DS4023 Machine Learning

Lecture 2:
Linear Regression

Mathematical Sciences
United International College

Outline

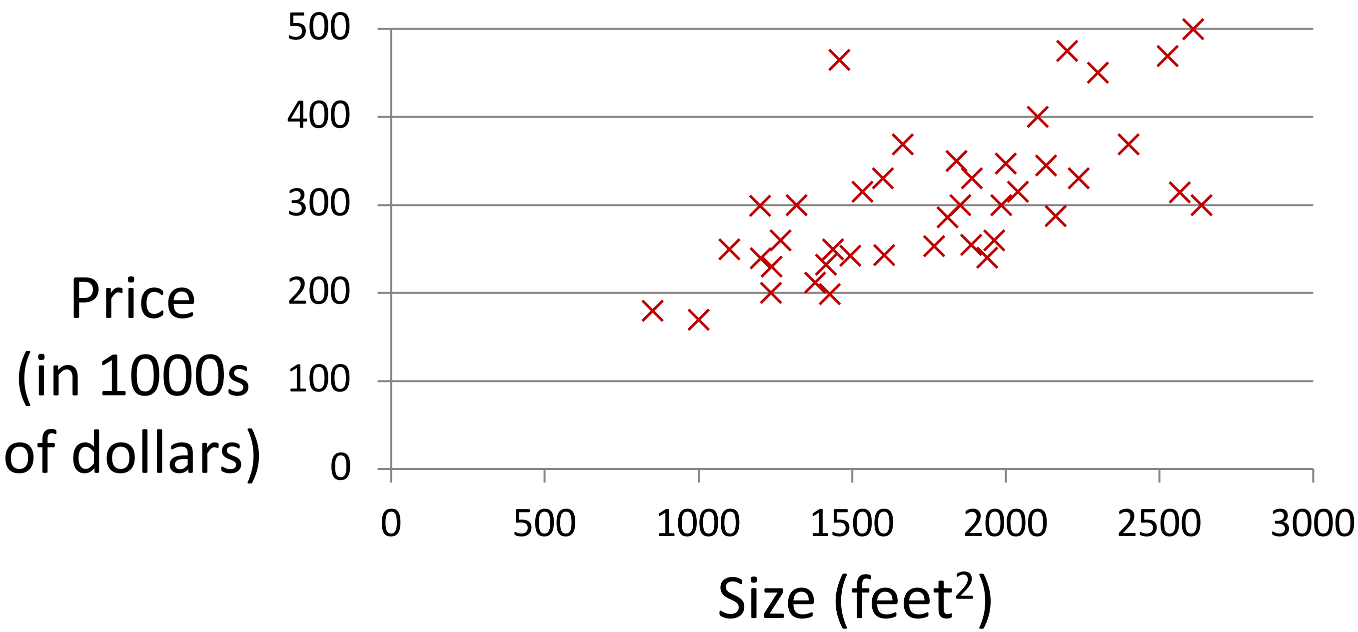
- Linear regression
- Cost function
- Gradient descent
- Linear regression for multiple variables

Supervised Learning Examples

- Suppose we have a dataset giving the living areas and prices of 47 houses from Portland, Oregon:

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Supervised Learning Examples



Given data like this, how can we learn to predict the prices of other houses in Portland, as a function of the size of their living areas?

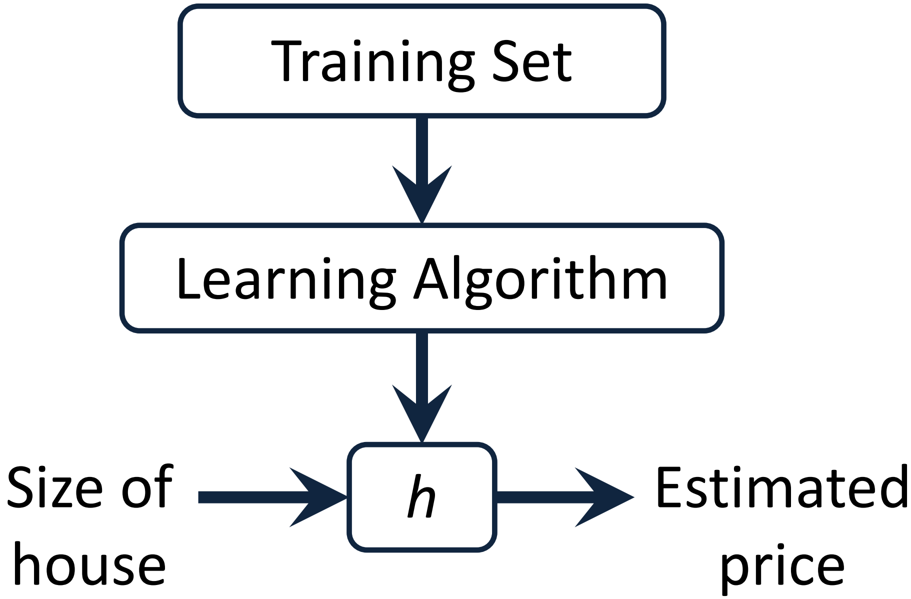
Supervised Learning Examples

- Supervised learning
  - Given the “right answer” for each example in the data.
- When the target variable that we’re trying to predict is continuous, such as in our housing example, we call the learning problem a regression problem.
- When the target variable can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment), we call it a classification problem.

Linear Regression

- Notation
  - Input variable/feature: $x^{(i)}$denote the i^{th} input variable
  - Output/target variable: $y^{(i)}$
  - Training example: $(x^{(i)}, y^{(i)})$
  - Training set: $\{(x^{(i)}, y^{(i)}); i = 1,2, \dots, m\}$ (a list of m training examples)
  - Space of input values: X ; space of output values: Y
- To describe the supervised learning problem slightly more formally, our goal is:
  - Given a training set, to learn a function (hypothesis) $h: X \longmapsto Y$, so that $h(x)$ is a “good” predictor for the corresponding value of y.

Model Representation



How do we represent h?

- $h_\theta(x) = \theta_0 + \theta_1 x$ or $h(x) = \theta_0 + \theta_1 x$
- We will predict that y is a linear function of x (straight line)
- Linear regression with one variable (univariate linear regression).

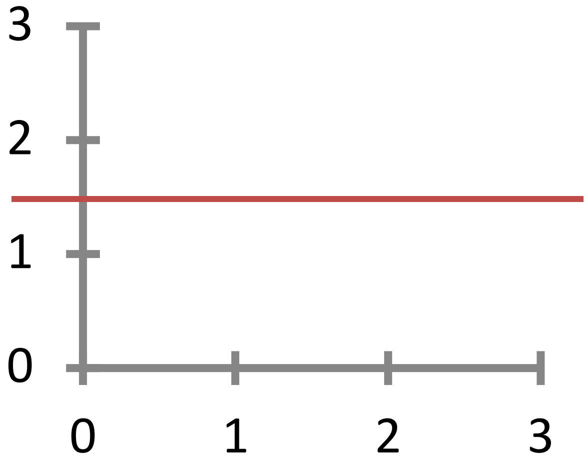
Cost Function

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

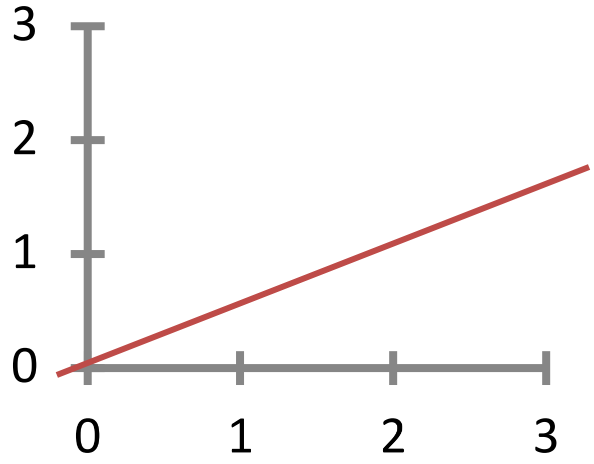
- Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$
  - $\theta_i{}'s$ are the parameters (weights)
- How to choose $\theta_i{}'s$?

Cost Function

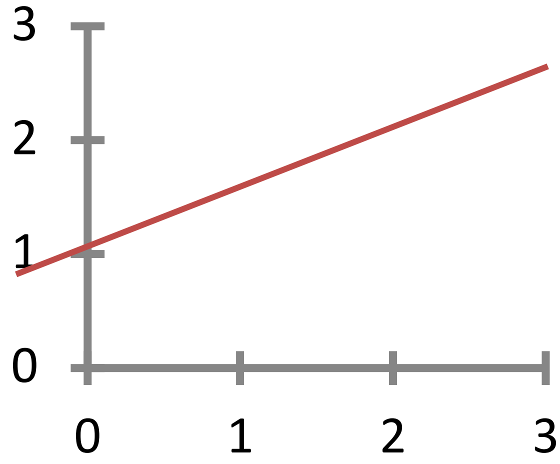
$$h_\theta(x) = \theta_0 + \theta_1 x$$



$\theta_0 = 1.5$
$\theta_1 = 0$



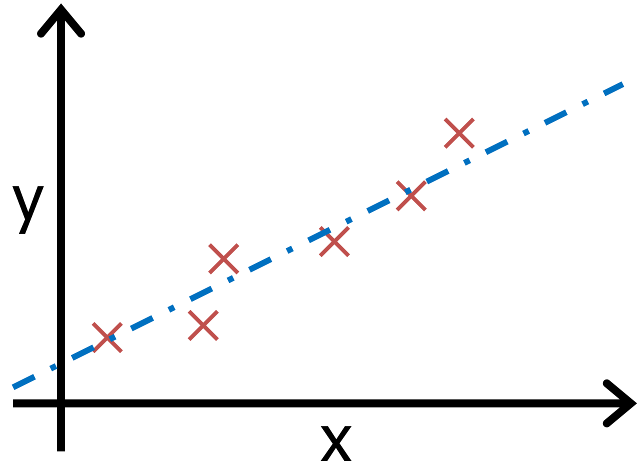
$\theta_0 = 0$
$\theta_1 = 0.5$



$\theta_0 = 1$
$\theta_1 = 0.5$

Cost Function

- Idea: Choose $\theta_i{}'s$, $(\theta_0,\ \theta_1)$, so that $h_\theta(x)$ is close to y for our training example.



$h_\theta(x) = \theta_0 + \theta_1 x$

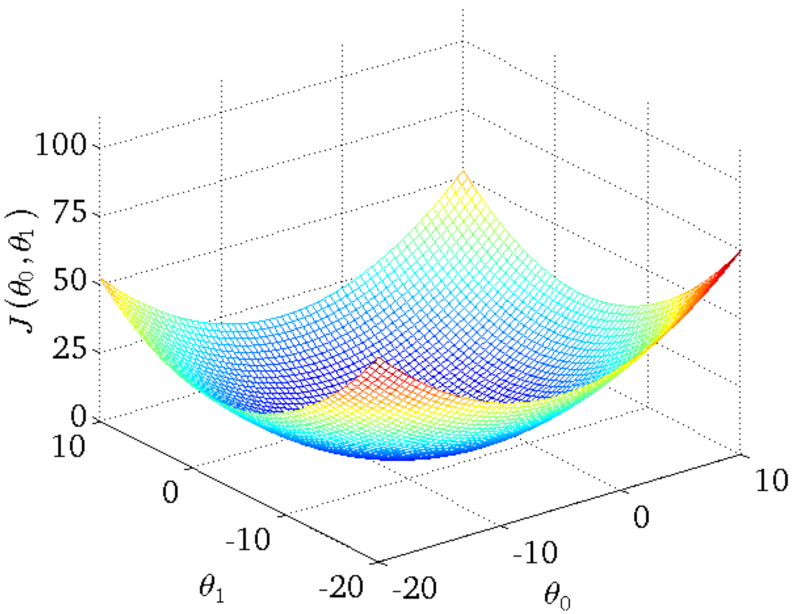
- To formalize this, we will define a cost function that measures, for each value of the $\theta's$, how close the $h_\theta\left(x^{(i)}\right)$'s are to the corresponding $y^{(i)}{}'s$:

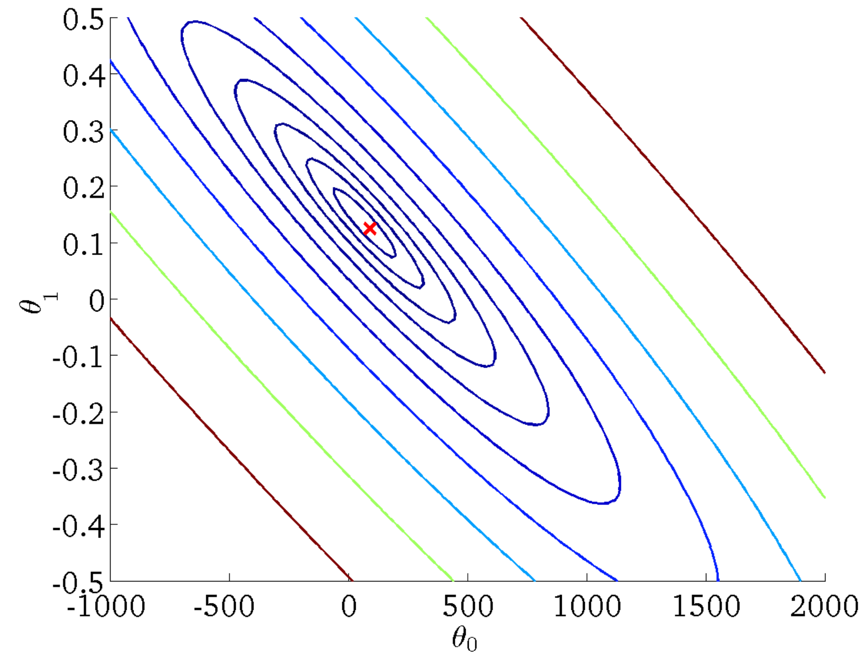
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)})^2 \text{ ,where } h_\theta\left(x^{(i)}\right) = \theta_0 + \theta_1 x^{(i)}$$

Linear Regression

- Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$
- Parameters: θ_0, θ_1
- Cost function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
- Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$
  - $J(\theta_0, \theta_1)$ is a function of the parameters θ_0, θ_1

Cost Function Visualization

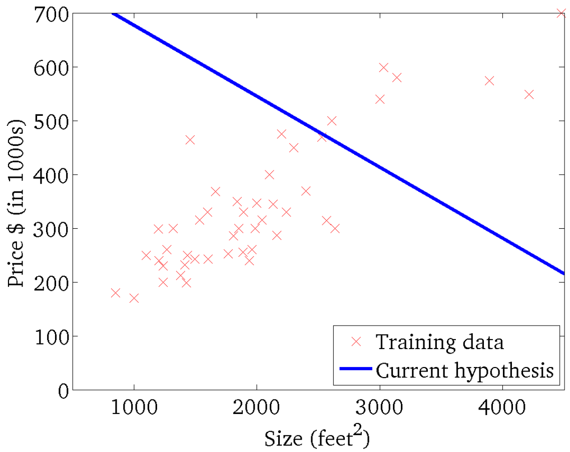


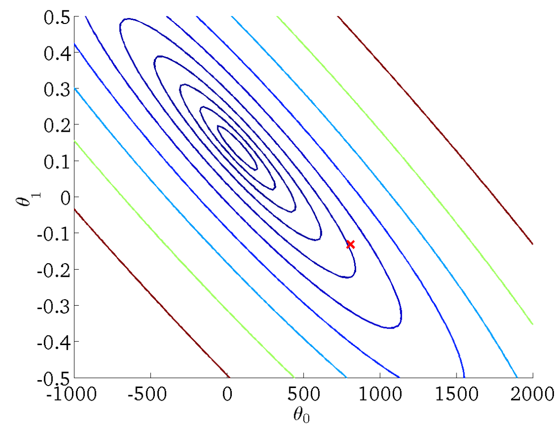


3-D surface plot, vary the parameter values, get different surface height, cost function value.

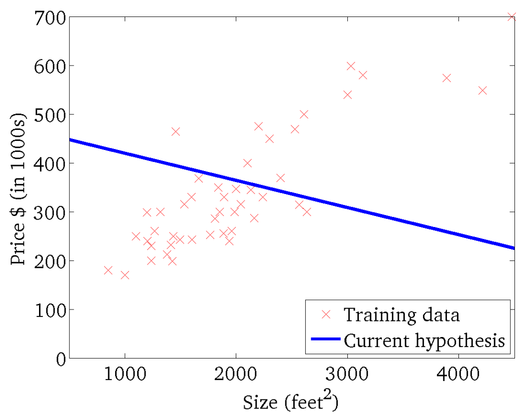
Contour plot, axis are θ_0, θ_1. Ellipse denotes a set of points, takes on the same value for $J(\theta_0, \theta_1)$

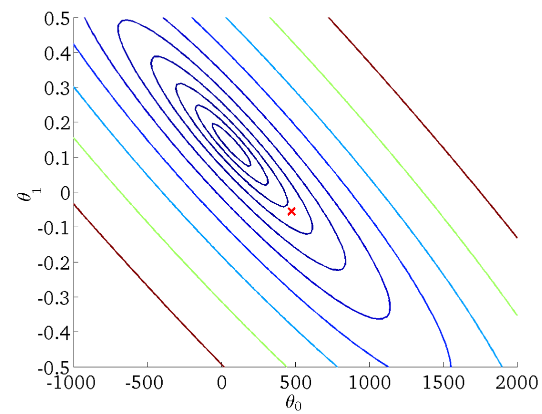
Cost Function Visualization





$\theta_0 = 800, \theta_1 = -0.15$





$\theta_0 = 450, \theta_1 = -0.05$

Given a particular point, corresponding to a set of parameters (shown in the contour plot), we will get the corresponding hypothesis (shown in the left figure).

Gradient Descent

- We need an efficient algorithm for automatically finding the value of parameters $(\theta_i' s)$ that minimized the cost function $J(\theta_0, \theta_1)$
- Gradient descent is a general algorithm, not only used in linear regression, but all over the place in machine learning.

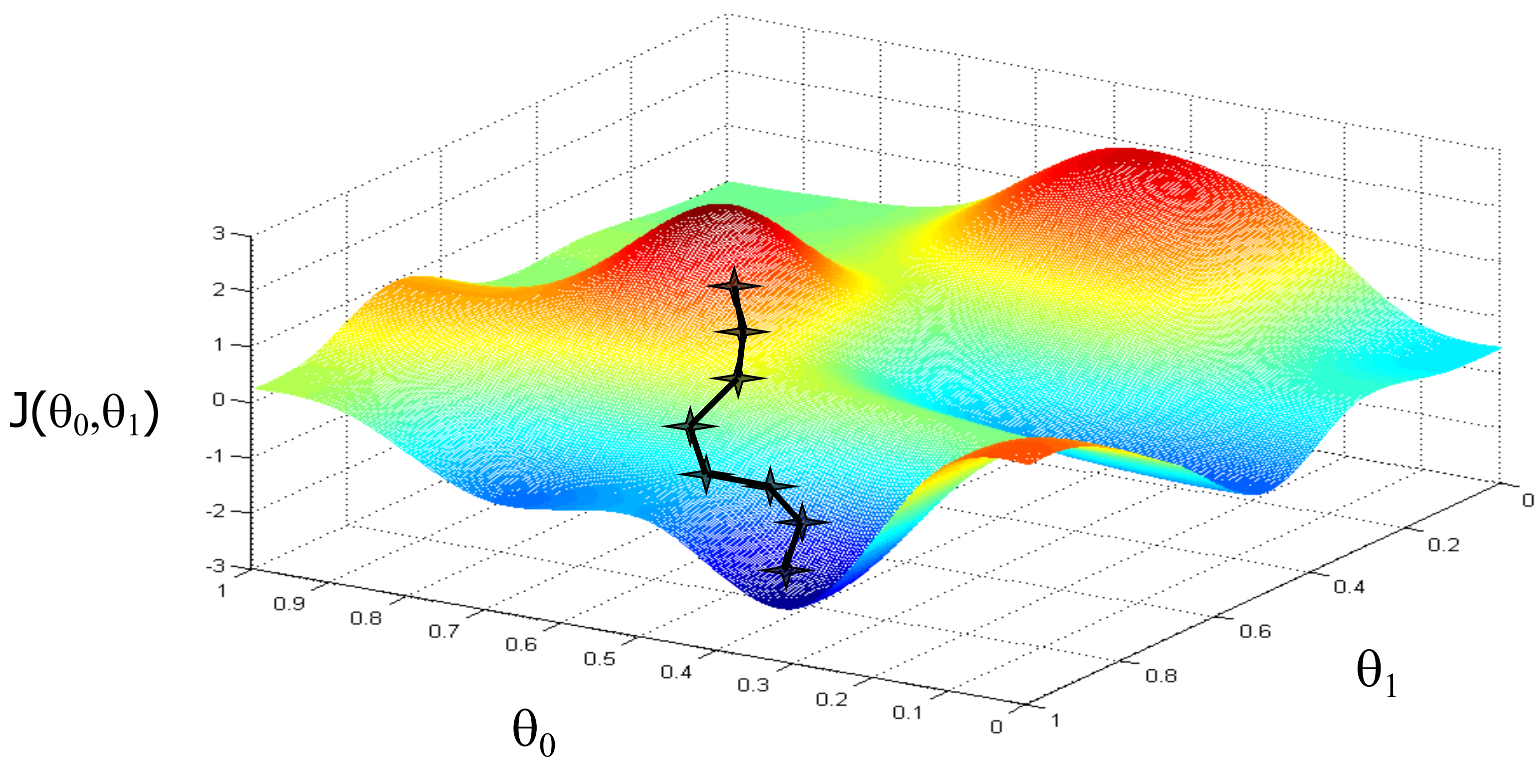
Gradient Descent

- The gradient of a function f, denoted as ∇f, is the collection of all its partial derivatives into a vector.
  - Can be interpreted as the "direction and rate of fastest increase". The direction of the gradient is the direction of fastest increase of the function at p, and its magnitude is the rate of increase in that direction.
- Gradient descent is an optimization problem used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.

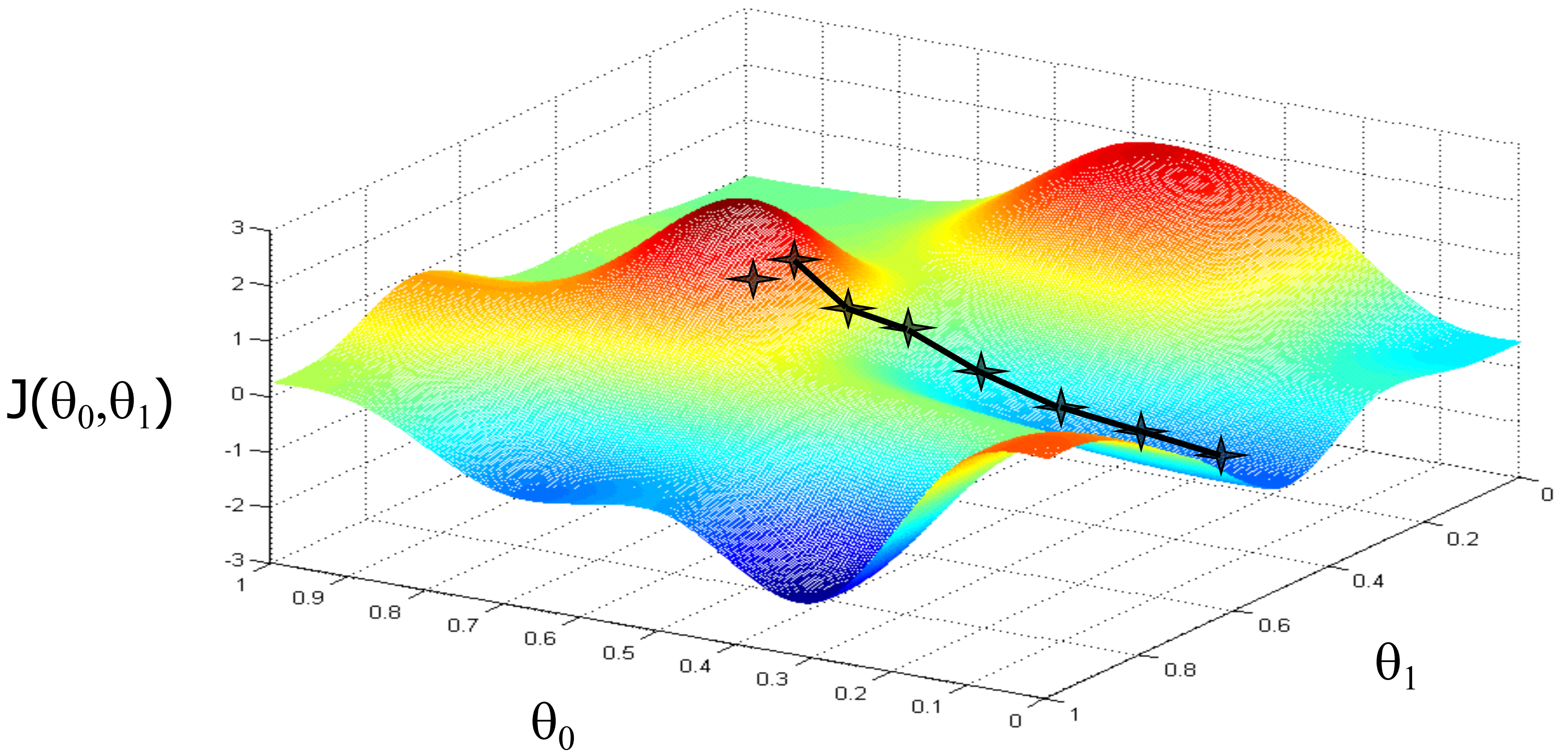
Gradient Descent

- Problem setting:
  - For some function $J(\theta_0, \theta_1)$
  - Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$
- Outline
  1. Start with some initial guess for θ_0, θ_1
  2. Repeatedly changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$, until we hopefully end up with a value of θ_0, θ_1 that minimizes $J(\theta_0, \theta_1)$.

Gradient Descent Visualization



Gradient Descent Visualization



Gradient Descent

- Problem setting:
  - For some function $J(\theta_0, \theta_1)$
  - Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$
- Gradient descent algorithm

`repeat until convergence`$\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

$\}$

- Note:
  - "$:=$" denotes the assignment operator
  - α is called the learning rate, used to control how big a step in gradient descent
  - $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is the partial derivative term

Gradient Descent

- Gradient descent algorithm

  repeat until convergence{

  $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \text{ (for } j = 0 \text{ and } j = 1\text{)}$$

  }
- Simultaneous update:
  1. $temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
  2. $temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
  3. $\theta_0 := temp0$
  4. $\theta_1 := temp1$

Gradient Descent

- Gradient descent algorithm

  repeat until convergence{

  $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \text{ (for } j = 0 \text{ and } j = 1)$$

  }

- In order to implement this algorithm, we have to work out what is the partial derivative term on the right hand
  - $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} (\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2)$

    $= \frac{\partial}{\partial \theta_j} (\frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2)$
  - $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$
  - $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) =$

Gradient Descent

- Gradient descent algorithm

  repeat until convergence{

  $$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right) - y^{(i)}$$

  $$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta\left(x^{(i)}\right) - y^{(i)}) x^{(i)}$$

  }

  - Update θ_0 and θ_1 simultaneously
- "Batch" Gradient Descent
  - Gradient descent on the original cost
  - Each step of gradient descent uses all the training examples

Stochastic Gradient Descent

- Another method is called stochastic gradient descent
  - Repeatedly run through the training set, each time encounter a training example, update the parameter using the single training example only.
- Stochastic Gradient descent algorithm

repeat until convergence{

$$\theta_0 := \theta_0 - \alpha(h_\theta(x^{(i)}) - y^{(i)})$$
$$\theta_1 := \theta_1 - \alpha(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

}

  - Update θ_0 and θ_1 simultaneously.
  - Gradient descent on the error of single example only, preferred when the training set is large.

Lab Exercise1

- In this exercise, you will implement linear regression with one variables get to see it work on data.
- Download *LinearRegression-Part1.ipynb* and *data1.txt* from iSpace
- Submit the completed notebook on iSpace.

Linear Regression with Multiple Variables

- House Price example with multiple features (variables)

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

- Notation
  - n: number of features; m: number of training examples
  - $x^{(i)}$: input of i^{th} training example
  - $x_j^{(i)}$: value of feature j in i^{th} training example
  - $y^{(i)}$: output/target of i^{th} training example

Linear Regression with Multiple Variables

- House Price example with multiple features (variables)

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

- Previous hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1$
- Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

Linear Regression with Multiple Variables

- Linear regression with multiple variables
  - Multivariate linear regression
  - Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
- To simplify our notation, we also introduce the convention of letting $x_0 = 1$ (this is the intercept term)
  - $h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$
  - on the right-hand, we are viewing θ and x both as vectors

Multivariate Linear Regression

- Hypothesis: $h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$
- Parameters: θ ($n + 1$ dimensional vector, $\theta_0, \theta_1, \ldots, \theta_n$)
- Cost function: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta\left(x^{(i)}\right) - y^{(i)})^2$
- Goal: $\min_\theta J(\theta)$
  - $J(\theta)$ is a function of the parameters $\theta_0, \theta_1, \ldots, \theta_n$
- Gradient descent algorithm

repeat until convergence{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$ (for $j = 0,1,2, \ldots, n$, update simultaneously)

}

Gradient Descent

- Gradient descent algorithm

  repeat until convergence{

  $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$ (for $j = 0,1,2,\ldots,n$, update simultaneously)

  }

- $\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2 \right)$

  $$= \frac{\partial}{\partial \theta_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots - y^{(i)} \right)^2 \right)$$

- $\frac{\partial}{\partial \theta_0} J(\theta) =$
- $\frac{\partial}{\partial \theta_1} J(\theta) =$
- $\frac{\partial}{\partial \theta_2} J(\theta) =$
- …

Gradient Descent

- Gradient descent algorithm (Batch)

repeat until convergence{

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for $j = 0,1,2, \dots, n$, update simultaneously)

}

- Stochastic gradient descent:

```
repeat until convergence{
    for i=1 to m { // scan all training examples
```

$$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for $j = 0,1,2, \dots, n$, update simultaneously)

```
    }
}
```

Gradient Descent vs Normal Equation

- $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)})^2$
- Use vector representation:
  - $J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$
  - X: $m \times n$ matrix; θ: n-dimensional vector; y: m-dimensional vector
- Gradient descent: ($\frac{\partial}{\partial X}f(AX + B) = A^T\frac{\partial}{\partial Y}f(Y), \frac{d}{dX}X^TX = 2X$)
  - $\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{m}X^T(X\theta - y)$
  - $\theta := \theta - \alpha\frac{1}{m}X^T(X\theta - y)$
- Normal equation:
  - $\theta = (X^TX)^{-1}X^Ty$

Gradient Descent

- Batch gradient descent has to scan through the entire training set before taking a single step
  - a costly operation if m is large
- Stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at.
- Often, stochastic gradient descent gets θ “close” to the minimum much faster than batch gradient descent.
  - when the training set is large, stochastic gradient descent is often preferred over batch gradient descent

Gradient Descent in Practice

- Feature scaling
  - Idea: Make sure features are on a similar scale
  - Example:
    - x_1 = size (0-2000 feet^2) => $x_1 = \frac{\text{size (feet}^2)}{2000}$
    - x_2 = number of bedrooms (1-5) => $x_1 = \frac{\text{number of bedrooms}}{5}$

Gradient Descent in Practice

- Min-Max Normalization
  - min_A, max_A: minimum and maximum values of attribute A
  - maps a value v of A to a new range $[nmin_A, nmax_A]$

$$v' = \frac{v - min_A}{max_A - min_A}(nmax_A - nmin_A) + nmin_A$$

- Z-Score normalization
  - It converts all indicators to a common scale with an mean of zero and standard deviation of one

$$v' = \frac{v - \bar{A}}{\sigma_A}$$

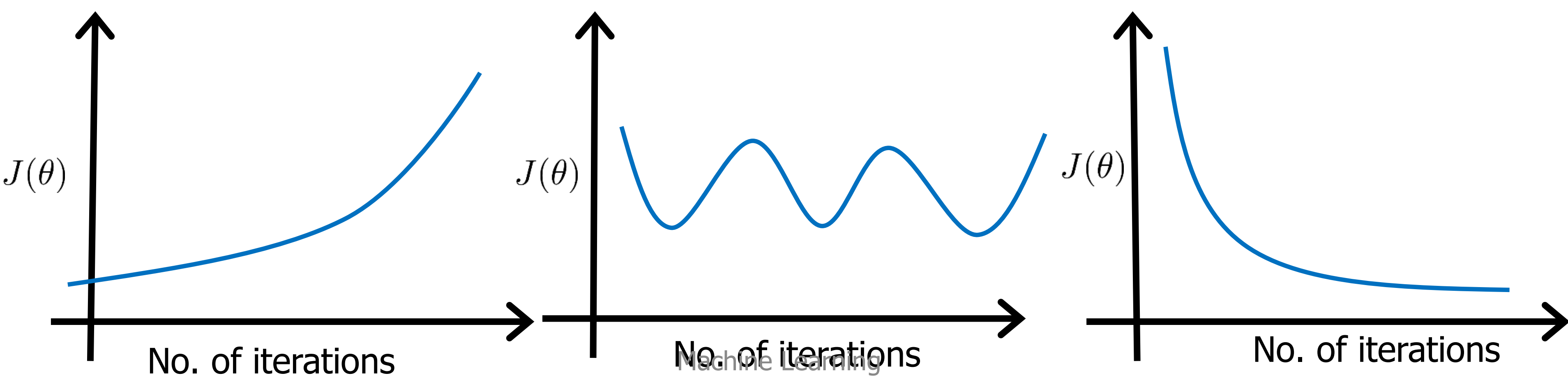
  - useful when we do not know the minimum and maximum of an attribute, or when we have outliers

Gradient Descent in Practice

- Learning rate:
  - $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
- How to make sure gradient descent is working correctly? How to choose learning rate?
- Declare convergence if $J(\theta)$ decreases by less than a small number (e.g., 10^{-3}) in one iteration.

Gradient Descent in Practice

- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
  – But if α is too small, gradient descent can be slow to converge.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.
- In order to debug, usually plot $J(\theta)$ as a function of the number of iterations
  – Too choose α, try 0.001,0.003,0.01,0.03,0.1,0.3,...
  – What is the problem of α in each following figure?



Feature Choice

- Consider the housing prices prediction
  - feature1: frontage
  - feature2: depth
  - Apply linear regression directly:

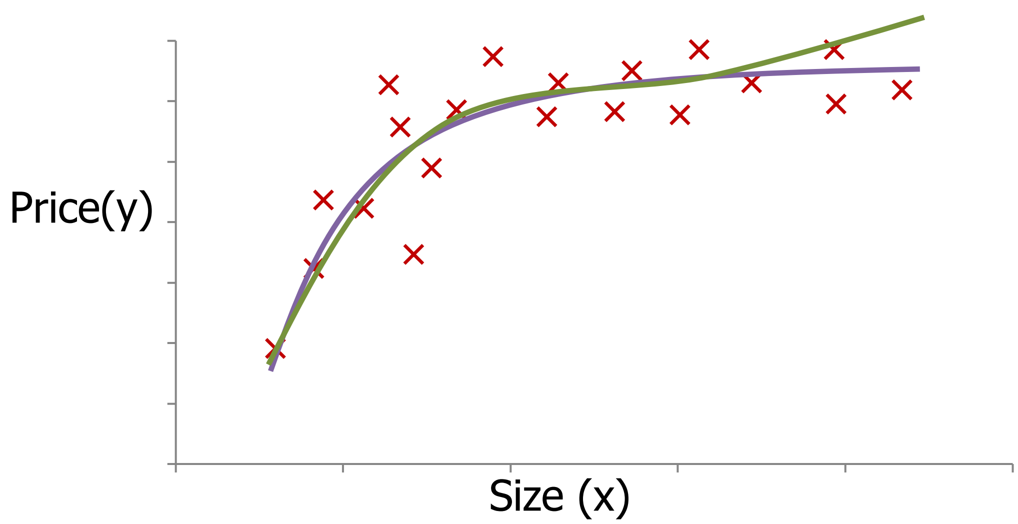
$$h_\theta(x) = \theta_0 + \theta_1 \times frontage + \theta_2 \times depth$$

- Create new features by yourself
  - feature_new : area = frontage × depth
  - Apply linear regression using new feature:

$$h_\theta(x) = \theta_0 + \theta_1 \times area$$

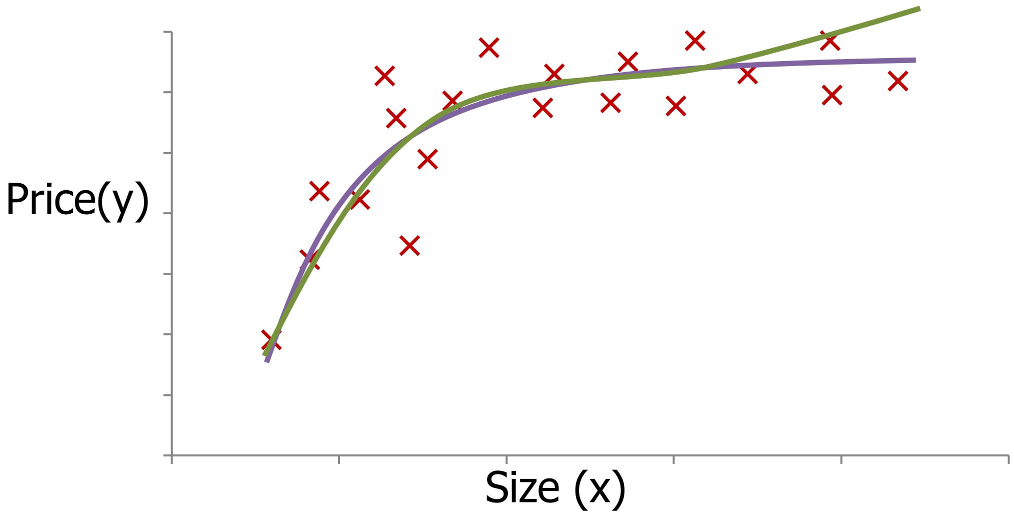
Polynomial Regression

- Given a house price dataset as illustrated in the figure, there may exists several models that fit the data
  - Quadratic model: $\theta_0 + \theta_1 x + \theta_2 x^2$
  - Cubic model: $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$



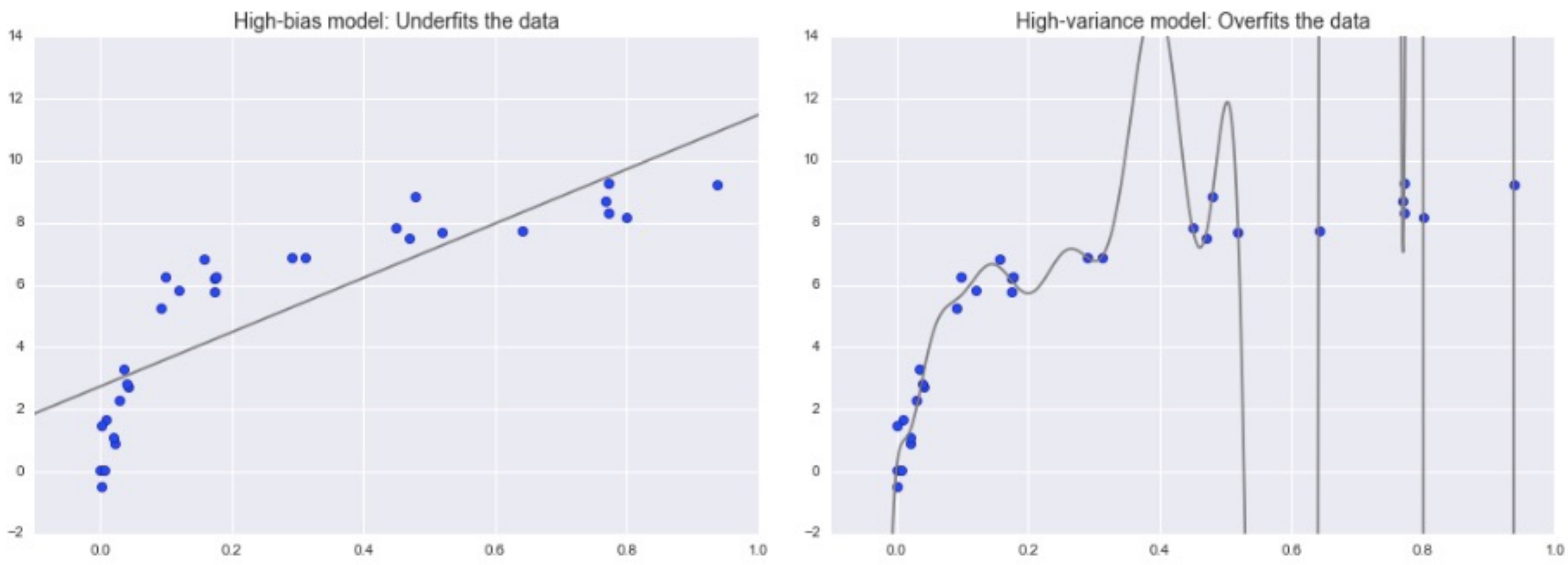
Polynomial Regression

- Polynomial regression
  - Cubic model: $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- We can fit cubic the model by making a simple modification:
  - $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$
  - $x_1 = (size), x_2 = (size)^2, x_3 = (size)^3$
  - feature scaling becomes more important in this case

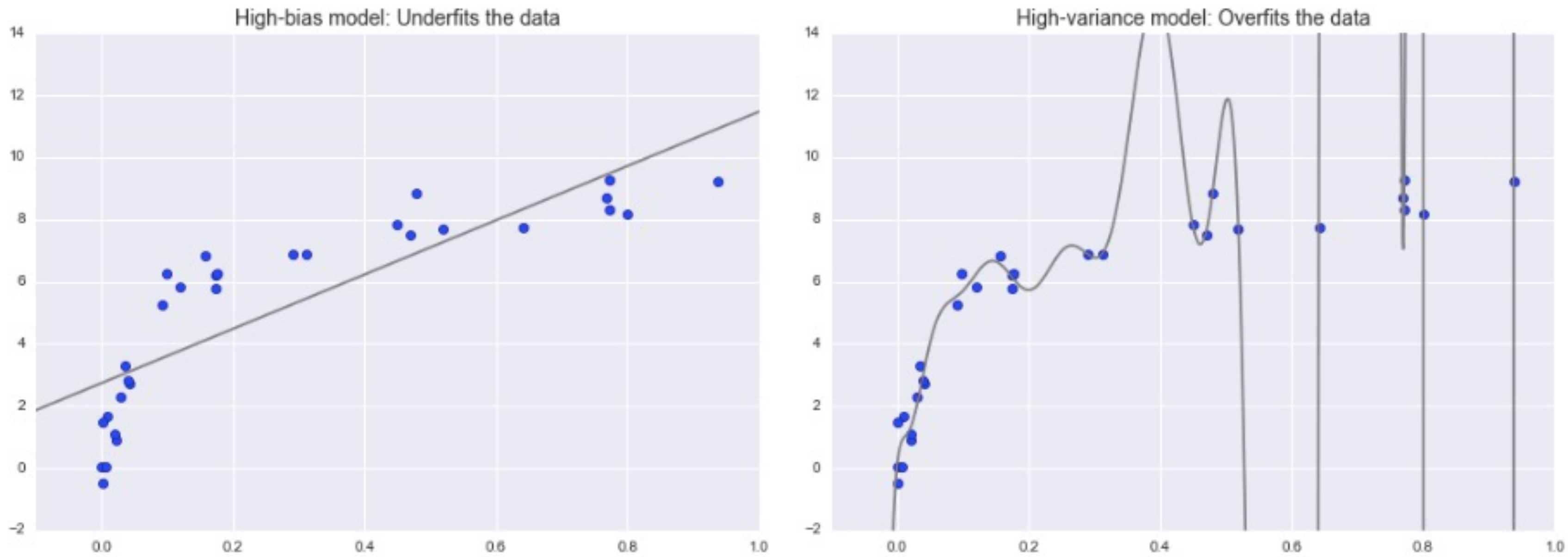


Model Complexity Selection

- "The best model" is about finding a sweet spot in the tradeoff between *bias* and *variance*.
  - Consider the following figure, which presents two regression fits to the same dataset:

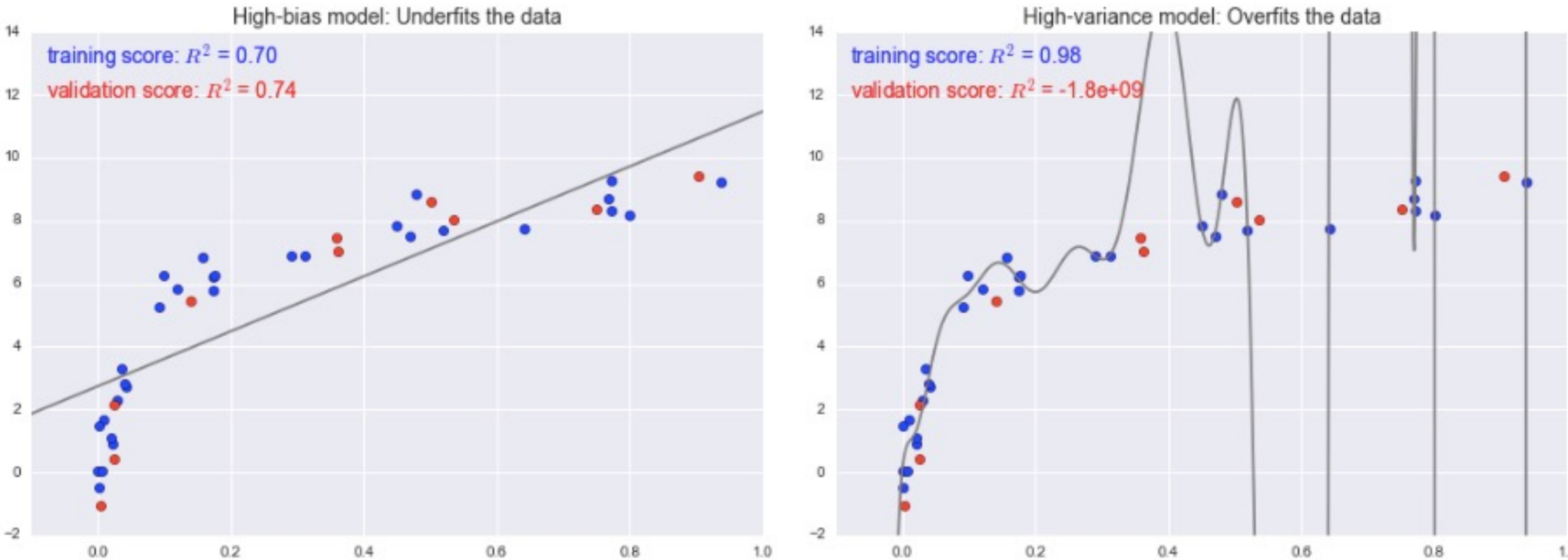


Model Complexity Selection



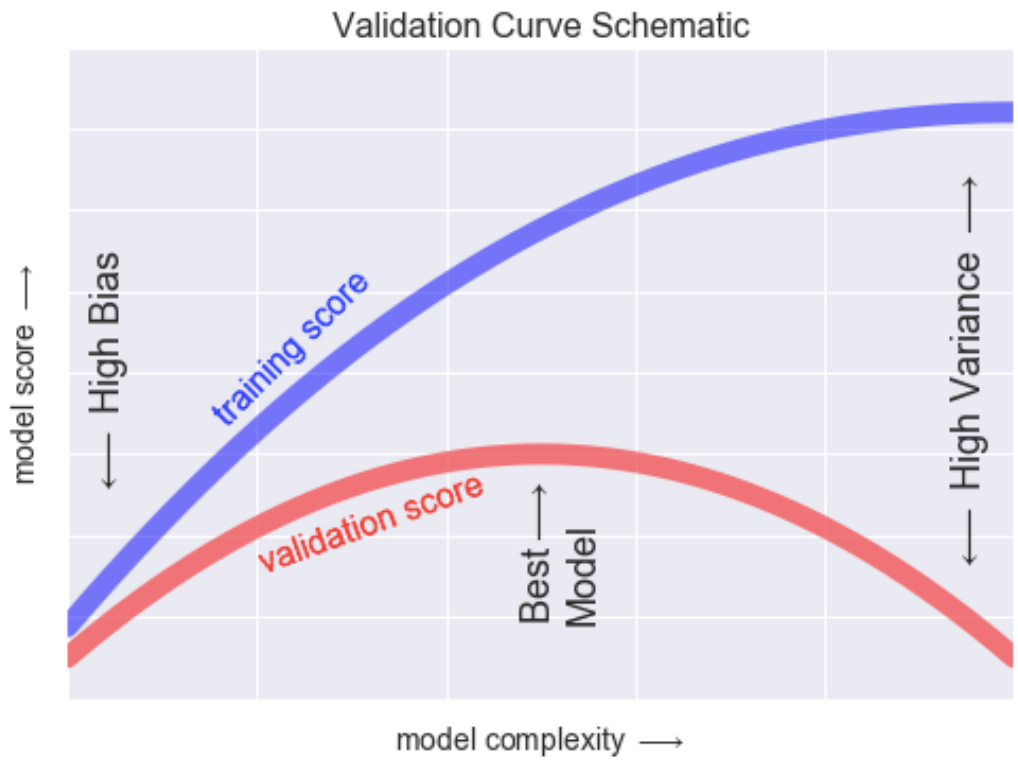
- Underfit: a model does not have enough model flexibility to suitably account for all the features in the data (high bias).
- Overfit: a model can accurately describes the all training data, even the random errors (high variance).

Model Complexity Selection



- Underfit: the performance of the model on the validation set is similar to the performance on the training set.
- Overfit: the performance of the model on the validation set is far worse than the performance on the training set.

Model Complexity Selection



- The training score is everywhere higher than the validation score.
- For very low model complexity, the model is a poor predictor both for the training data and for any previously unseen data.
- For very high model complexity , model predicts the training data very well, but fails for any previously unseen data.

Lab Exercise2

- In this exercise, you will implement linear regression with multiple variables get to see it work on data.
- Download *LinearRegression-Part2.ipynb* and *data2.txt* from iSpace
- Submit the completed notebook on iSpace.