LECTURE 16

Feature Engineering

Transforming data to improve model performance.

Data Science, Spring 2024@ Knowledge Stream

Sana Jabbar

sklearn

Lecture 16

- sklearn
- Feature Engineering
- One-Hot Encoding
- Polynomial Features
- Complexity and Overfitting

sklearn: a Standard Library for Model Creation

We have done “heavy lifting” of model creation ourselves – via calculus, ordinary least squares, or gradient descent.

$$\hat{\theta}_0 = \bar{y} - \hat{\theta}_1 \bar{x}$$

$$\hat{\theta}_1 = r \frac{\sigma_y}{\sigma_x}$$

sklearn: a Standard Library for Model Creation

We have done "heavy lifting" of model creation ourselves – via calculus, ordinary least squares, or gradient descent

$$\hat{\theta}_0 = \bar{y} - \hat{\theta}_1 \bar{x}$$

$$\hat{\theta}_1 = r \frac{\sigma_y}{\sigma_x}$$

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

`sklearn`: a Standard Library for Model Creation

We have done “heavy lifting” of model creation ourselves – via calculus, ordinary least squares, or gradient descent

$$\hat{\theta}_0 = \bar{y} - \hat{\theta}_1 \bar{x}$$

$$\hat{\theta}_1 = r \frac{\sigma_y}{\sigma_x}$$

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

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

sklearn: a Standard Library for Model Creation

We have done “heavy lifting” of model creation ourselves – via calculus, ordinary least squares, or gradient descent

sklearn uses an object-oriented programming. Different types of models are defined as their own classes. To use a model, we initialize an instance of the model class. .

$$\hat{\theta}_0 = \bar{y} - \hat{\theta}_1 \bar{x}$$

$$\hat{\theta}_1 = r \frac{\sigma_y}{\sigma_x}$$

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

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

sklearn: a Standard Library for Model Creation

In Data science, we will use Scikit-Learn, commonly called **sklearn**

```
import sklearn
my_model = linear_model.LinearRegression()
my_model.fit(X, y)
my_model.predict(X)
```

The `sklearn` Workflow

At a high level, there are three steps to creating an **`sklearn`** model:

1

Initialize a new model instance

Make a “copy” of the model template

2

Fit the model to the training data

Save the optimal model parameters

3

Use fitted model to make predictions

Fitted model outputs predictions for y

The `sklearn` Workflow

At a high level, there are three steps to creating an **`sklearn`** model:

1	Initialize a new model instance *Make a "copy" of the model template*	`my_model = lm.LinearRegression()`
2	Fit the model to the training data *Save the optimal model parameters*	`my_model.fit(X, y)`
3	Use fitted model to make predictions *Fitted model makes predictions for y*	`my_model.predict(X)`

To extract the fitted parameters: `my_model.coef_` and `my_model.intercept_`

Feature Engineering

Lecture 16

- sklearn
- Feature Engineering
- One-Hot Encoding
- Polynomial Features
- Complexity and Overfitting

Feature Engineering

- **Features in machine learning**
  - Building blocks
  - input variables
- **Types of features**
- **Numerical Features**
  - Continuous values that can be measured on a scale.
  - Examples: age, height, weight, and income
  - Can be use directly
- **Categorical Features**
  - Discrete values can be grouped into categories
  - Examples: gender, color, and zip code
  - Need to be converted to numerical
  - Using one-hot, label, and ordinal encoding

Feature Engineering

- **Time-series Features-**
  - These features are measurement taken over time
  - Examples: stock prices, weather data, and sensor reading
  - You can use these features to train machine learning models that can predict future values or identify patterns in the data.
- **Text Features**
  - These features are text strings represent words, phrases, or sentences
  - Examples of text features include product reviews, social media posts, and medical records
  - You can use text features to train machine learning models that can understand the meaning of text or classify text into different categories

Feature Engineering

- Feature engineering in ML contains mainly four processes:
    1. **Feature Creation** refers to the creation of new features from existing data to help with better predictions

       Examples: one-hot-encoding, binning, splitting, and calculated features.
    2. **Transformations** include steps for replacing missing features or features that are not valid.

       Examples: Binning numeric variables into categories, creating domain-specific features.

Feature Engineering

3. **Feature Extraction** involves reducing the amount of data to be processed using dimensionality reduction techniques.

   Techniques: Principal Components Analysis (PCA), Independent Component Analysis (ICA), Linear Discriminant Analysis (LDA).

4. **Feature Selection** is the process of selecting a subset of extracted features.

   This is the subset that is relevant and contributes to minimizing the error rate of a trained model.

   Feature importance score and correlation matrix can be factors in selecting the most relevant features for model training

https://www.javatpoint.com/feature-engineering-for-machine-learning

Feature Engineering

Feature engineering is the process of transforming raw features into more informative features for use in modeling

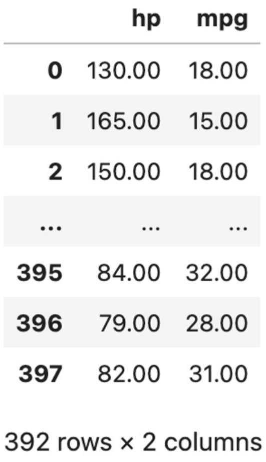
Allows us to:

- Capture domain knowledge
- Use non-numeric features in models

Feature Functions

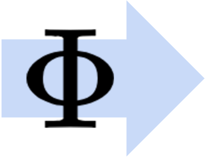
A **feature function** describes the transformations we apply to raw features in the dataset to create transformed features. Often, the dimension of the *featurized* dataset increases.

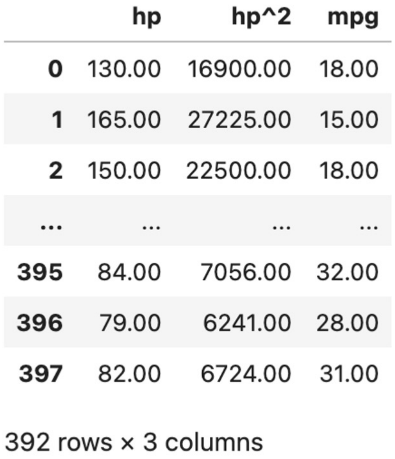
Example: a feature function that adds a squared feature to the design matrix



	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

Dataset of raw features:

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

After applying the feature function Φ:

$$\Phi(\mathbb{X}) \in \mathbb{R}^{n \times p'}$$

Feature Functions

A **feature function** describes the transformations we apply to raw features in the dataset to create transformed features. Often, the dimension of the *featurized* dataset increases.

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

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

$$\hat{y} = \theta_0 + \theta_1 \phi_1 + \theta_2 \phi_2$$
$$\hat{\mathbb{Y}} = \Phi\theta$$

Shorthand for “the design matrix after feature engineering”

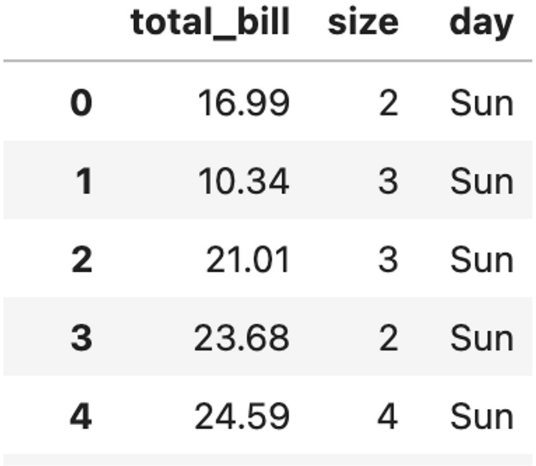
- Implementing Models in Code
- `sklearn`
- Feature Engineering
- **One-Hot Encoding**
- Polynomial Features
- Complexity and Overfitting

One-Hot Encoding

Lecture 16

Regression Using Non-Numeric Features

In `tips` dataset we used when first exploring regression



	total_bill	size	day
0	16.99	2	Sun
1	10.34	3	Sun
2	21.01	3	Sun
3	23.68	2	Sun
4	24.59	4	Sun

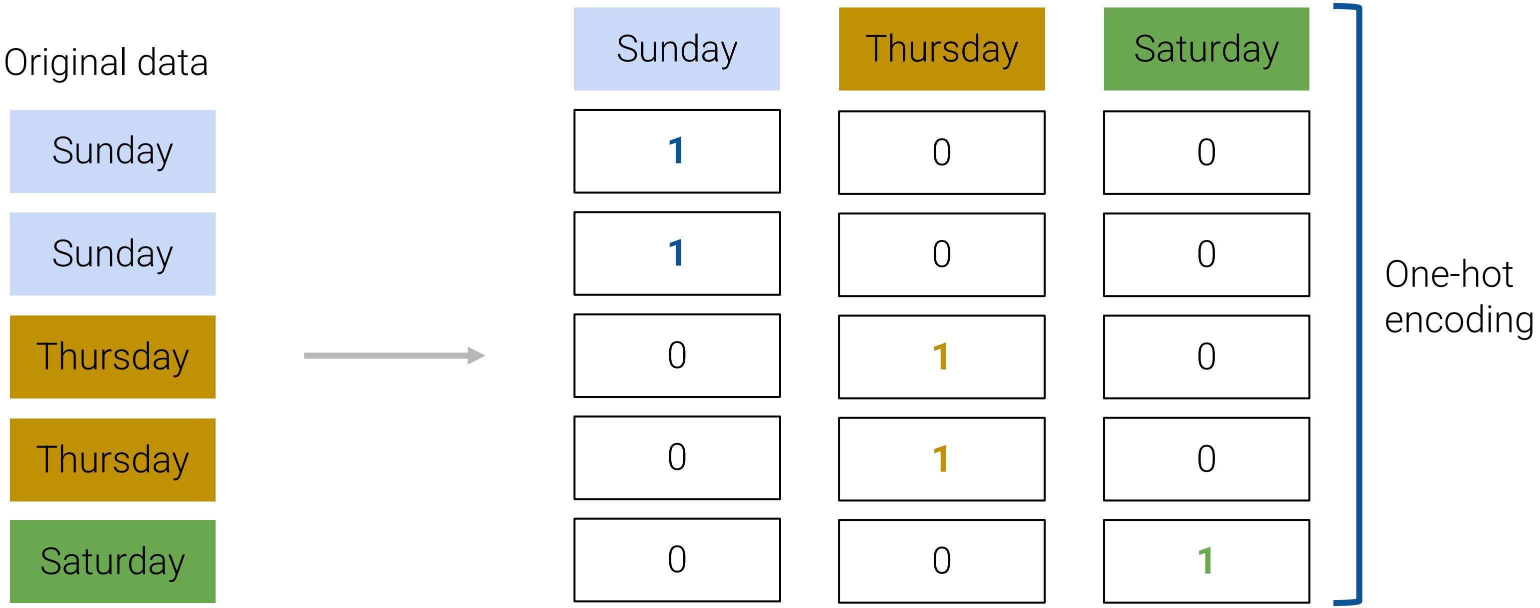
Before, we were limited to only using numeric features in a model – `total_bill` and `size`

By performing feature engineering, we can incorporate *non-numeric* features like the day of the week

One-hot Encoding

One-hot encoding is a feature engineering technique to transform non-numeric data into numeric features for modeling

- Each category of a categorical variable gets its own feature
  - Value = 1 if a row belongs to the category
  - Value = 0 otherwise



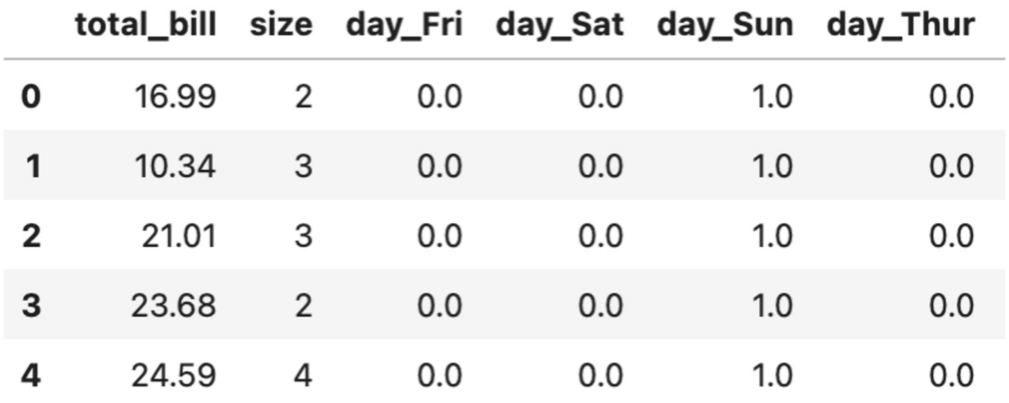
Original data
Sunday
Sunday
Thursday
Thursday
Saturday

Sunday	Thursday	Saturday
1	0	0
1	0	0
0	1	0
0	1	0
0	0	1

One-hot encoding

Regression Using the One-Hot Encoding

The one-hot encoded features can then be used in the design matrix to train a model



	total_bill	size	day_Fri	day_Sat	day_Sun	day_Thur
0	16.99	2	0.0	0.0	1.0	0.0
1	10.34	3	0.0	0.0	1.0	0.0
2	21.01	3	0.0	0.0	1.0	0.0
3	23.68	2	0.0	0.0	1.0	0.0
4	24.59	4	0.0	0.0	1.0	0.0

Raw features

One-hot encoded features

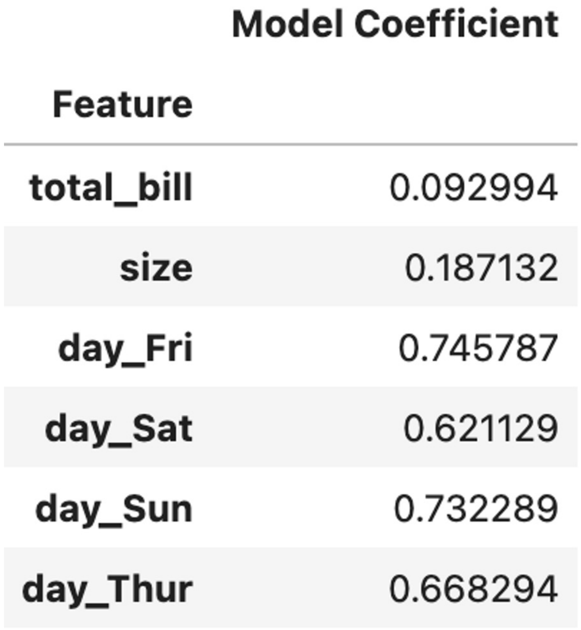
$$\hat{y} = \theta_1(\text{total\_bill}) + \theta_2(\text{size}) + \theta_3(\text{day\_Fri}) + \theta_4(\text{day\_Sat}) + \theta_5(\text{day\_Sun}) + \theta_6(\text{day\_Thur})$$

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

Regression Using the One-Hot Encoding

Using `sklearn` to fit the new model:

$$\hat{y} = \theta_1(\text{total\_bill}) + \theta_2(\text{size}) + \theta_3(\text{day\_Fri}) + \theta_4(\text{day\_Sat}) + \theta_5(\text{day\_Sun}) + \theta_6(\text{day\_Thur})$$



	Model Coefficient
Feature	
total_bill	0.092994
size	0.187132
day_Fri	0.745787
day_Sat	0.621129
day_Sun	0.732289
day_Thur	0.668294

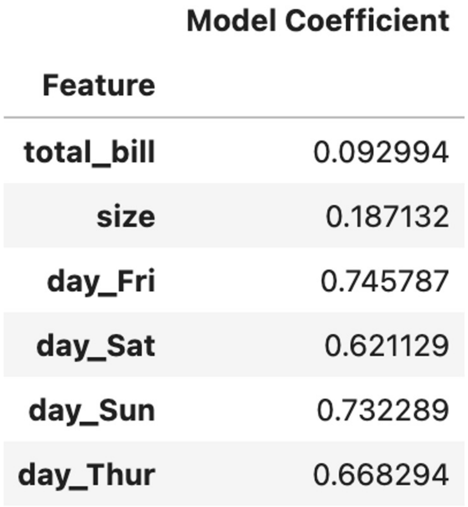
Interpretation: how much the fact that it is Friday impacts the predicted tip

Regression Using the One-Hot Encoding

Party of 3, $50 total bill, eating on a Friday:

$$\hat{y} = \theta_1(\text{total\_bill}) + \theta_2(\text{size}) + \theta_3(\text{day\_Fri}) + \theta_4(\text{day\_Sat}) + \theta_5(\text{day\_Sun}) + \theta_6(\text{day\_Thur})$$

$$\hat{y} = 0.092994(50) + 0.187132(3) + 0.745787(1) + 0.621129(0) + 0.732289(0) + 0.668294(0)$$

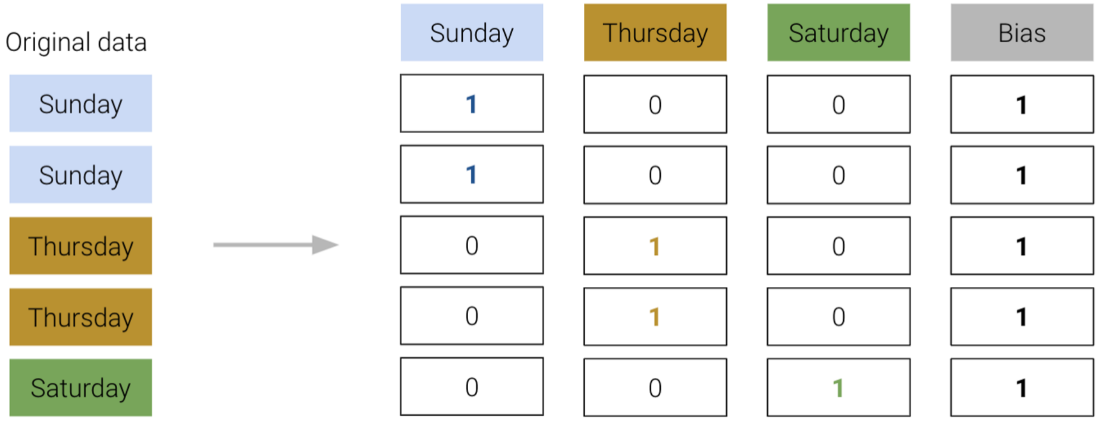


	Model Coefficient
Feature	
total_bill	0.092994
size	0.187132
day_Fri	0.745787
day_Sat	0.621129
day_Sun	0.732289
day_Thur	0.668294

$$\hat{y} = 5.9568643$$

One-hot Encode Wisely!

Any set of one-hot encoded columns will always sum to a column of all ones.



Original data

Original data
Sunday
Sunday
Thursday
Thursday
Saturday

Sunday	Thursday	Saturday	Bias
1	0	0	1
1	0	0	1
0	1	0	1
0	1	0	1
0	0	1	1

The bias column is a linear combination of the OHE columns

If we also include a bias column in the design matrix, there will be linear dependence in the model. $\mathbb{X}^\top \mathbb{X}$ is not invertible, and our OLS estimate $\hat{\theta} = (\mathbb{X}^\top \mathbb{X})^{-1} \mathbb{X}^\top \mathbb{Y}$ fails.

How to resolve? Omit one of the one-hot encoded columns *or* do not include an intercept term

One-hot Encode Wisely!

How to resolve? Omit one of the one-hot encoded columns *or* do not include an intercept term

Adjusted design matrices:

Sunday	Thursday	~~Saturday~~	Bias
1	0	~~0~~	1
1	0	~~0~~	1
0	1	~~0~~	1
0	1	~~0~~	1
0	0	~~1~~	1

or

Sunday	Thursday	Saturday	~~Bias~~
1	0	0	~~1~~
1	0	0	~~1~~
0	1	0	~~1~~
0	1	0	~~1~~
0	0	1	~~1~~

We still retain the same information – in both approaches, the omitted column is simply a linear combination of the remaining columns

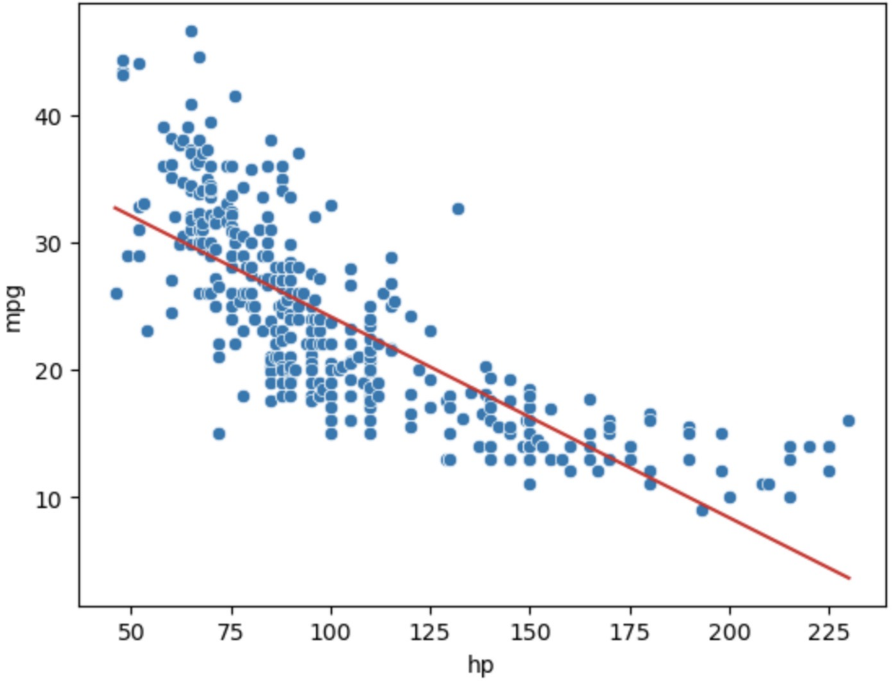
Polynomial Features

Lecture 16

- `sklearn`
- Feature Engineering
- One-Hot Encoding
- **Polynomial Features**
- Complexity and Overfitting

Accounting for Curvature

We've seen a few cases now where models with linear features have performed poorly on datasets with a clear non-linear curve.



$$\hat{y} = \theta_0 + \theta_1(\text{hp})$$

MSE: 23.94

When our model uses only a single linear feature (**hp**), it cannot capture non-linearity in the relationship

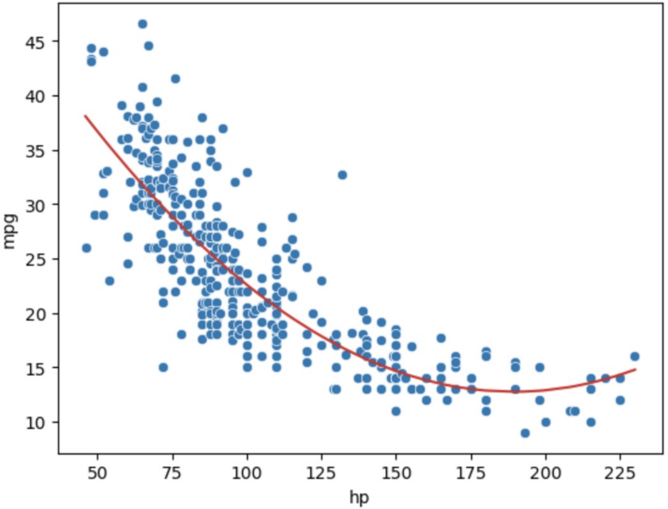
Solution: incorporate a non-linear feature!

Polynomial Features

We create a new feature: the square of the **hp**

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2\left(\text{hp}^2\right)$$

This is still a **linear model**. Even though there are non-linear *features*, the model is linear with respect to θ

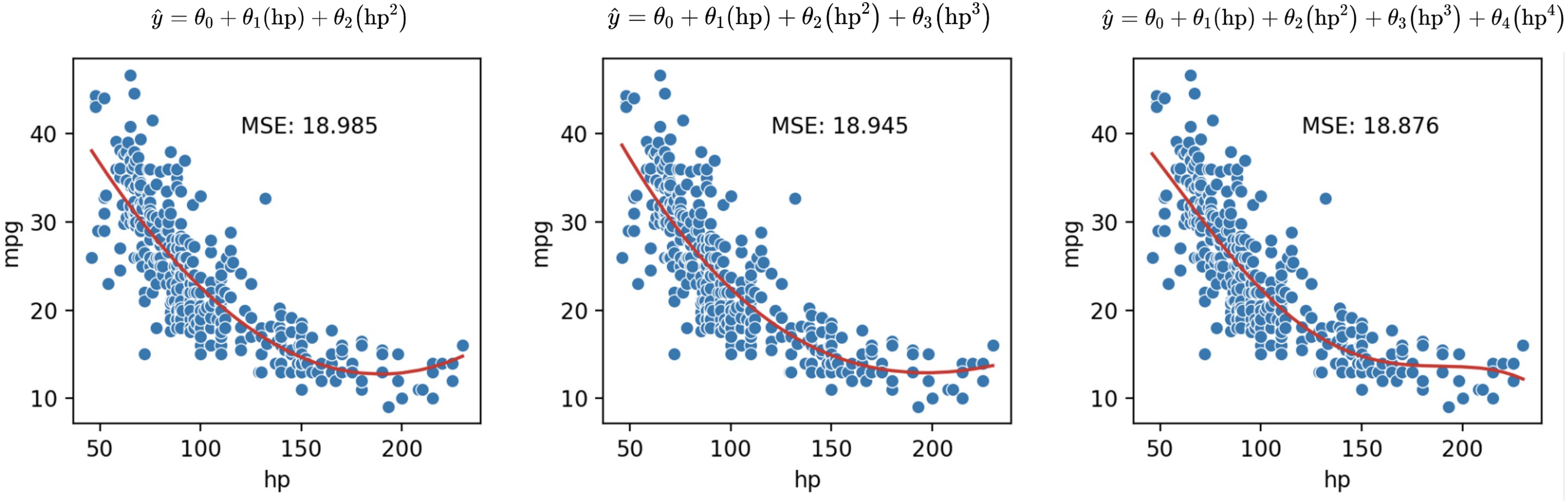


Degree of model: 2
MSE: 18.98

Looking a lot better: our predictions capture the curvature of the data.

Polynomial Features

What if we add more polynomial features?



$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2\left(\text{hp}^2\right)$$

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2\left(\text{hp}^2\right) + \theta_3\left(\text{hp}^3\right)$$

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2\left(\text{hp}^2\right) + \theta_3\left(\text{hp}^3\right) + \theta_4\left(\text{hp}^4\right)$$

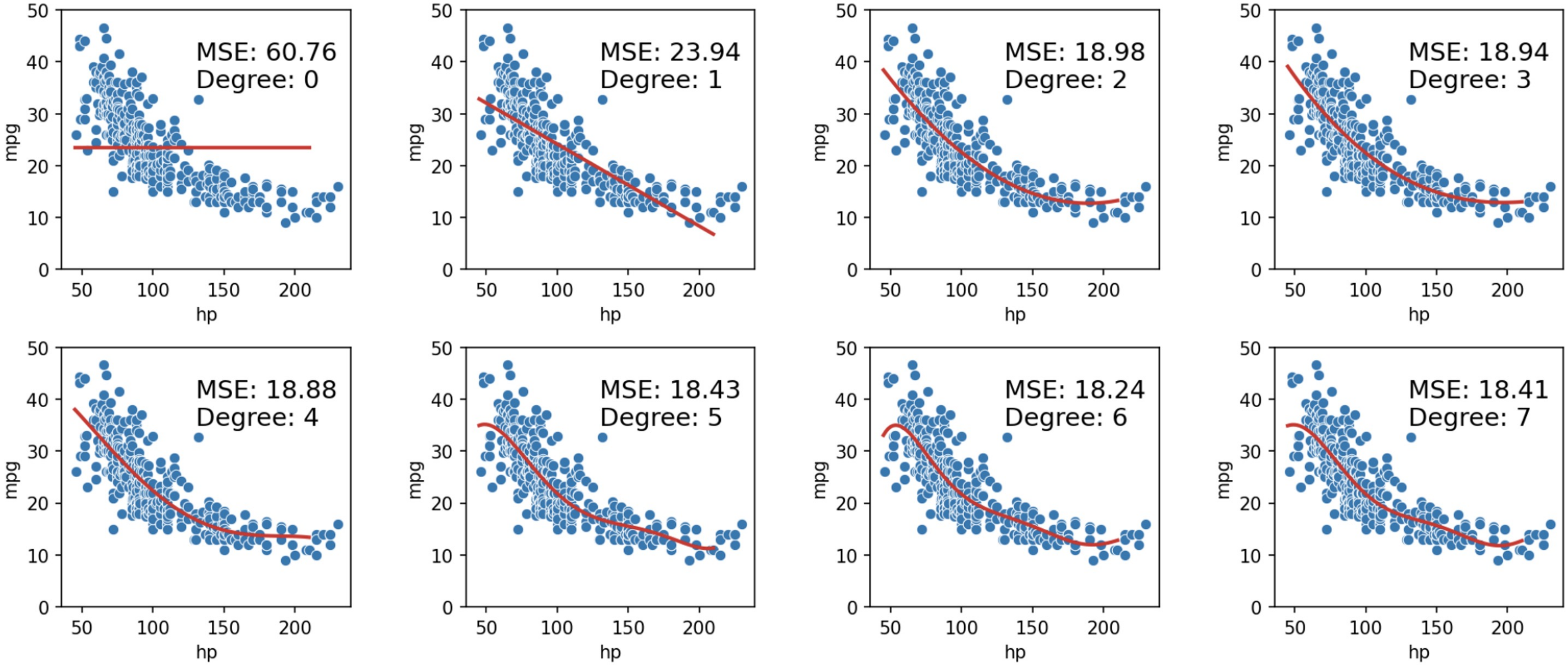
MSE continues to decrease with each additional polynomial term

Complexity and Overfitting

Lecture 16

- `sklearn`
- Feature Engineering
- One-Hot Encoding
- Polynomial Features
- **Complexity and Overfitting**

How Far Can We Take This?

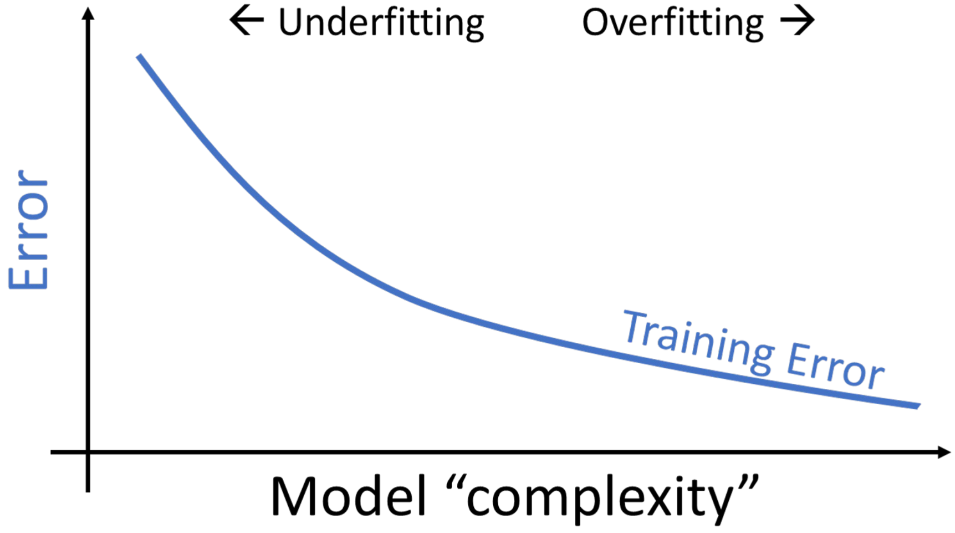


Model Complexity

As we continue to add more and more polynomial features, the MSE continues to decrease

Equivalently: as the **model complexity** increases, its *training error* decreases

General trend for an arbitrary dataset

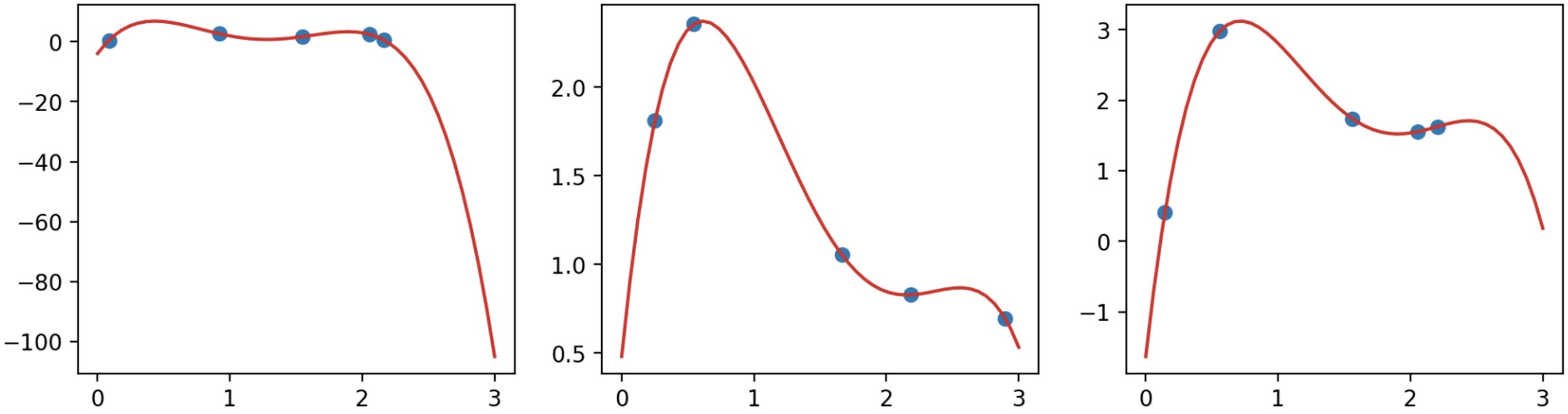


Seems like a good deal?

An Extreme Example: Perfect Polynomial Fits

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

For example, there always exists a degree-4 polynomial curve that can perfectly model a dataset of 5 datapoints

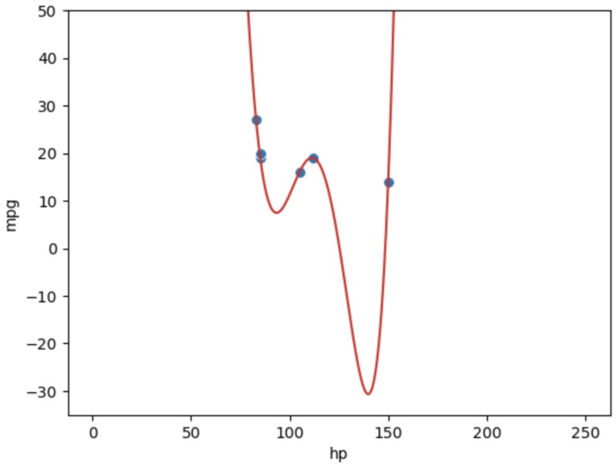


Model Performance on Unseen Data

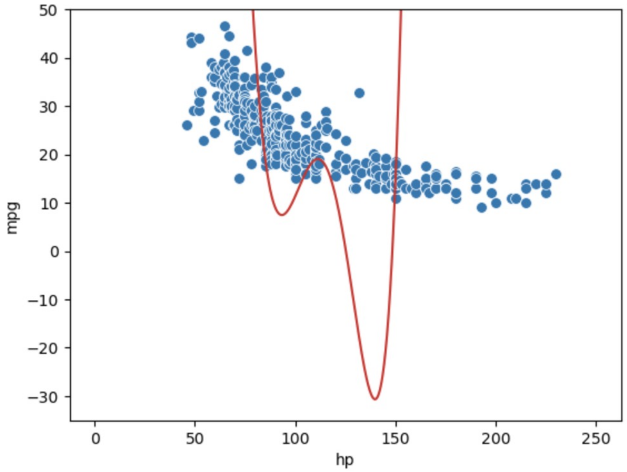
New (more realistic) example:

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

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



Complex model makes perfect predictions on the training data…



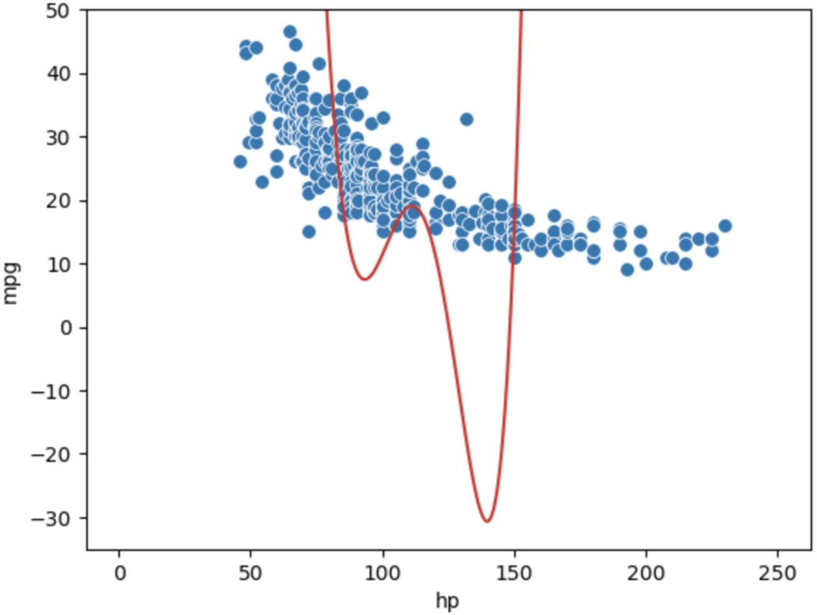
…but performs *horribly* on the rest of the population!

Model Performance on Unseen Data

What went wrong?

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

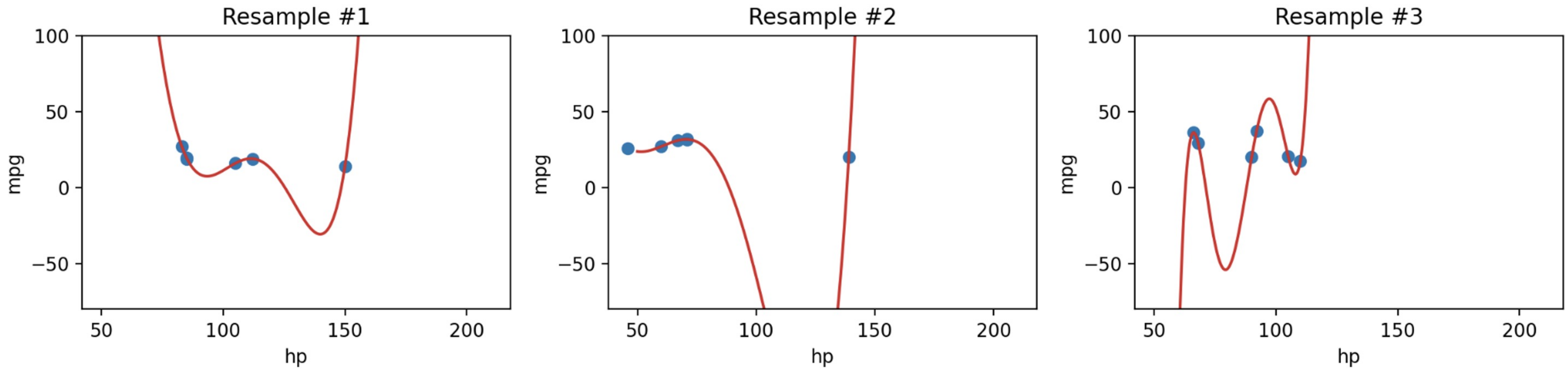
This is a problem: we want models that are generalizable to "unseen" data



Model Variance

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

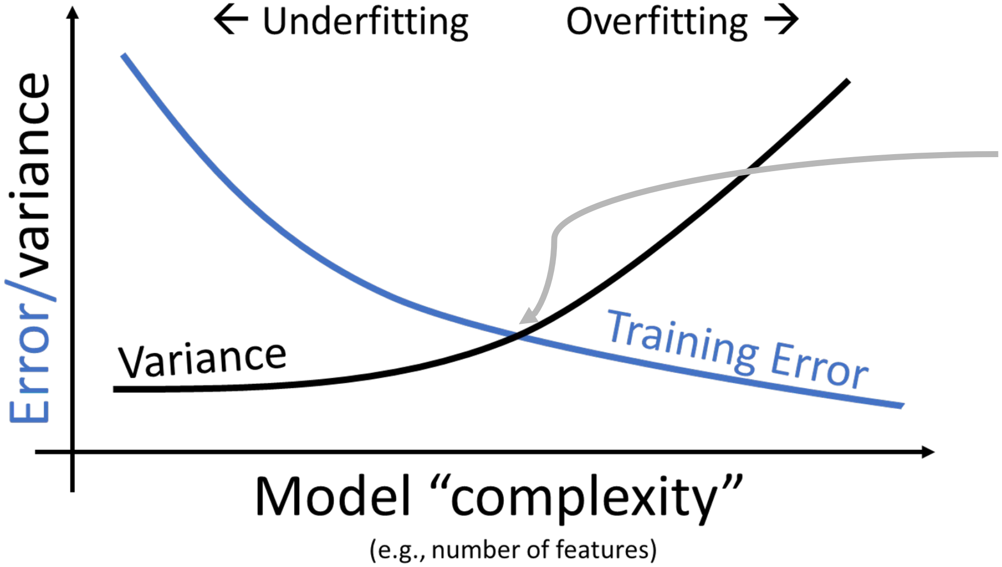
Our degree-5 model varies erratically when we fit it to different samples of 6 points from `vehicles`



Error, Variance, and Complexity

We face a dilemma:

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



Our goal: find this "sweet spot"

Stay tuned for future lectures covering this!