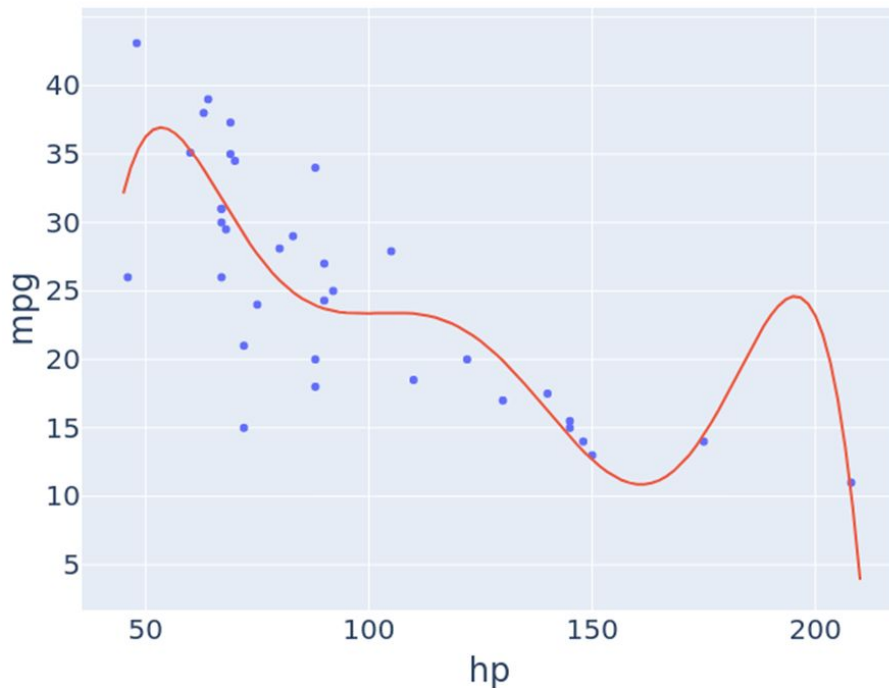
k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001



An Intuitively Overfit Model

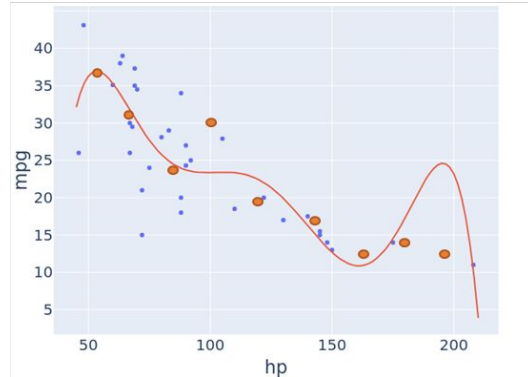
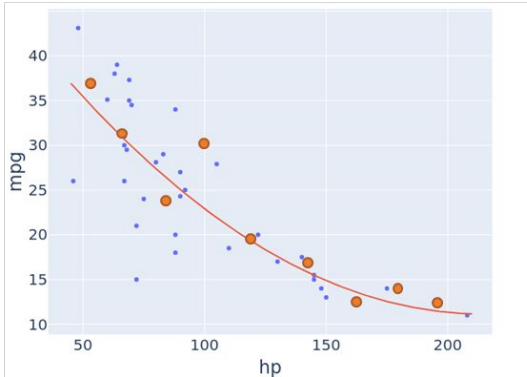
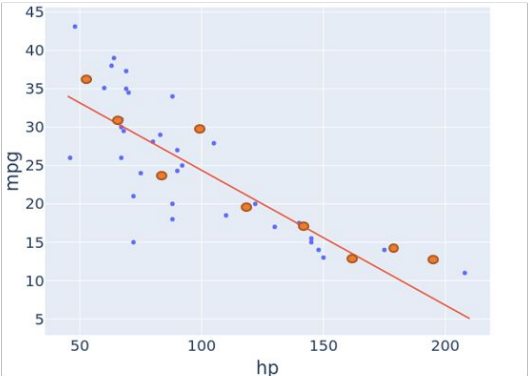
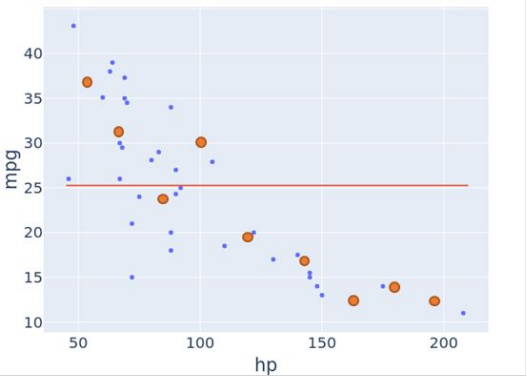
Intuitively, the degree 6 model below feels like it is overfit.

- 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

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

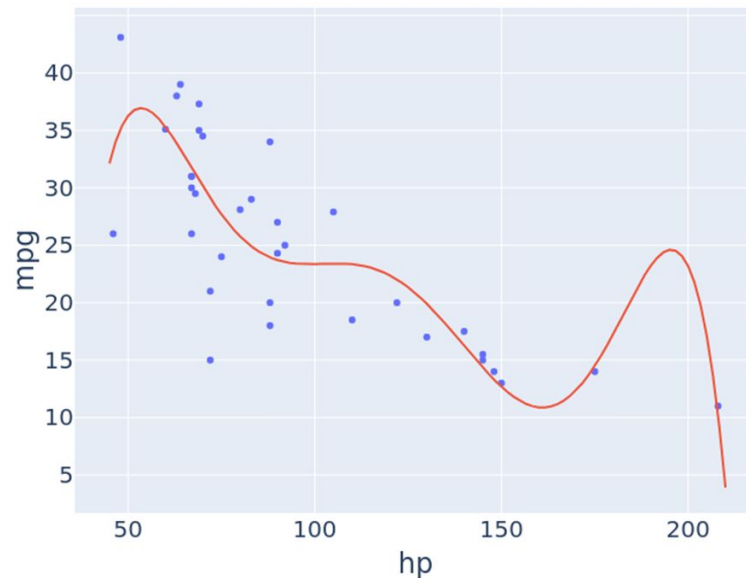
Collecting More Data to Prove a Model is Overfit

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 13

Gradient Descent, Feature Engineering

Content credit: Josh Hug, Joseph Gonzalez