

Linear Models

The Academy of AI 2018/19

Session 3

Students for AI

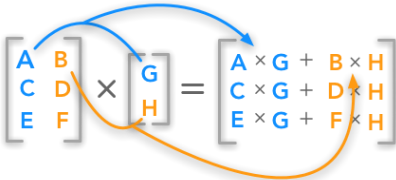
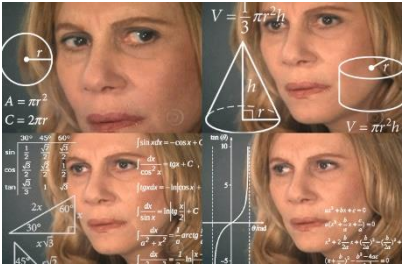
Students for AI present the twelve workshop series:

1. Introduction to Python
2. Scientific Python
- 3. Linear Models in Data Science**
4. Logistic Regression
5. Introduction to Databases (SQL)
6. Traditional Approaches to ML
7. Deep Learning in PyTorch
8. Datasets and Transforms
9. Regularisation for Deep Learning
10. Convolutional Neural Networks
11. RNNs and LSTM
12. Generative Models

By the end of this lecture you should:

- Know what a linear model looks like...
- ... and how it works!
- Know when to use a linear model
- Create your first linear model through machine learning (if you haven't before)
- Know how to validate your model

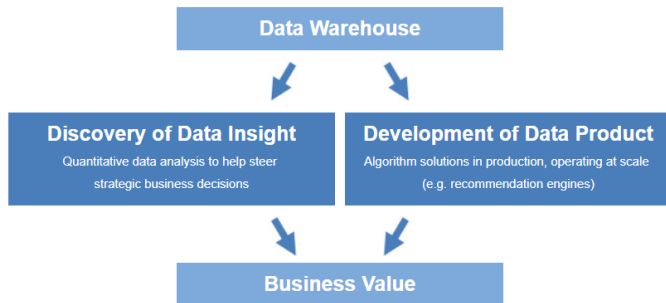
Prerequisite Knowledge



1. Introduction to Data Science
2. Linear Regression and How it Works
3. Practice!
4. Multivariate Linear Regression
5. Practice!
6. Homework

Introduction to Data Science

Data science is about using data in creative ways to generate business value



Uncovering findings, understanding complex behaviors, trends, and inferences to enable companies to make smarter business decisions.

For example:

- Netflix data mines movie viewing patterns to understand what drives user interest
- Target identifies what are major customer segments within it's base and the unique shopping behaviors within those segments
- Proctor & Gamble utilizes time series models to more clearly understand future demand.

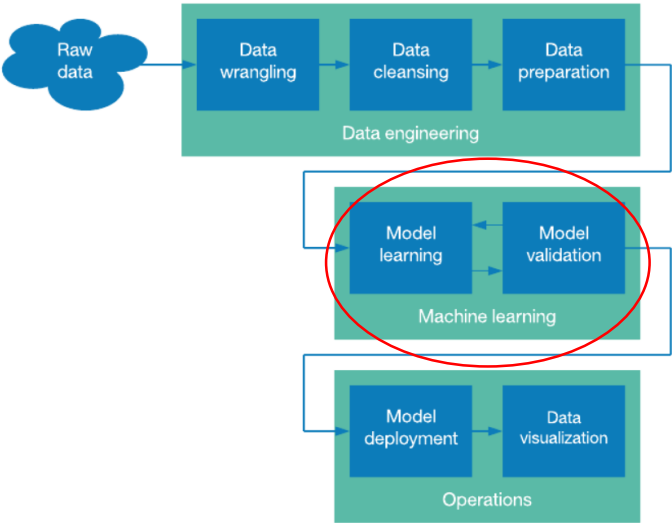
A "data product" is a technical asset that:

1. utilizes data as input, and
2. processes that data to return algorithmically-generated results.

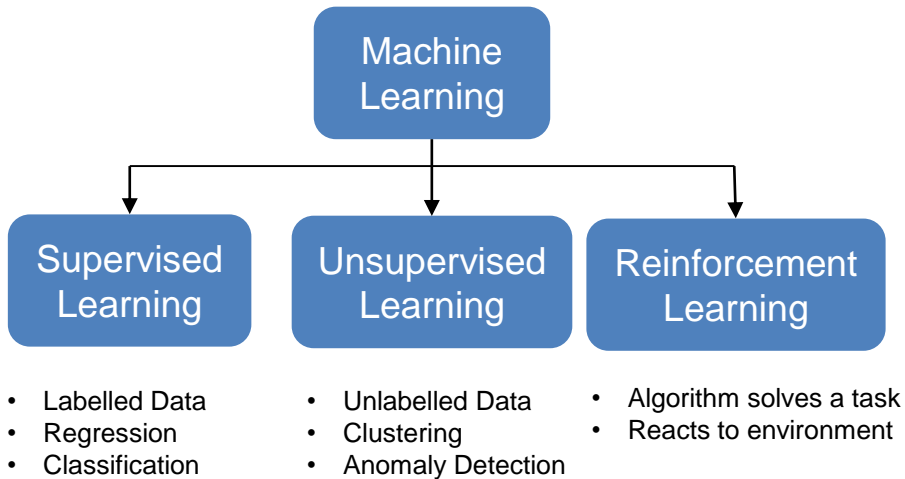
For example:

- Amazon's recommendation engines suggest items for you to buy, determined by their algorithms.
- Gmail's spam filter is an algorithm behind the scenes processes incoming mail and determines if a message is junk or not.
- Computer vision used for self-driving cars is uses machine learning algorithms that are able to recognize traffic lights, other cars on the road, pedestrians, etc.

Data Science Pipeline



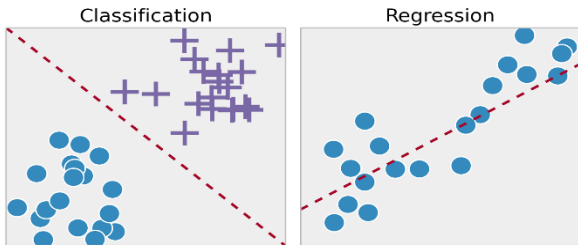
Types of machine learning:




Supervised Learning

Supervised learning regroups different techniques which all share the same principles:

- The training dataset contains inputs data (your predictors) and the value you want to predict (labels)
- The model will use the training data to learn a link between the input and the outputs. Underlying idea is that the training data can be generalized and that the model can be used on new data with some accuracy.

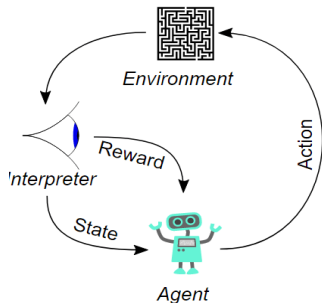


Unsupervised learning **does not** use labelled output data.
Unsupervised algorithms can be split into different categories:

- **Clustering algorithm**, such as K-means, hierarchical clustering or mixture models. These algorithms try to discriminate and separate the observations in different groups. 
- **Dimensionality reduction** algorithms such as PCA, ICA or autoencoder. These algorithms find the best representation of the data with fewer dimensions.
- **Anomaly detections** to find outliers in the data, i.e. observations which do not follow the data set patterns.

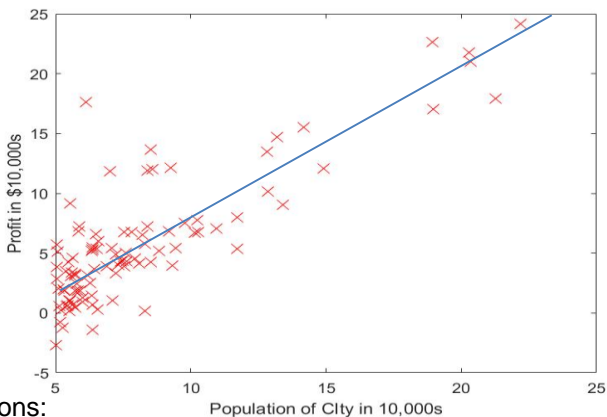
Most of the time unsupervised learning algorithms are used to pre-process the data, during the exploratory analysis or to pre-train supervised learning algorithms.

Reinforcement learning algorithms try to find the best ways to earn the greatest reward. Reinforcement learning algorithms follow the different circular steps:



- Given its and the environment's states, the agent will choose the action which will maximize its reward or will explore a new possibility.
- These actions will change the environment's and the agent states.
- They will also be interpreted to give a reward to the agent.
- By performing this loop many times, the agents will improve its behavior.

Supervised Learning – Linear Regression



Applications:

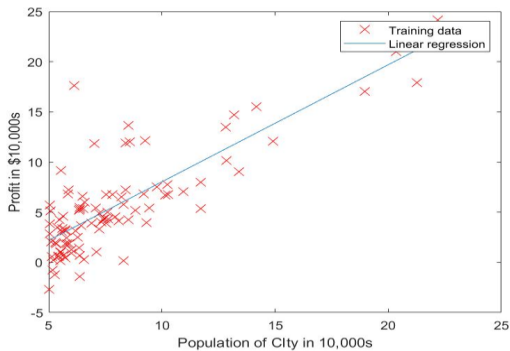
- Studying engine data from test performance
- Market research studies and customer survey analysis
- Causal relationships in biological systems
- And many more...

Linear Regression and How it Works

How do we do linear regression in python?

```
# Create linear regression object  
regr = linear_model.LinearRegression()  
  
# Train the model using the training sets  
regr.fit(diabetes_X_train, diabetes_y_train)
```

What is Linear Regression?

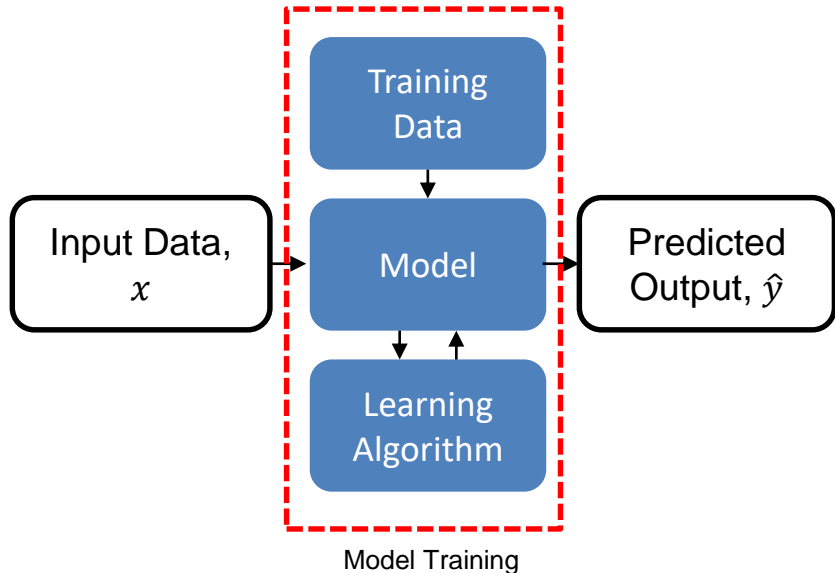


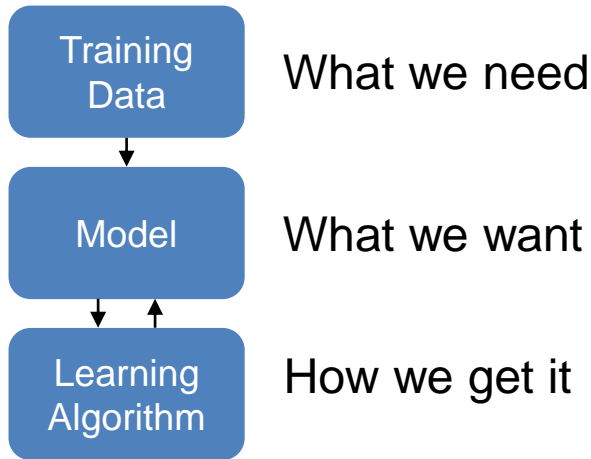
Supervised Learning

Data has an expected output for every input (y for every x)

Regression Problem

Predict the output for a given input





Notation:

- m = Number of training examples
- $x^{(i)}$ = input variables/**features**/predictors
- $y^{(i)}$ = output/target variable/**labels**

| $x^{(i)}$ | $y^{(i)}$ |
|-----------|-----------|
| $x^{(1)}$ | $y^{(1)}$ |
| $x^{(2)}$ | $y^{(2)}$ |
| . | . |
| . | . |
| $x^{(m)}$ | $y^{(m)}$ |

Provided a set S of m $(x^{(i)}, y^{(i)})$ pairs:

$$S = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

Find a function f that maps $x^{(i)} \rightarrow y^{(i)}$:

$$f(x^{(i)}) \rightarrow y^{(i)}$$

f is a:

- **(Discriminate) Model**
- **Map Function**
- **Function Approximation**

for S , such that:


$$f(x^{(i)}) = \widehat{y^{(i)}} \approx y^{(i)}, \quad \forall i$$

Linear Regression Model with One Variable

Equation of a line :

$$\begin{aligned}f(x^{(i)}) &= w_0 + w_1 * x^{(i)} \\ \widehat{y^{(i)}} &= w_0 + w_1 * x^{(i)}\end{aligned}$$

Bias = 1


$$f(x^{(i)}) = w_0 * x_0^{(1)} + w_1 * x_1^{(1)}$$

How do we find our weights?
(w_0 and w_1)

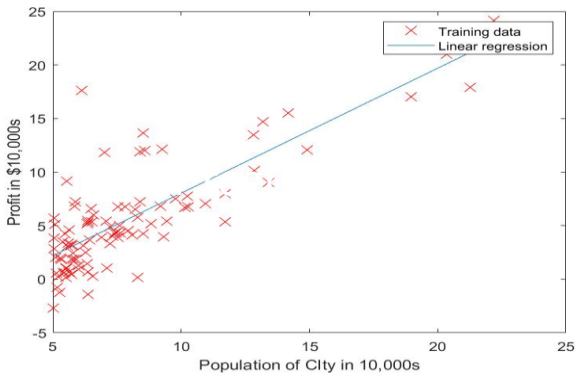
Objective:

- Have some cost function, $J(w_0, w_1)$ that determines how bad a model is performing/how much error between the model and observed data
- Minimize the cost function, $J(w_0, w_1)$

Outline:

- Start with some weights w_0, w_1 (can be anything!)
- Keep changing w_0, w_1 to reduce $J(w_0, w_1)$ until we end up at a minimum

Cost Function



Use **Mean Squared Error (MSE)** as the cost function (denoted with $J(w_0, w_1)$ in order to evaluate how good/bad the model f is, such that:

$$MSE = J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)})^2$$

Where $\widehat{y^{(i)}} = w_0 + w_1 * x^{(i)}$

Model:

$$\widehat{y^{(i)}} = w_0 + w_1 * x^{(i)}$$

Weights:

$$w_0, w_1$$

Cost Function:

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)})^2$$

Goal:

Adjust weights to minimize cost function

$$w_0 \ w_1$$

$$J(w_0, w_1)$$

Gradient Descent Algorithm

Repeat Until Convergence:

$$w_j := w_j - \alpha \frac{\delta}{\delta w_j} J(w_0, w_1)$$

for both $j=0, j=1$

Learning
Rate

Derivative of
Cost Function

Gradient Descent Algorithm:

$$j=0: \quad \frac{\delta}{\delta w_j} J(w_0, w_1) = \frac{1}{m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)})$$

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)})$$

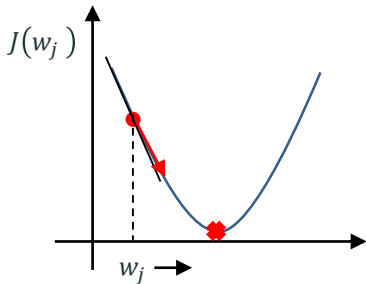
$$j=1: \quad \frac{\delta}{\delta w_j} J(w_0, w_1) = \frac{1}{m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)}) x^{(i)}$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)}) x^{(i)}$$

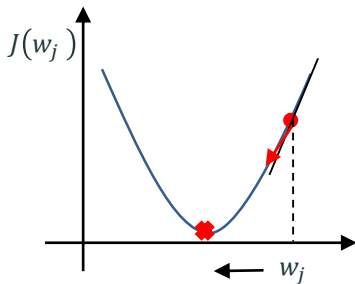
Gradient Descent – Why it works

$$w_j := w_j - \alpha \frac{\delta}{\delta w_j} J(w_0, w_1)$$

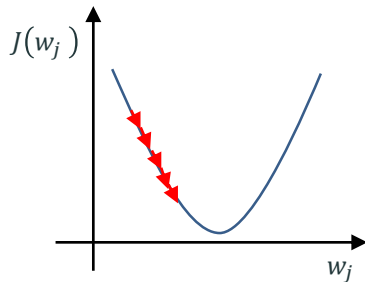
$$\frac{\delta}{\delta w_j} J(w_0, w_1) < 0$$



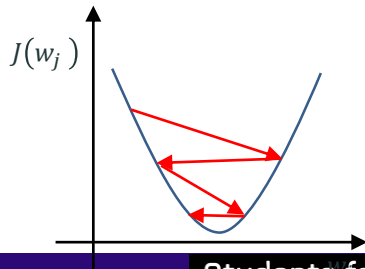
$$\frac{\delta}{\delta w_j} J(w_0, w_1) > 0$$

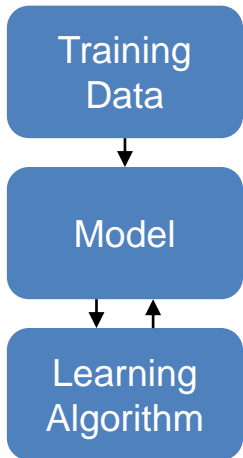


If α is too small, gradient descent is slow



If α is too large, gradient descent can overshoot, and fail to converge.





Step by Step:

- Initialise random weights for model
- Compare predicted output with actual output and calculate cost function for model

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)})^2$$

- Perform gradient descent and adjust weights

$$w_j := w_j - \alpha \frac{\delta}{\delta w_j} J(w_0, w_1)$$

- Plot line of best fit

Practice!

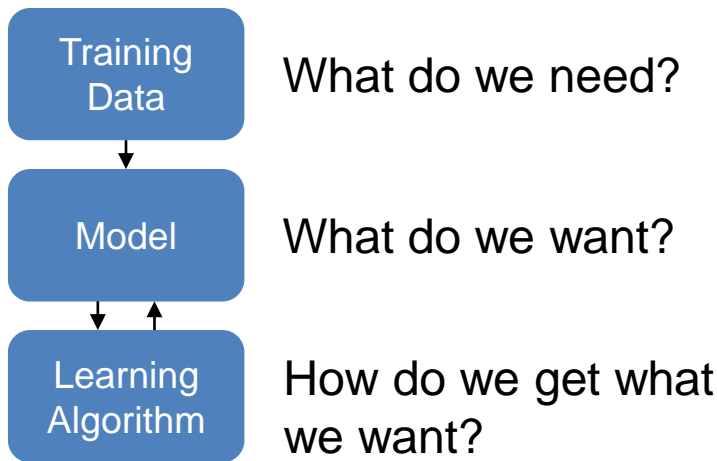
Access this link and download the linear regression demo notebook along with the data folder

<https://tinyurl.com/y7t3qufd>

Multivariate Linear Regression

What if we have more than one feature?

| $x_1^{(i)}$ | $x_2^{(i)}$ | $y^{(i)}$ |
|-------------|-------------|-----------|
| $x_1^{(1)}$ | $x_2^{(1)}$ | $y^{(1)}$ |
| $x_1^{(2)}$ | $x_2^{(2)}$ | $y^{(2)}$ |
| . | . | . |
| . | . | . |
| $x_1^{(m)}$ | $x_2^{(m)}$ | $y^{(m)}$ |



The process is the same!

Data with multiple features/variables

For n number of features and m number of training examples:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ \vdots & & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

$m \times n$ array

$$W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$n \times 1$ array

$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$n \times 1$ array

$x_j^{(i)}$ ← Training Data Number
 ← Feature Number

Multivariate Linear Regression Model

For $n = 1$ (simple linear regression):

$$\widehat{y^{(i)}} = w_0 + w_1 * x^{(i)}$$

For $n > 1$ (multivariate linear regression):

$$\widehat{y^{(i)}} = w_0 * x_0^{(i)} + w_1 * x_1^{(i)} + w_2 * x_2^{(i)} + w_3 * x_3^{(i)} + \dots + w_n * x_n^{(i)}$$

= $X * W$ (Bold letters denote matrix notation!)

$$\hat{\mathbf{y}} = \mathbf{X} * \mathbf{W}$$

For $n = 1$ (simple linear regression):

$$MSE = \frac{1}{2m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)})^2$$

For $n > 1$ (multivariate linear regression):

Exactly the same!

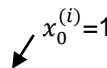
$$MSE = \frac{1}{2m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)})^2$$

In matrix notation,

$$MSE = \frac{1}{2m} (\widehat{\mathbf{y}} - \mathbf{y})^T (\widehat{\mathbf{y}} - \mathbf{y})$$

$$w_j := w_j - \alpha \frac{\delta}{\delta w_j} J(\mathbf{W})$$

j=0: $w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)}) x_0^{(i)}$



j=1: $w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)}) x_1^{(i)}$

j=2: $w_2 := w_2 - \alpha \frac{1}{m} \sum_{i=1}^m (\widehat{y^{(i)}} - y^{(i)}) x_2^{(i)}$

...

OR

$$\mathbf{W} = \mathbf{W} - \alpha \frac{1}{m} (\widehat{\mathbf{y}} - \mathbf{y})^T \mathbf{X}$$

Polynomial Regression is a case of multivariate linear regression

Eg. Quadratic Equation:

$$\hat{y} = w_0 + w_1 * x_1 + w_2 * x_1^2$$

Interactions:

$$\hat{y} = w_0 + w_1 * x_1 + w_2 * x_1 x_2$$

Idea: Make sure the features are on the same scale

Motivation:

- Promotes convergence
- Normalizes the contribution of each feature to the final model

Linear Scaling

Goal: Get every feature into the $-1 \leq x_i \leq 1$ range

Eg. Predicting price of a house

$$x_1 = \text{Size}(0-1000 \text{ sq. metre}) \longrightarrow x_1 = \frac{\text{Size}}{1000}$$

$$x_2 = \text{No. of Bedrooms (1-5)} \longrightarrow x_2 = \frac{\text{No. of Bedrooms}}{5}$$

Mean Normalization

Goal: Make features have approximately 0 mean

$$x_j = \frac{x_j - \mu_j}{\sigma_j}$$

where μ_j is the mean and σ_j is the standard deviation of feature x_j

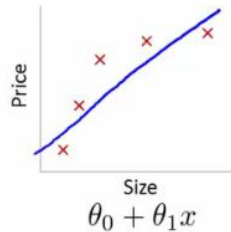
Eg. Predicting price of a house

$$x_1 = \frac{\text{Size} - \mu_1}{\sigma_1}$$

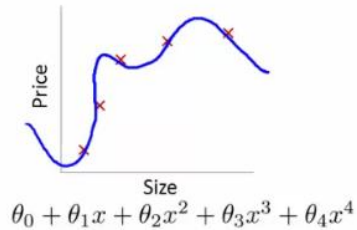
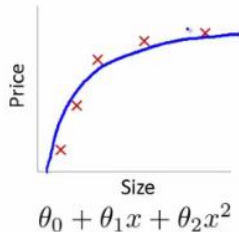
$$x_2 = \frac{\text{No.of Bedrooms} - \mu_2}{\sigma_2}$$

Model Evaluation

Overfitting vs Underfitting

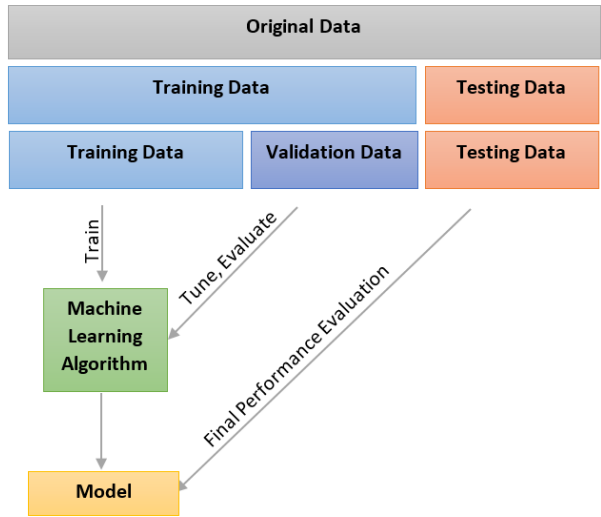


Underfitting



Overfitting

Model Evaluation Data Sets



Training Error:

$$J_{train}(\mathbf{W}) = \frac{1}{2m} \sum_{i=1}^m \left(\widehat{y^{(i)}} - y^{(i)} \right)^2$$

Minimised to perform gradient descent during training

Cross-Validation Error:

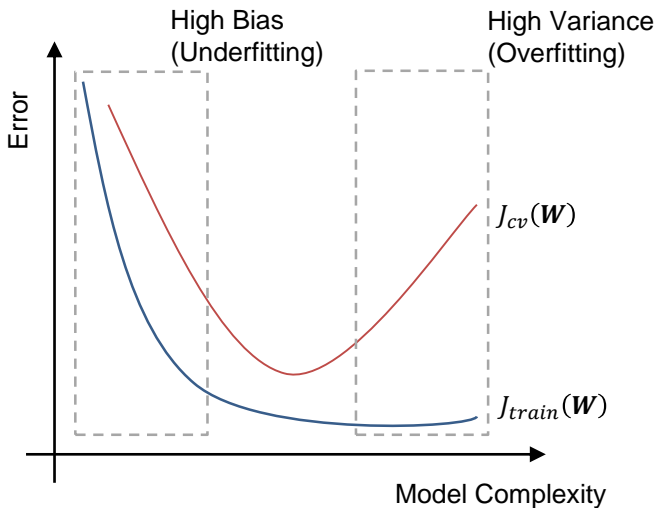
$$J_{cv}(\mathbf{W}) = \frac{1}{2m} \sum_{i=1}^m \left(\widehat{y_{cv}^{(i)}} - y_{cv}^{(i)} \right)^2$$

Used to compare performance of different models

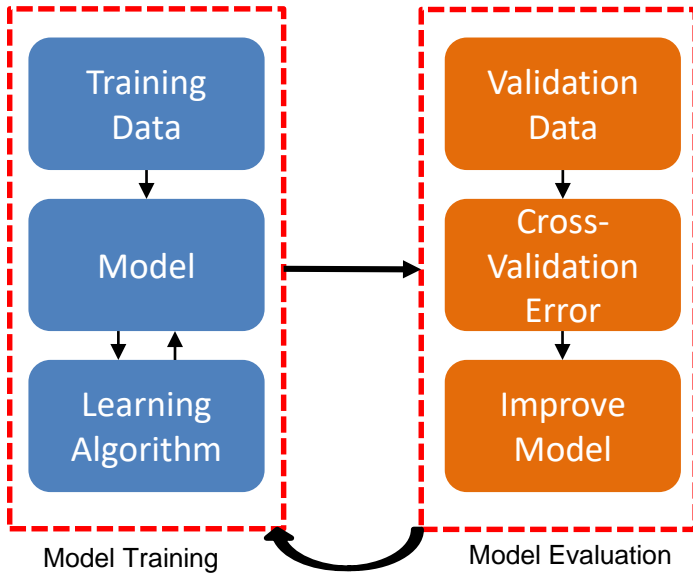
Test Error:

$$J_{test}(\mathbf{W}) = \frac{1}{2m} \sum_{i=1}^m \left(\widehat{y_{test}^{(i)}} - y_{test}^{(i)} \right)^2$$

Used to evaluate the accuracy of the chosen model with actual test data



Putting Everything Together



Practice!

Access this link and download the multivariate regression demo notebook along with the data folder

<https://tinyurl.com/y7t3qufd>

Access this link and download the Mini Project notebook along with the data folder

<https://tinyurl.com/y7t3qufd>

This project will:

- Guide you through the steps of preliminary data analysis
- Model Evaluation
- Fitting Polynomial Models
- The dangers of overfitting