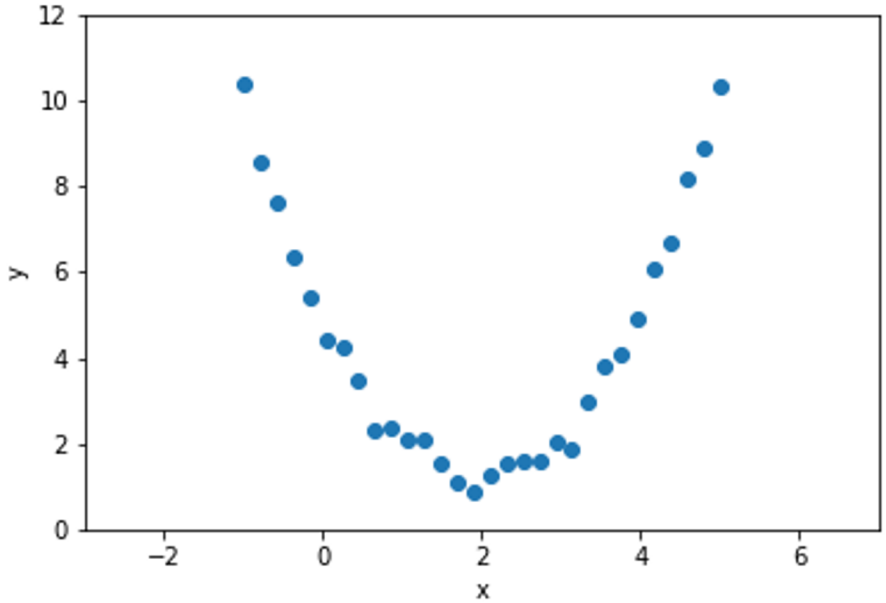
Lecture 13

Feature Engineering

Transforming Data to Improve Our Models.

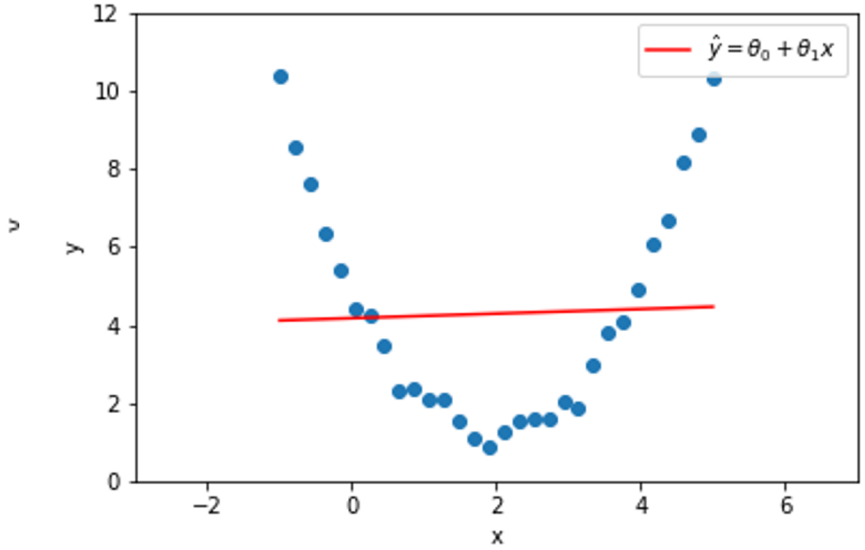
Can we fit a model to this data?



Can we fit a model to this data?

Simple Linear Regression?

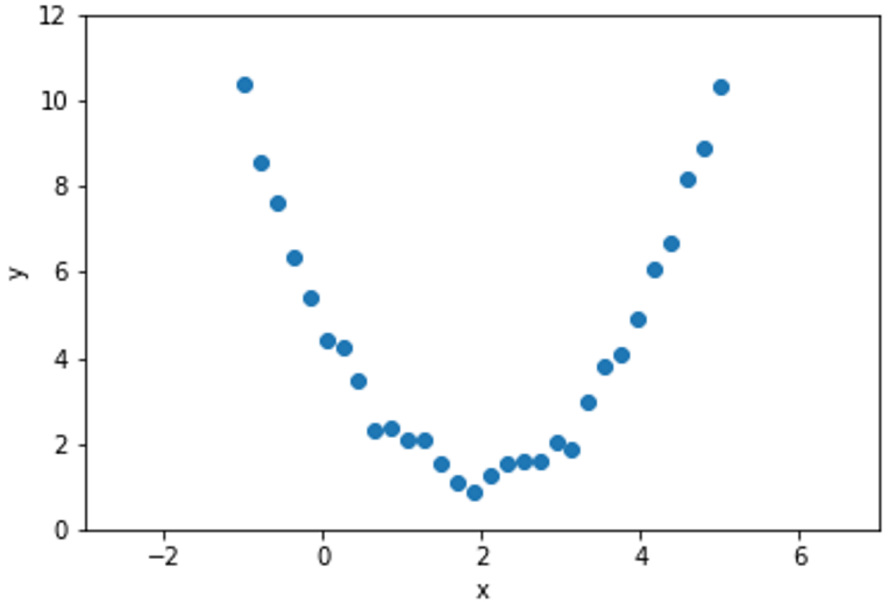
No, because the data is fundamentally nonlinear



Can we fit a model to this data?

Multiple Linear Regression?

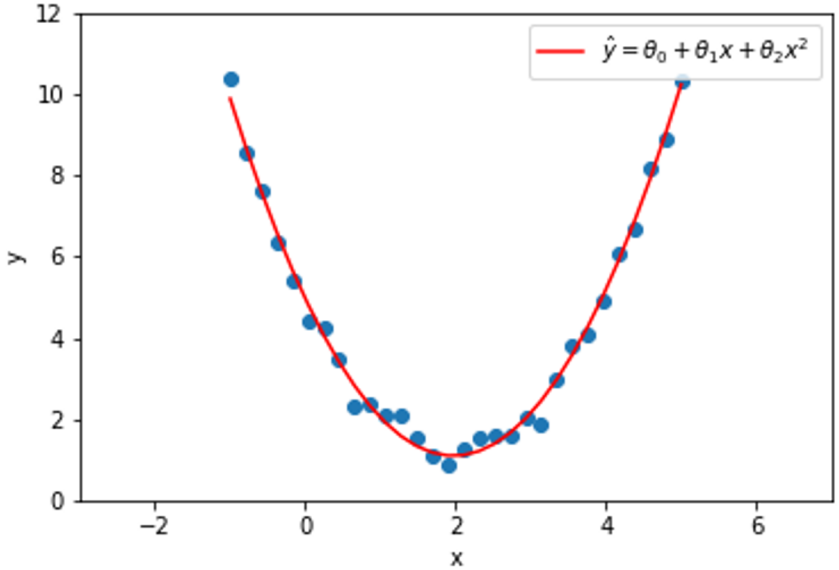
No, because there are no other features to add



Can we fit a model to this data?

Idea: *Create* an extra feature to use in the model. What feature should we add?

Since the data looks like a parabola, let's add a quadratic feature



What is a feature?

A feature is an input to our model

- So far, we have just used the raw data as features

We can also create new features to use an inputs to our model

We can choose/create **x** any way we like as long as our model follows the form $\hat{y} = x^T \theta$

The rest of the lecture will discuss different techniques we can use to create **x**:

- How can we create features from **quantitative data**?
- How can we create features from **categorical data**?
- How can we create features from **text data**?

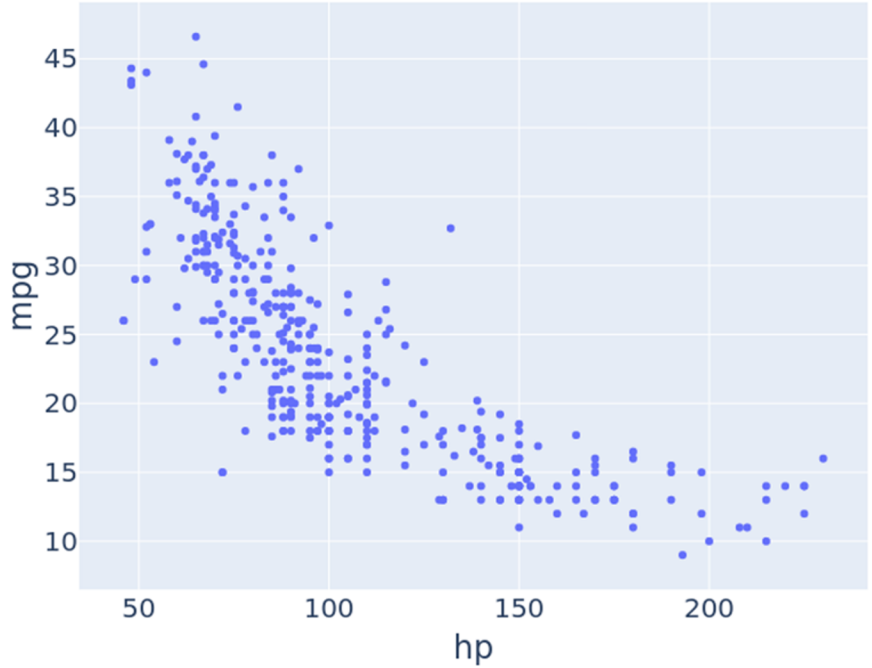
Today's Roadmap

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

A Challenge for Linear Models

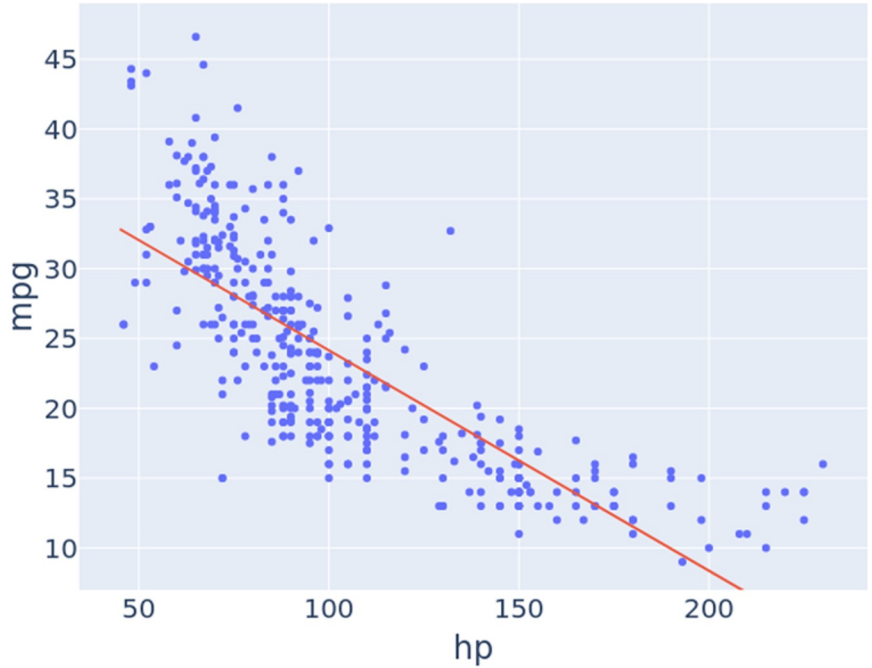
The plot below shows fuel efficiency vs. engine power of many different models of car.

- 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.



MSE: 23.94

Fitting a… Parabola?

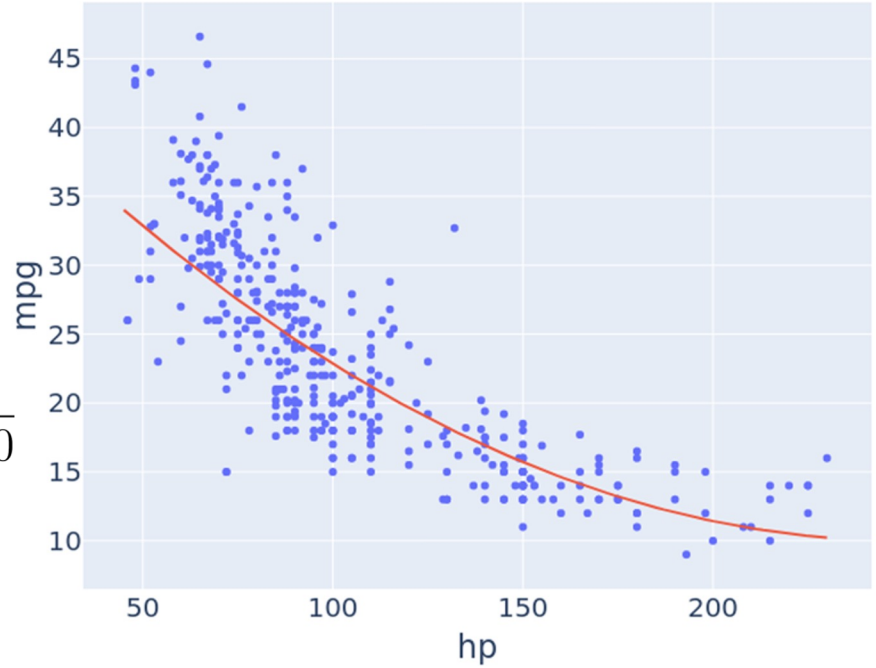
Just eyeballing this data, it seems that a quadratic model might do a better job on the range of data given.

- 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!

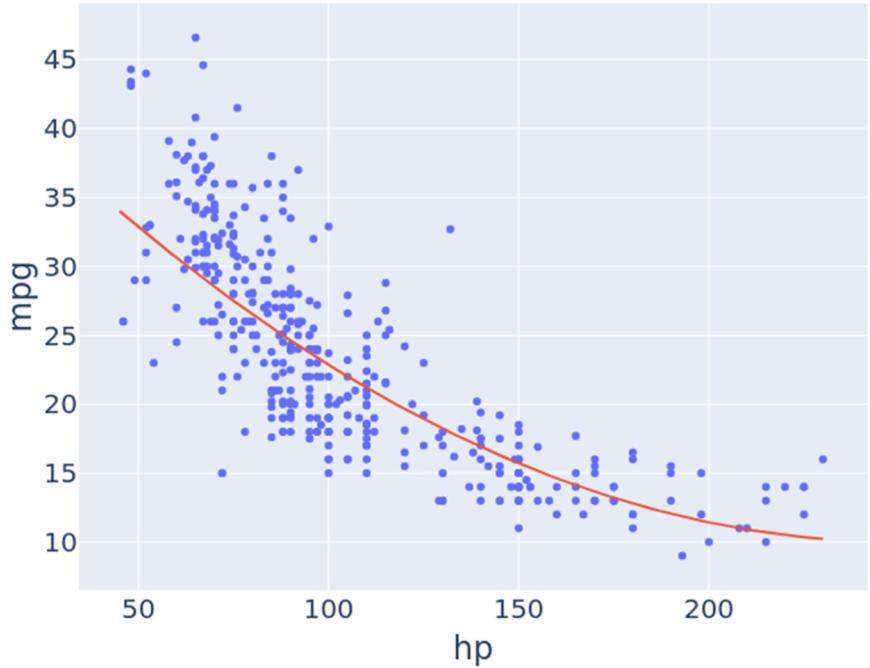
Our Model Is Nonlinear in X

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:

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

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



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.

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

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

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

392 rows × 3 columns

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

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

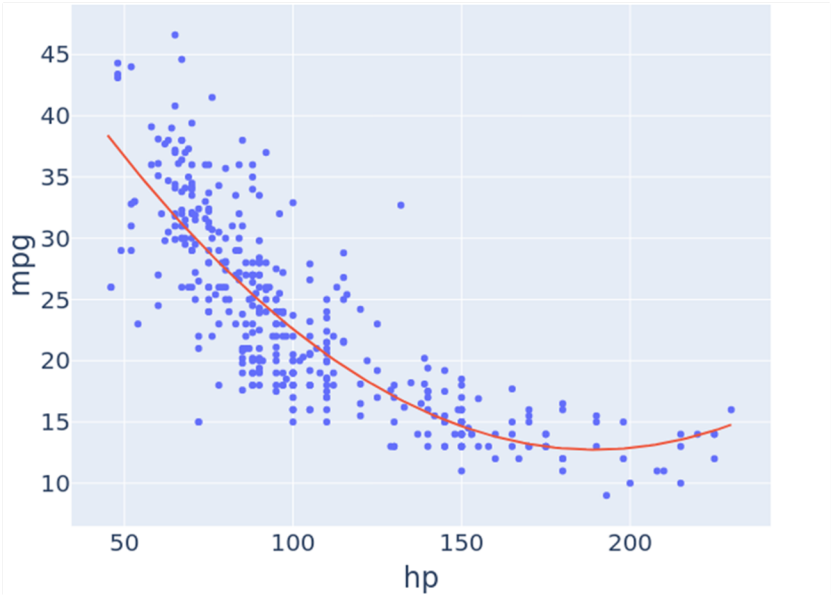
Result on next slide

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'])
```

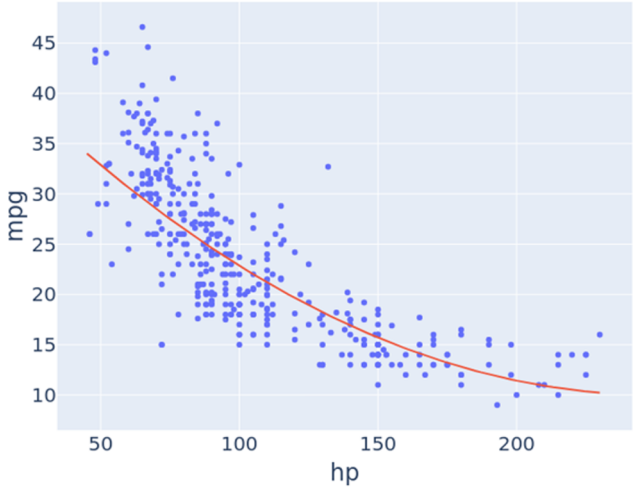


Comparing Our Models

My eyeballed parabolic model:

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

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

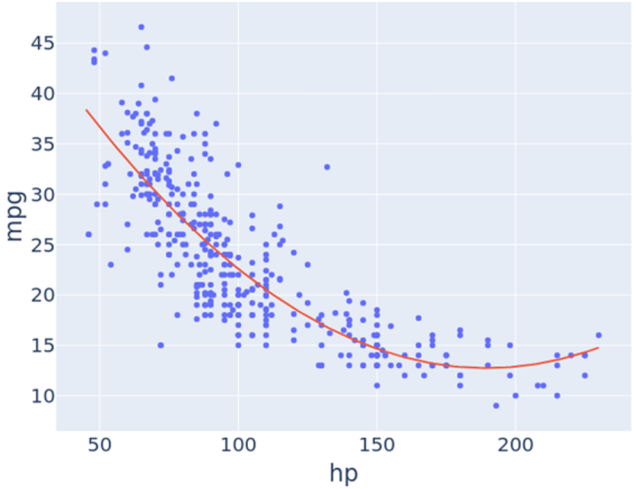


MSE: 21

Our linear regression model using hp and hp^2.

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

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



MSE: 18.98

Feature Engineering Overview

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

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.

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:

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

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

Feature Functions

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 the project.
- 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.

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

High Dimensional Feature Engineering Example

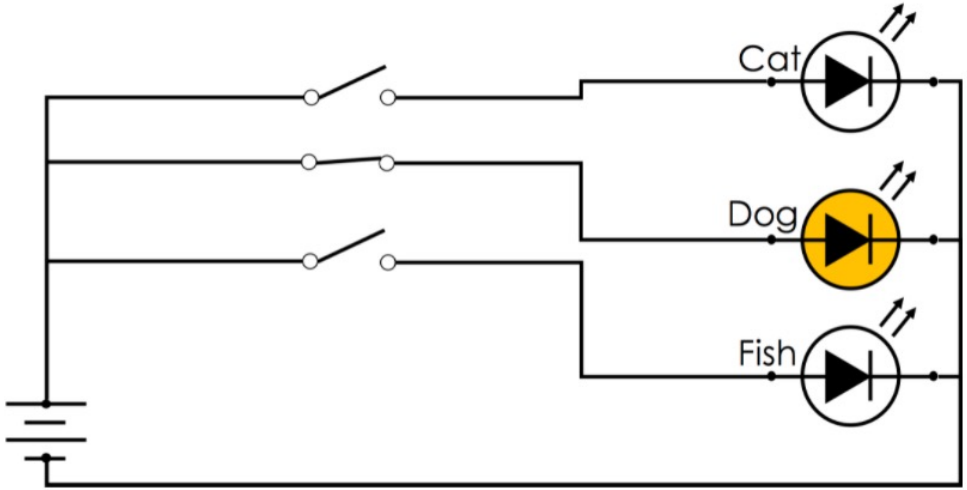
- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- **High Dimensional Feature Engineering Example**
- One Hot Encoding
- Word Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

One Hot Encoding

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

One-Hot Encoding

- One-Hot encoding, sometimes also called **dummy encoding**
- It is a simple mechanism to encode categorical data as real numbers such that the magnitude of each dimension is meaningful. Suppose a feature can take on k distinct values
- For each distinct *possible* value, a new feature (dimension) is created. For each record, all the new features are set to zero except the one corresponding to the value in the original feature.
- The term one-hot encoding comes from a digital circuit encoding of a categorical state as particular "hot" wire:



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.

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

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

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)
```

Fitting a Model

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?

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?

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

Verifying in Python

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.

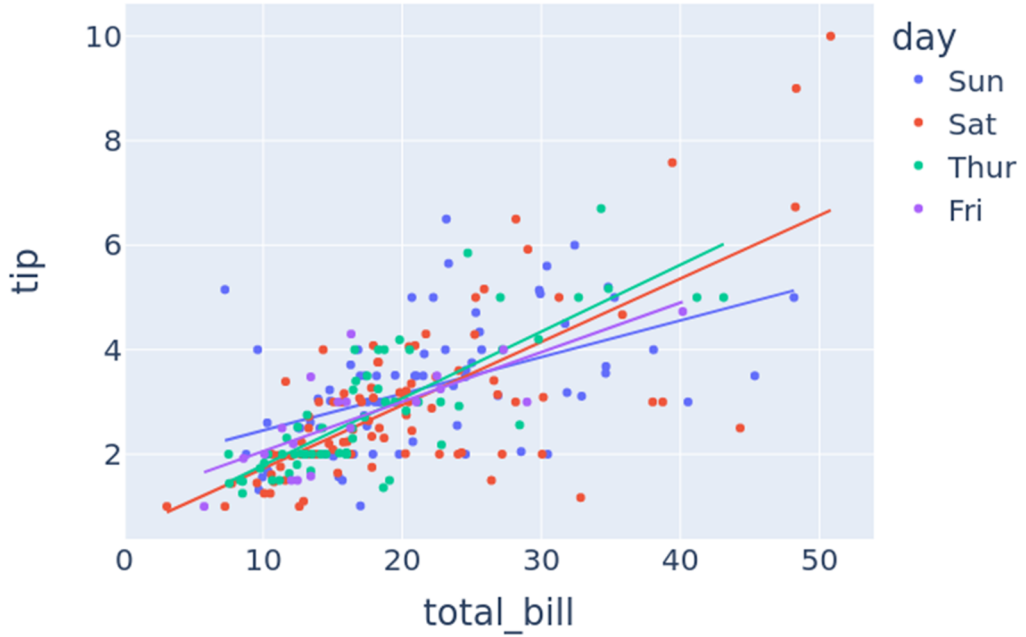
- Bizarre to imagine that humans have a base tip that they start with for every day of the week.
- My guess: This model will not generalize well to newly collected data.

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")
```

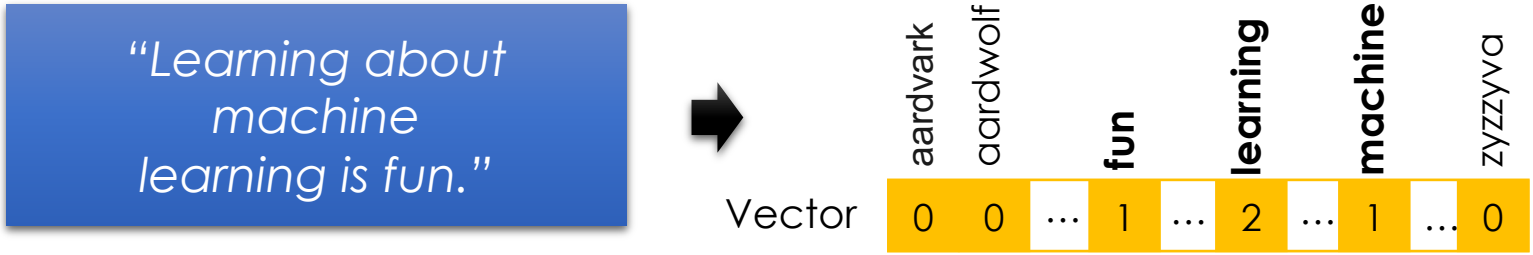


Word Encoding

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

Bag-of-words Encoding

- Generalization of one-hot-encoding for a string of text:



- Encode text as a long vector of word counts (Issues?)
  - Typically high dimensional (millions of columns) and very sparse
  - Word order information is lost… (is this an issue?)
  - What happens when you see a word not in the dictionary?

- A bag is another term for a multiset: *an unordered collection which may contain multiple instances of each element.*

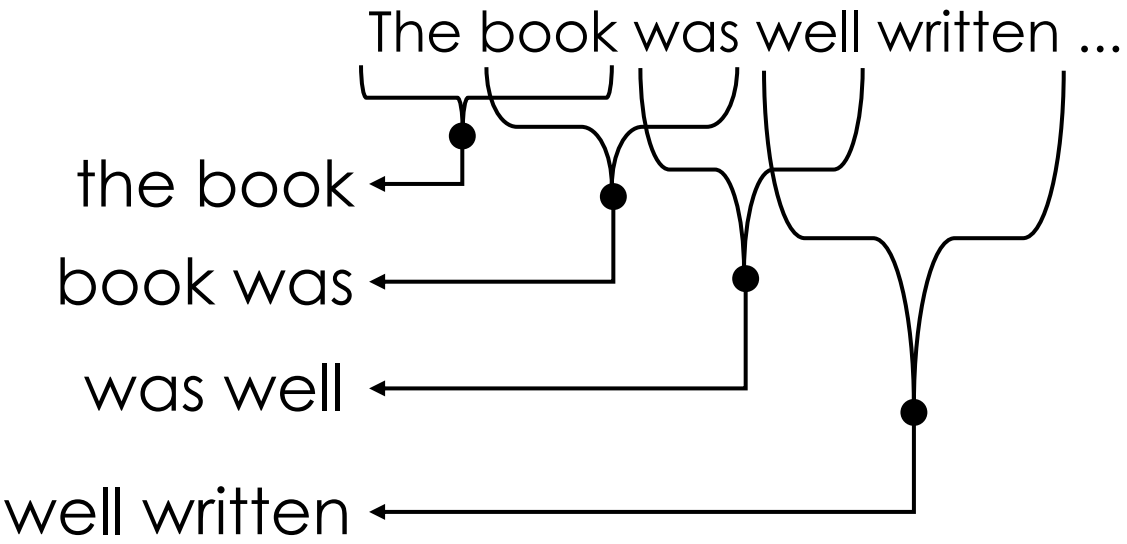
- Stop words: words that do not contain significant information
  - Examples: the, in, at, or, on, a, an, and …
  - Typically removed

N-Gram Encoding

- Sometimes word order matters:

*The book was **not** well written but I did enjoy it.* ➡ *The book was well written but I did **not** enjoy it.*

- How do we capture word order in a “vector” model?
  - N-Gram: *“Bag-of- sequences-of-words”*



2-Gram Encoding

	aardvark is	apple shines	...	**the book**	...	**book was**	...	**was well**	...	**well written**	...	zebra ran
Vector	0	0	...	1	...	1	...	1	...	1	...	0

N-Gram Encoding

- Sometimes word order matters:

*The book was **not** well written but I did enjoy it.* ➡ *The book was well written but I did **not** enjoy it.*

- How do we capture word order in a “vector” model?
  - N-Gram: *“Bag-of- sequences-of-words”*

- Issues:
  - Can be very sparse (many combinations occur only once)
  - Many combinations will only occur at prediction time

High Order Polynomial Example

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

Cubic Fit

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:

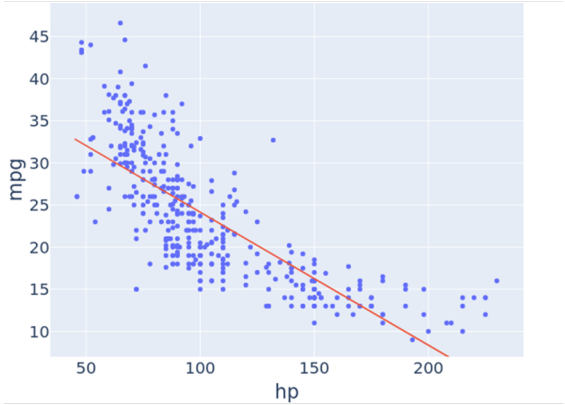
hp	hp2	hp3	mpg
130.0	16900.0	2197000.0	18.0
165.0	27225.0	4492125.0	15.0
150.0	22500.0	3375000.0	18.0
150.0	22500.0	3375000.0	16.0
140.0	19600.0	2744000.0	17.0

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

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

Cubic Fit Results

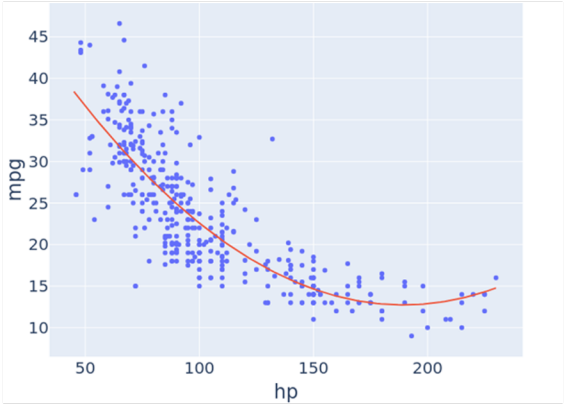
MSE: 21



Degree 1 Features

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

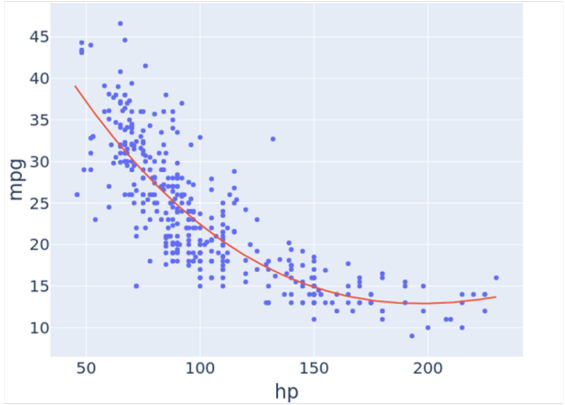
MSE: 18.98



Degree 2 Features

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

MSE: 18.94



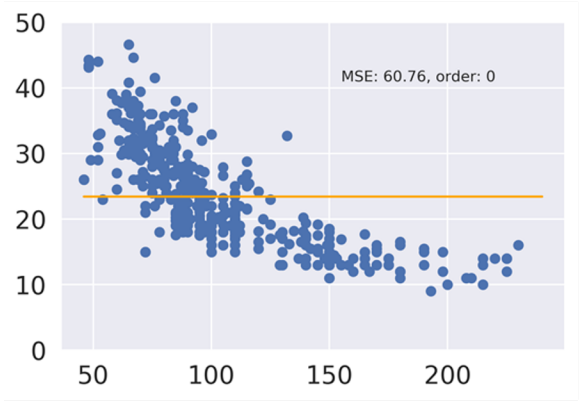
Degree 3 Features

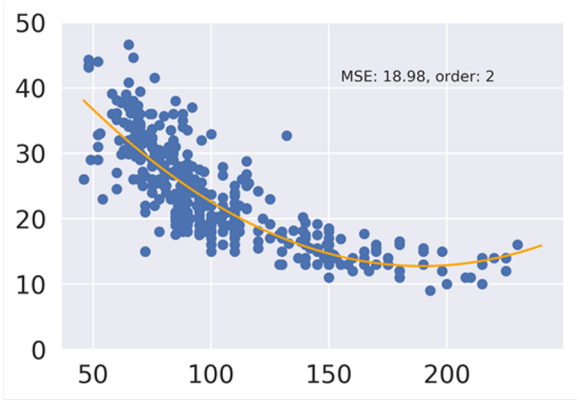
```
fit(vehicle_data[['hp', 'hp2', 'hp3']]
```

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.

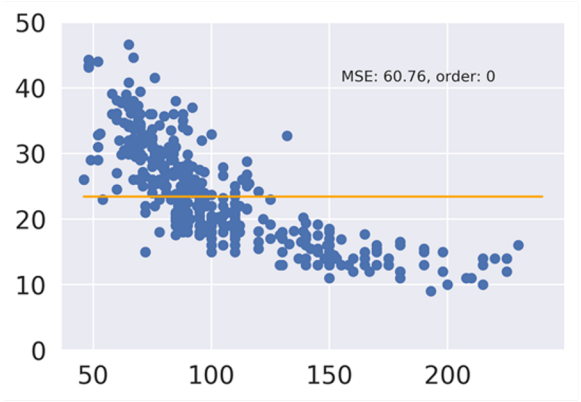
See notebook for today if you're curious.

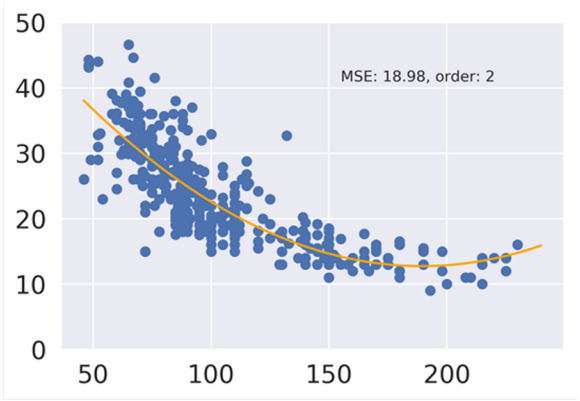
```
pipelined_model = Pipeline([
    ('josh_transform', PolynomialFeatures(degree = k)),
    ('josh_regression', LinearRegression(fit_intercept = True))
])
pipelined_model.fit(vehicle_data[["hp"]], vehicle_data["mpg"])
```

Variance and Training Error

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

Going Even Higher Order

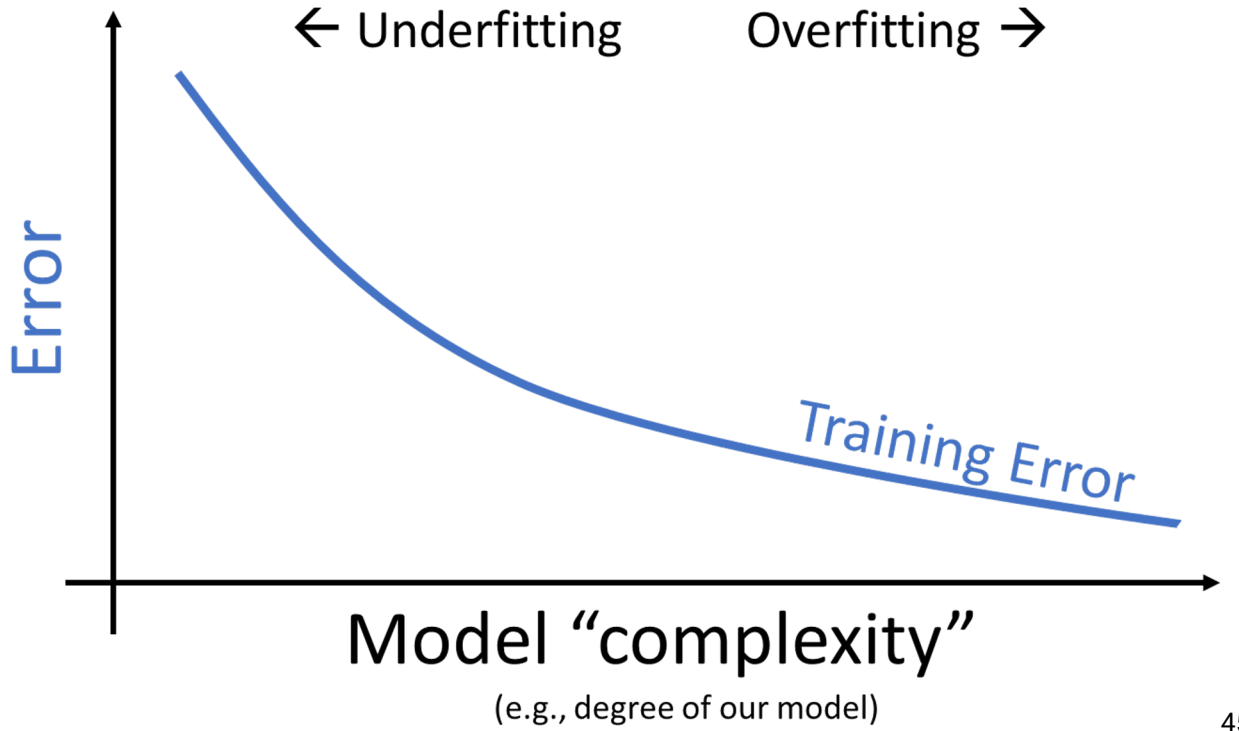




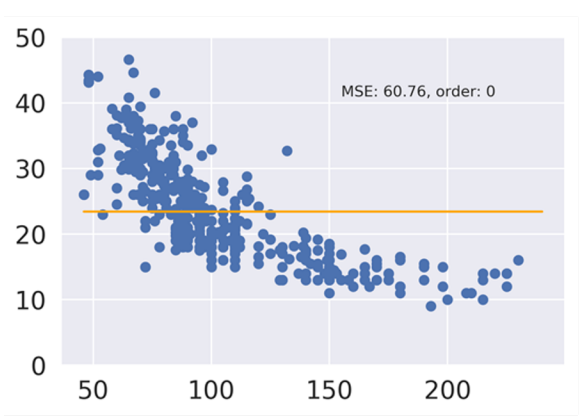
As we increase model complexity, MSE drops from 60.76 to 23.94 to … 18.43.

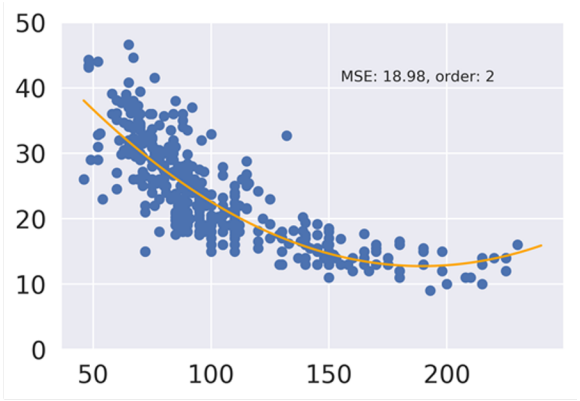
Error vs. Complexity

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





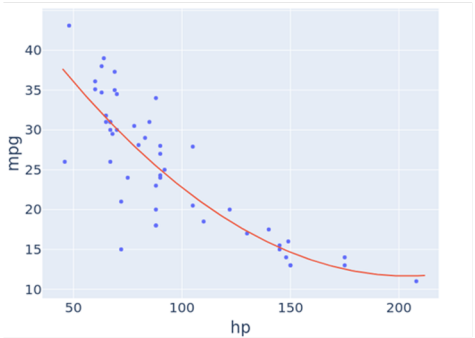
As we **increase model complexity**, **MSE drops** from 60.76 to 23.94 to … 18.43. **At the same time**, the fit curve grows increasingly erratic and **sensitive** to the data.

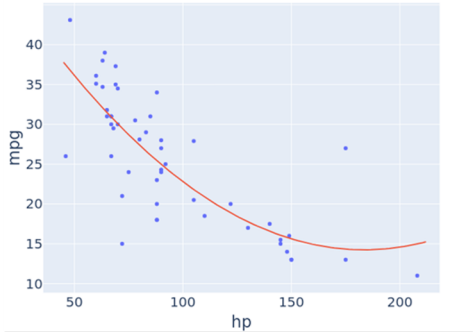
Example on a Subset of the Data

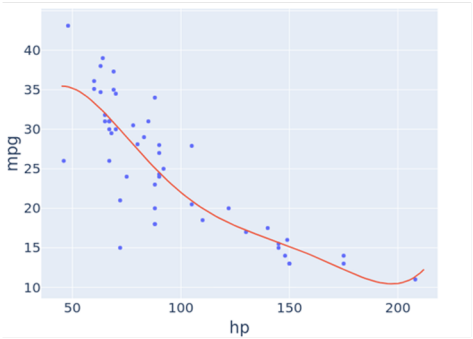
On top, we see the results of fitting two very similar datasets using an order 2 model ($\theta_1 + \theta_2\, x + \theta_3\, x^2$). The resulting fit (model parameters) is close.

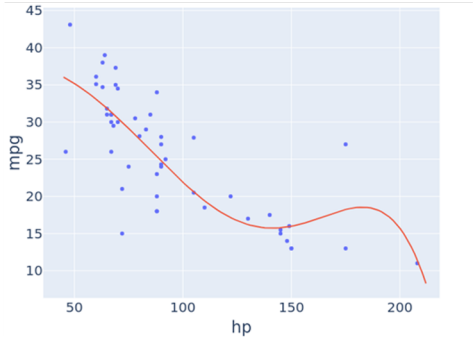
On bottom, we see the results of fitting the same datasets using an order 6 model ($\theta_1 + \ldots + \theta_7\, x^6$). We see very different predictions, especially for hp around 170.

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





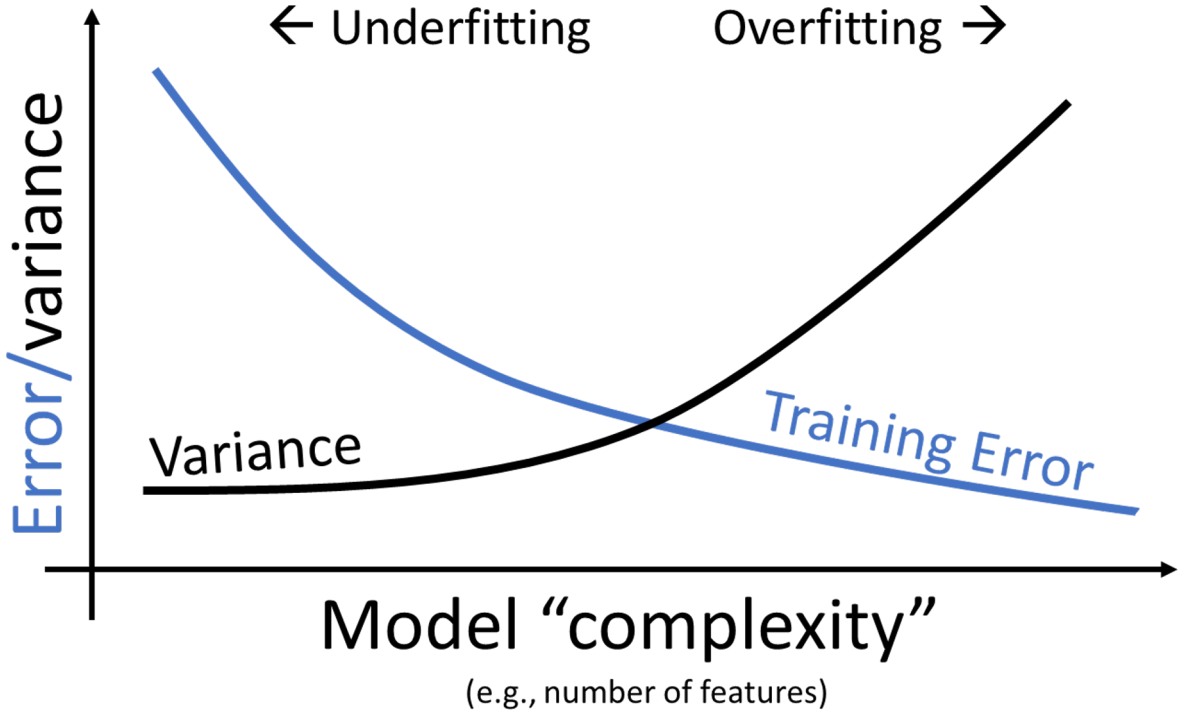




Error vs. Complexity

As we increase the complexity of our model:

- Training error decreases.
- Variance increases.



Overfitting

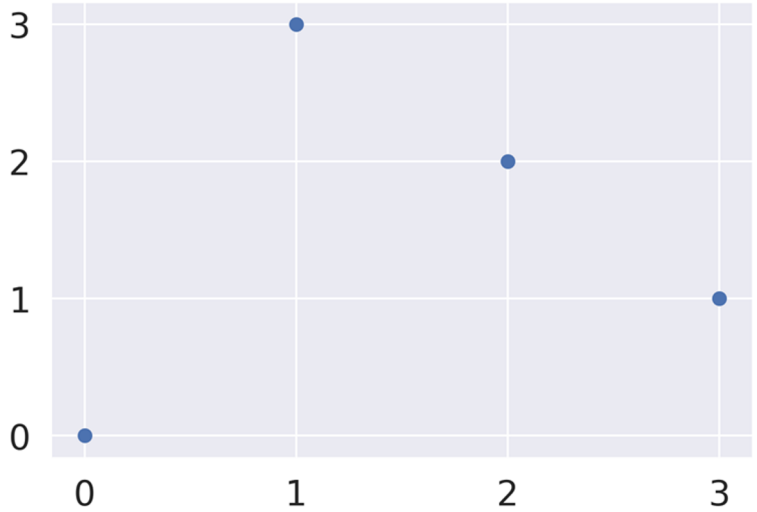
- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example
- One Hot Encoding
- Word Encoding
- High Order Polynomial Example
- Variance and Training Error
- **Overfitting**
- Detecting Overfitting

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

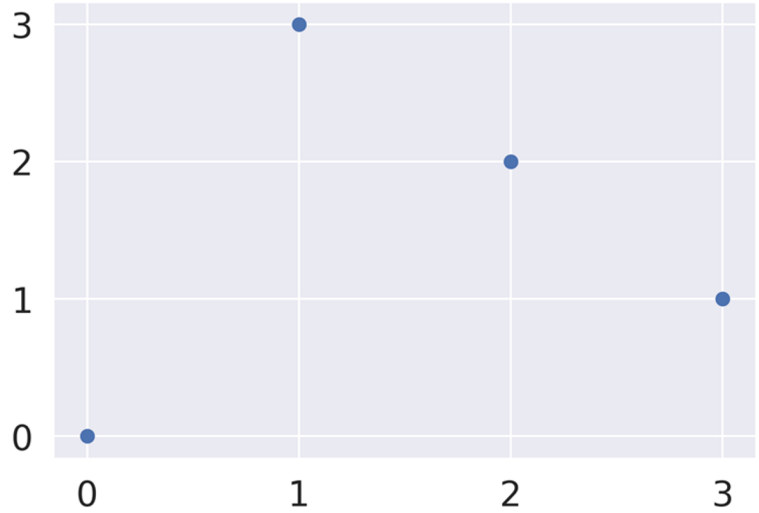
Interesting fact: Given N data points, we can always find a polynomial of degree N-1 that goes through all those points (as long as no point is directly above any other).

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

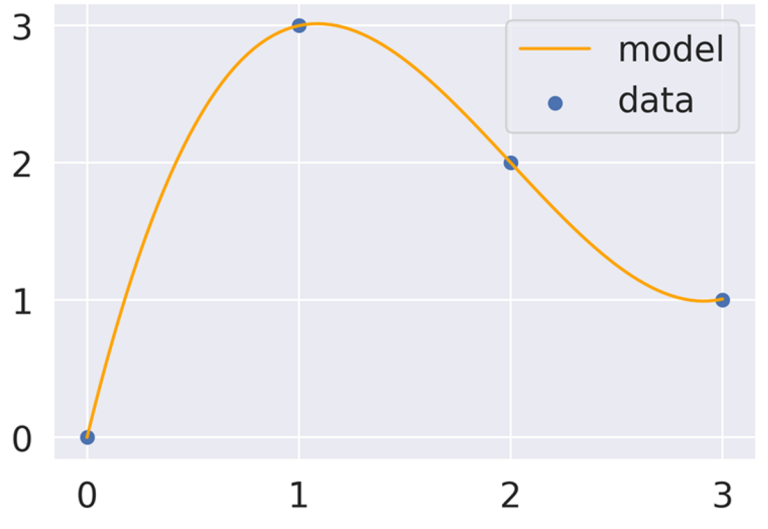
Interesting observation: Given N data points, we can always find a polynomial of degree N-1 that goes through all those points.

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:

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



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

Solving our linear equations is equivalent to a matrix inversion.

$$\theta_0 = 0$$
$$\theta_0 + \theta_1 + \theta_2 + \theta_3 = 3$$
$$\theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 = 2$$
$$\theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 = 1$$

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

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

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

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.

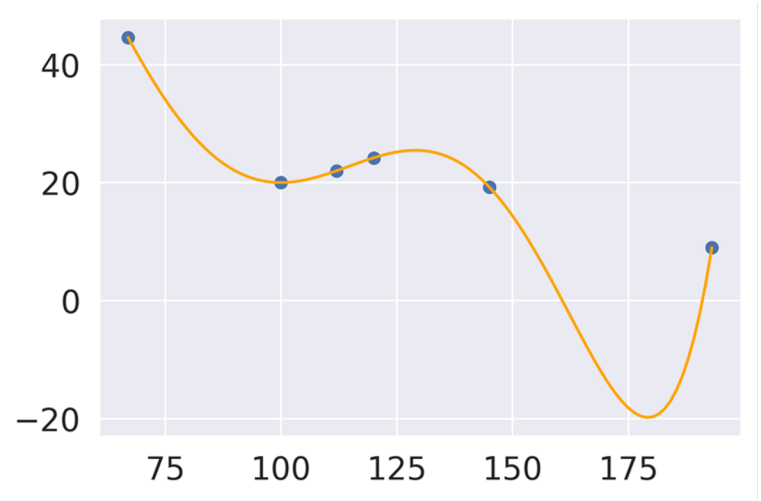
The problem we're facing here is "overfitting". Our model is effectively just memorizing existing data and cannot handle new situations at all.

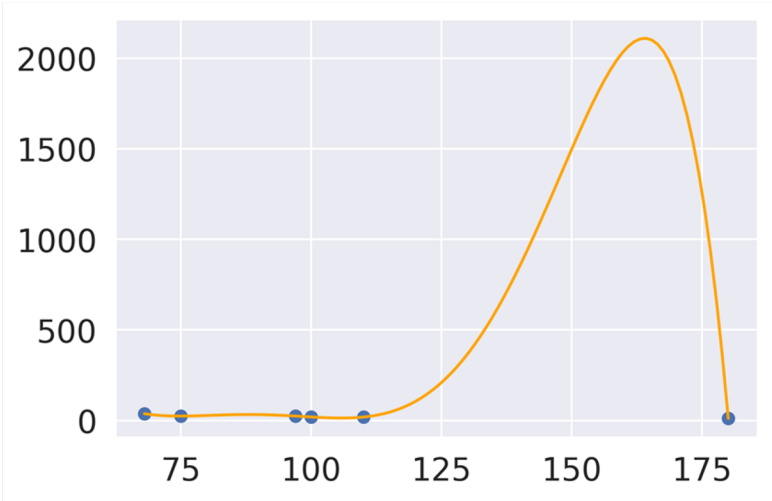
To get a better handle on this problem, let's build a model that perfectly fits 6 randomly chosen vehicles from our fuel efficiency dataset.

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.)

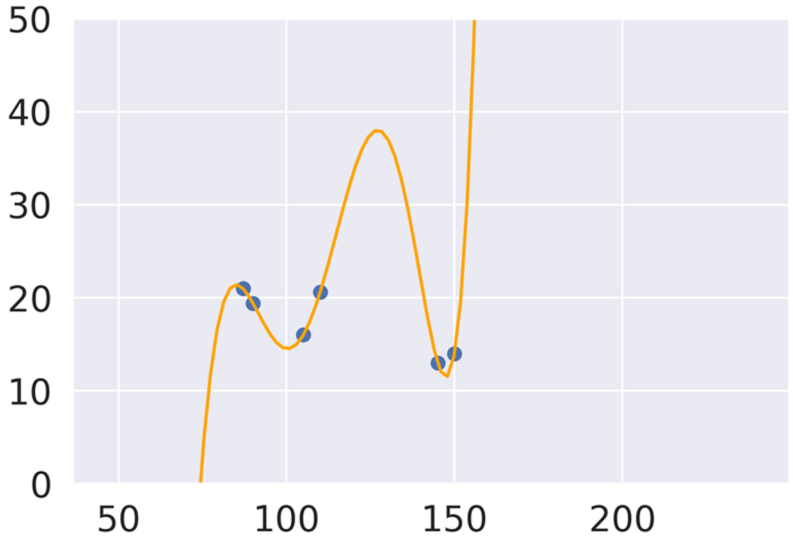


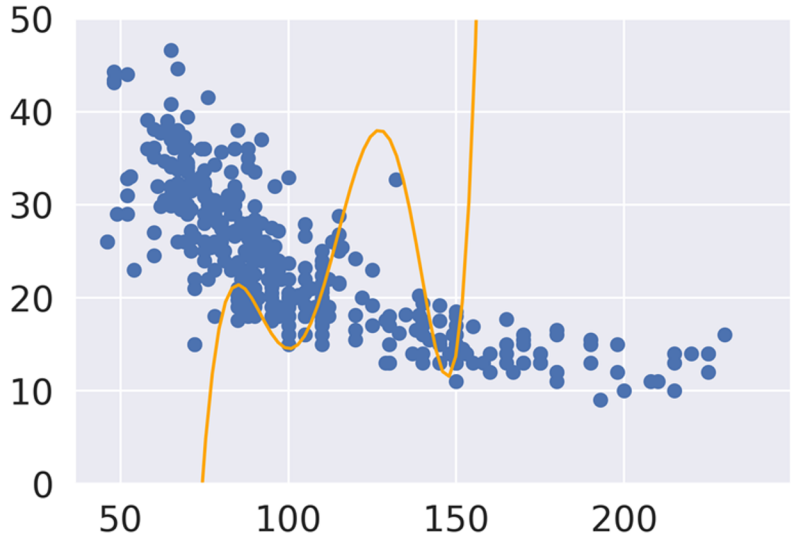


Comparing a Fit On Our Six Data Points with the Full Data Set

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!





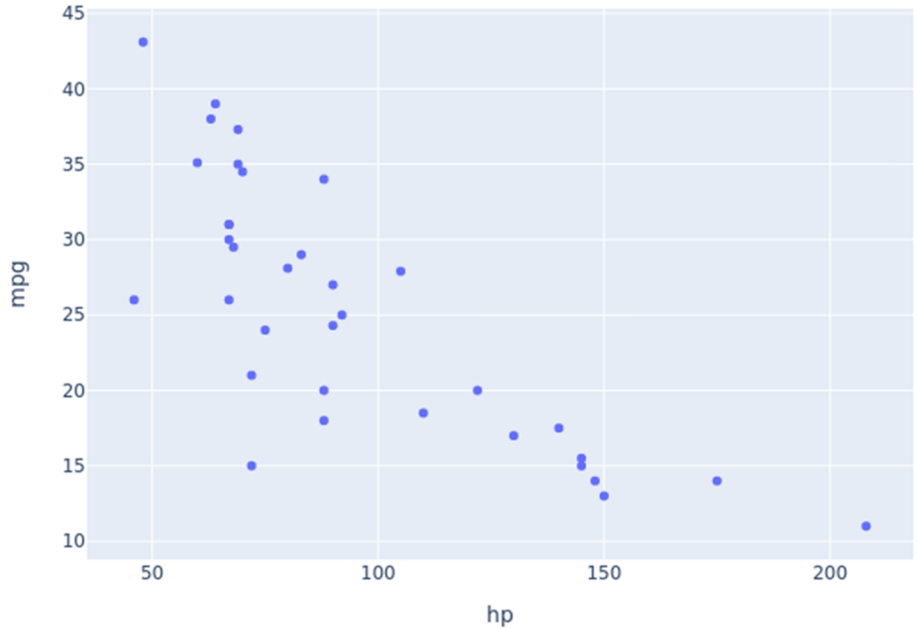
Detecting Overfitting

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

Our 35 Samples

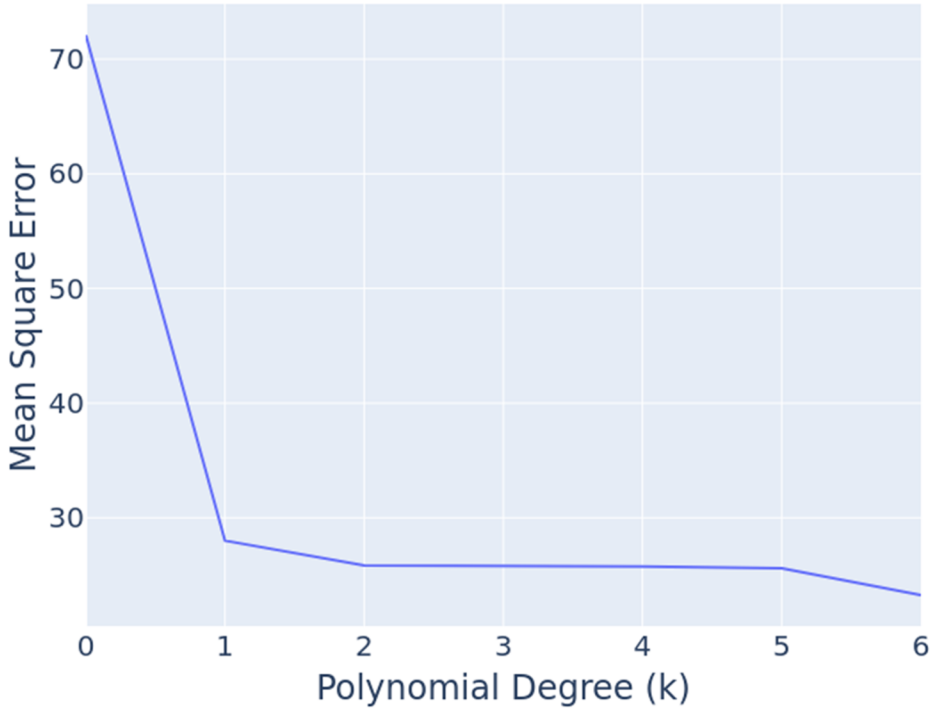
Consider a model fit on only the 35 data points.

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



Fitting Various Degree Models

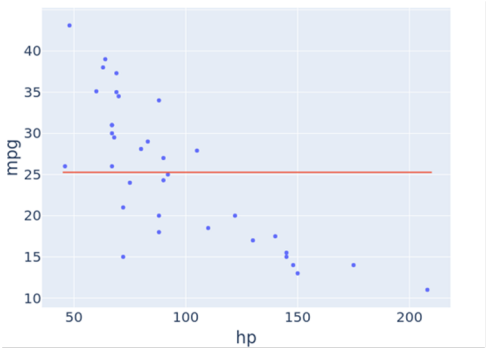
If we fit models of degree 0 through 7 of this model. The MSE is as shown below.

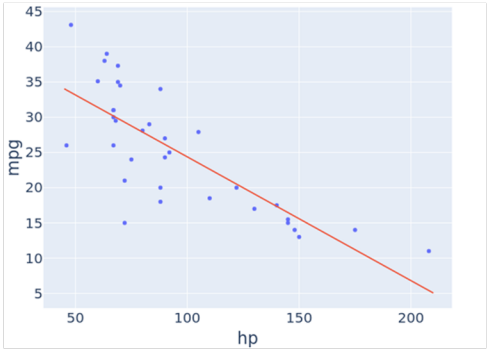


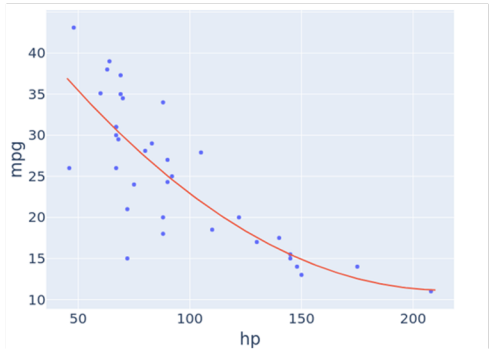
k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001

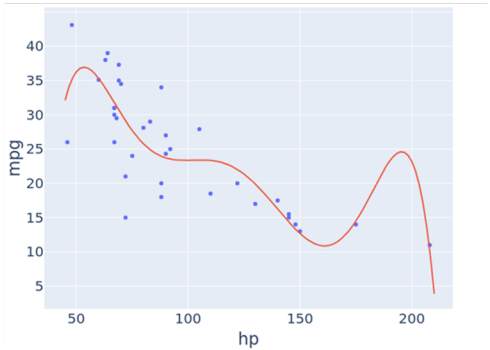
Visualizing the Models

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







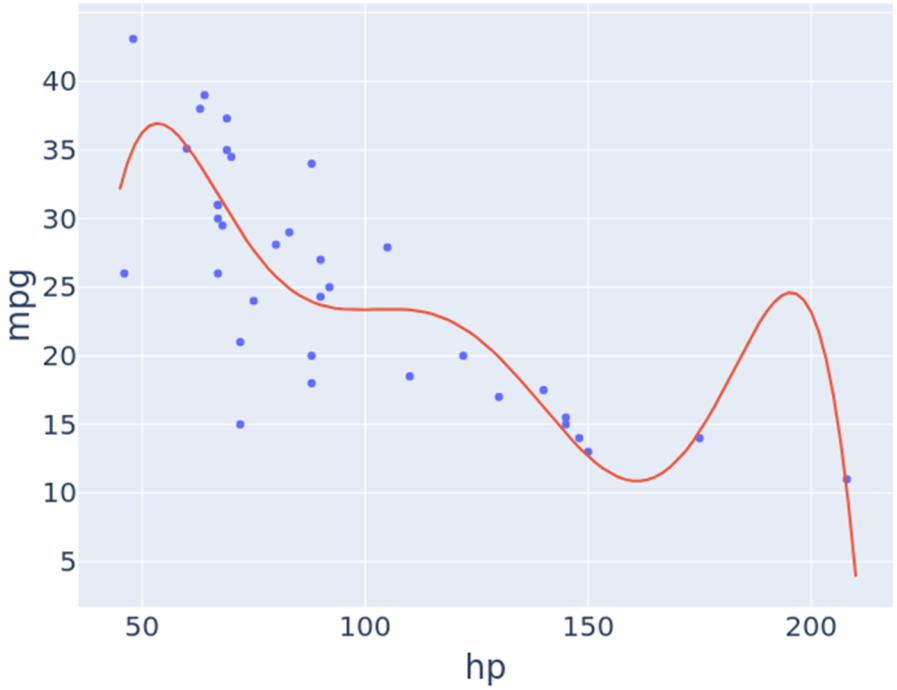


k	MSE
0	72.091396
1	28.002727
2	25.835769
3	25.831592
4	25.763052
5	25.609403
6	23.269001

An Intuitively Overfit Model

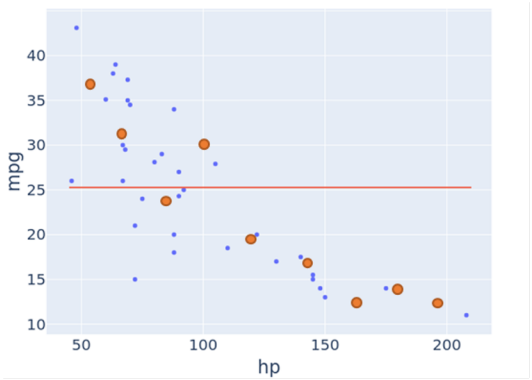
Intuitively, the degree 6 model below feels like it is overfit.

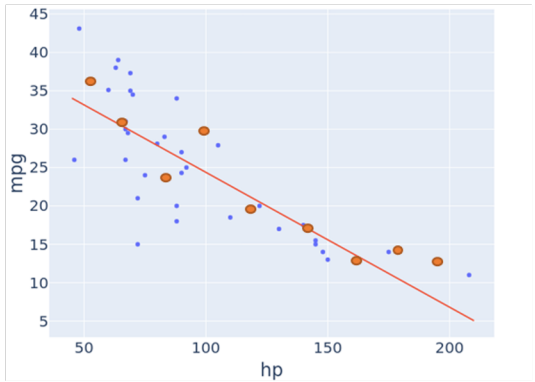
- More specifically: It seems that if we collect more data, i.e. draw more samples from the same distribution, we are worried this model will make poor predictions.

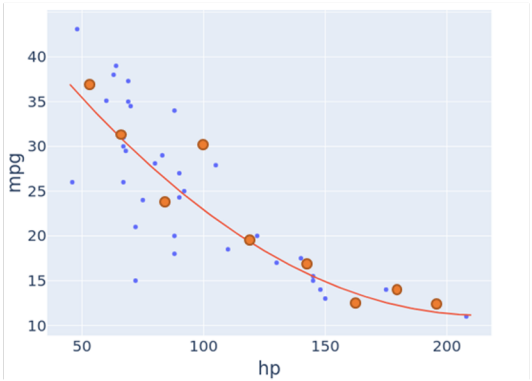


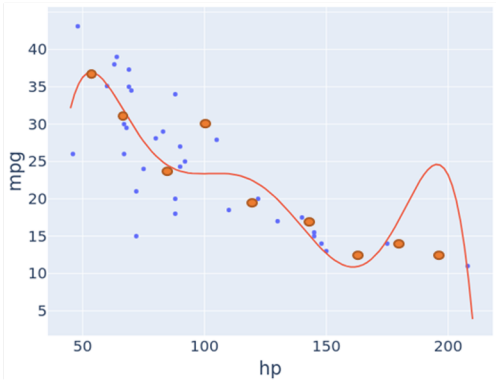
Collecting More Data to Prove a Model is Overfit

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.