Linear Regression with Multiple Variables

Multiple Features

Multiple features (variables)

Size in feet^2 (x)	Price (\$) in 1000's (y)
2104	400
1416	232
1534	315
852	178
…	…

$$f_{w,b}(x) = wx + b$$

Multiple features (variables)

Size in feet^2	Number of bedrooms	Number of floors	Age of home in years	Price (\$) in \$1000's
x_1	x_2	x_3	x_4	
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

$i=2$

$j=1\ldots4$

$n=4$

x_j = j^{th} feature

n = number of features

$\vec{x}^{(i)}$ = features of i^{th} training example

$x_j^{(i)}$ = value of feature j in i^{th} training example

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

$\vec{w} = [w_1 \ w_2 \ w_3 \ \ldots \ w_n]$ parameters of the model

vector $\vec{x} = [x_1 \ x_2 \ x_3 \ \ldots \ x_n]$

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b$$

dot product

multiple linear regression

(not multivariate regression)

Linear Regression with Multiple Variables

Vectorization

Parameters and features

$\vec{w} = [w_1 \quad w_2 \quad w_3]$ $\quad n=3$

b is a number

$\vec{x} = [x_1 \quad x_2 \quad x_3]$

linear algebra: count from 1

NumPy

w[0] w[1] w[2]

```
w = np.array([1.0,2.5,-3.3])
b = 4                          x[0] x[1] x[2]
x = np.array([10,20,30])
```

code: count from 0

Without vectorization

$$f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_j x_j\right) + b \qquad \sum_{j=1}^{n} \rightarrow j=1\ldots n \quad 1,2,3$$

range(0,n) → j=0...n-1

```
f = 0          range(n)
for j in range(0,n):
    f = f + w[j] * x[j]
f = f + b
```

Without vectorization

$n = 100{,}000$

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

```
f = w[0] * x[0] +
    w[1] * x[1] +
    w[2] * x[2] + b
```

Vectorization

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

```
f = np.dot(w,x) + b
```

Without vectorization

```
for j in range(0,16):
    f = f + w[j] * x[j]
```

t_0

```
f + w[0] * x[0]
```

t_1

```
f + w[1] * x[1]
```

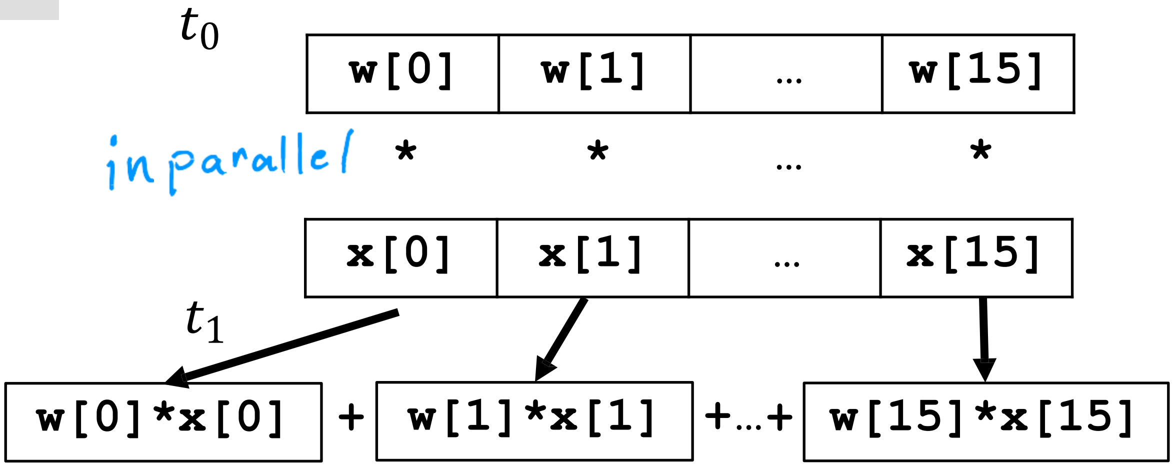
…

t_{15}

```
f + w[15] * x[15]
```

Vectorization

```
np.dot(w,x)
```



efficient → scale to large datasets

Gradient descent

$\vec{w} = (w_1 \quad w_2 \quad \cdots \quad w_{16})$ $\cancel{b}$ parameters

derivatives $\vec{d} = (d_1 \quad d_2 \quad \cdots \quad d_{16})$

```
w = np.array([0.5, 1.3, … 3.4])
d = np.array([0.3, 0.2, … 0.4])
```

compute $w_j = w_j - \overbrace{0.1}^{\text{learning rate } \alpha} d_j$ for $j = 1 \ldots 16$

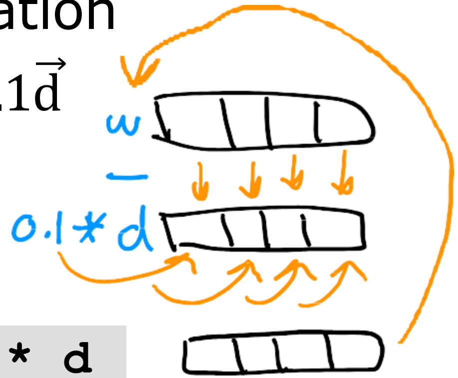
Without vectorization

$$w_1 = w_1 - 0.1 d_1$$
$$w_2 = w_2 - 0.1 d_2$$
$$\vdots$$
$$w_{16} = w_{16} - 0.1 d_{16}$$

```
for j in range(0,16):
    w[j] = w[j] - 0.1 * d[j]
```

With vectorization

$$\vec{w} = \vec{w} - 0.1\vec{d}$$




```
w = w - 0.1 * d
```

Linear Regression with Multiple Variables

Gradient Descent for Multiple Regression

	Previous notation	Vector notation
Parameters	$w_1, \cdots, w_n$ b	$\vec{\mathrm{w}} = [w_1 \quad \cdots \quad w_n]$ ← vector of length n b still a number
Model	$f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = w_1 x_1 + \cdots + w_n x_n + b$	$f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \vec{\mathrm{w}} \cdot \vec{\mathrm{x}} + b$ ↑ dot product
Cost function	$J(\underbrace{w_1, \cdots, w_n, b})$	$J(\vec{\mathrm{w}}, b)$

Gradient descent

Previous notation	Vector notation

repeat {
$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\underbrace{w_1, \cdots, w_n}, b)$
$b = b - \alpha \frac{\partial}{\partial b} J(\underbrace{w_1, \cdots, w_n}, b)$
}
	repeat {
	$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b)$
	$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$
	}

Gradient descent

One feature

repeat {

$$w = w - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}}_{\frac{\partial}{\partial w} J(w,b)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)$$

simultaneously update w, b

}

n features $(n \geq 2)$

repeat {

$j=1$

$$w_1 = w_1 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right) x_1^{(i)}}_{\frac{\partial}{\partial w_1} J(\vec{w},b)}$$

$\vdots$

$j=n$

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right) x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)$$

simultaneously update

w_j (for $j = 1, \cdots, n$) and b

}

An alternative to gradient descent

Normal equation

- Only for linear regression
- Solve for w, b without iterations

Disadvantages

- Doesn’t generalize to other learning algorithms.
- Slow when the number of features is large (> 10,000)

What you need to know

- Normal equation method may be used in machine learning libraries that implement linear regression.
- Gradient descent is the recommended method for finding parameters w,b

Practical Tips for Linear Regression

Feature Scaling

Feature and parameter values

$price = w_1 x_1 + w_2 x_2 + b$

size #bedrooms

x_1: size (feet²)
range: $300 - 2{,}000$

x_2: # bedrooms
range: $0 - 5$

House: $x_1 = 2000$, $x_2 = 5$, $price = \$500\text{k}$ one training example

size of the parameters w_1, w_2?

$w_1 = 50$, $w_2 = 0.1$, $b = 50$

$price = \underbrace{50 * 2000}_{100{,}000\text{K}} + \underbrace{0.1 * 5}_{0.5\text{K}} + \underbrace{50}_{50\text{K}}$

$price = \$100{,}050.5\text{k}$ $= \$100{,}050{,}500$

$w_1 = 0.1$, $w_2 = 50$, $b = 50$

small

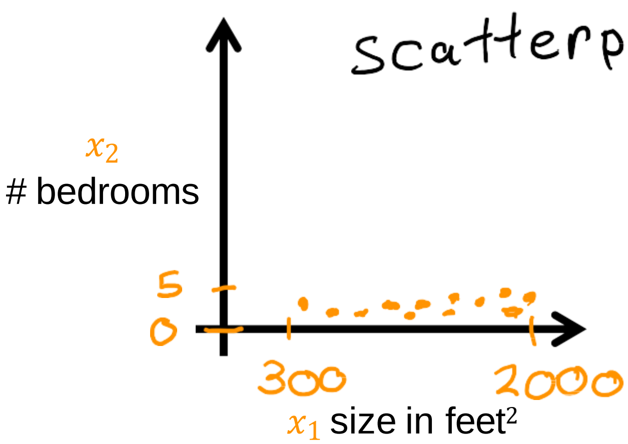
$price = \underbrace{0.1 * 2000k}_{200\text{K}} + \underbrace{50 * 5}_{250\text{K}} + \underbrace{50}_{50\text{K}}$

$price = \$500\text{k}$ more reasonable

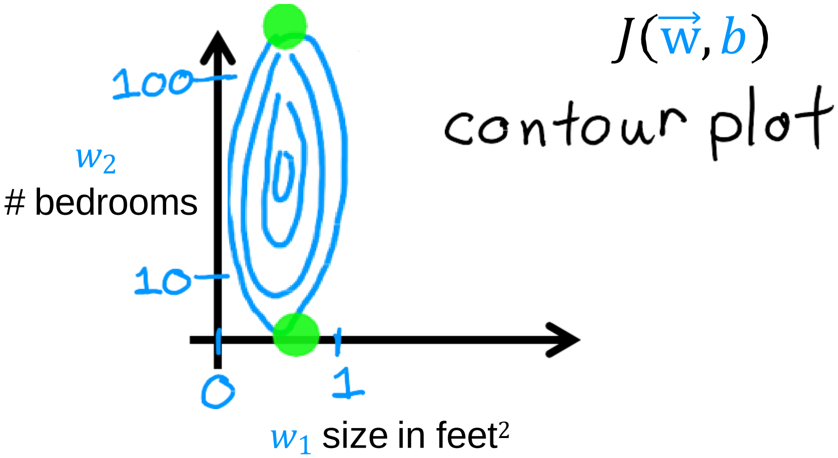
Feature size and parameter size

	size of feature x_j	size of parameter w_j
size in feet²		
#bedrooms		

Features

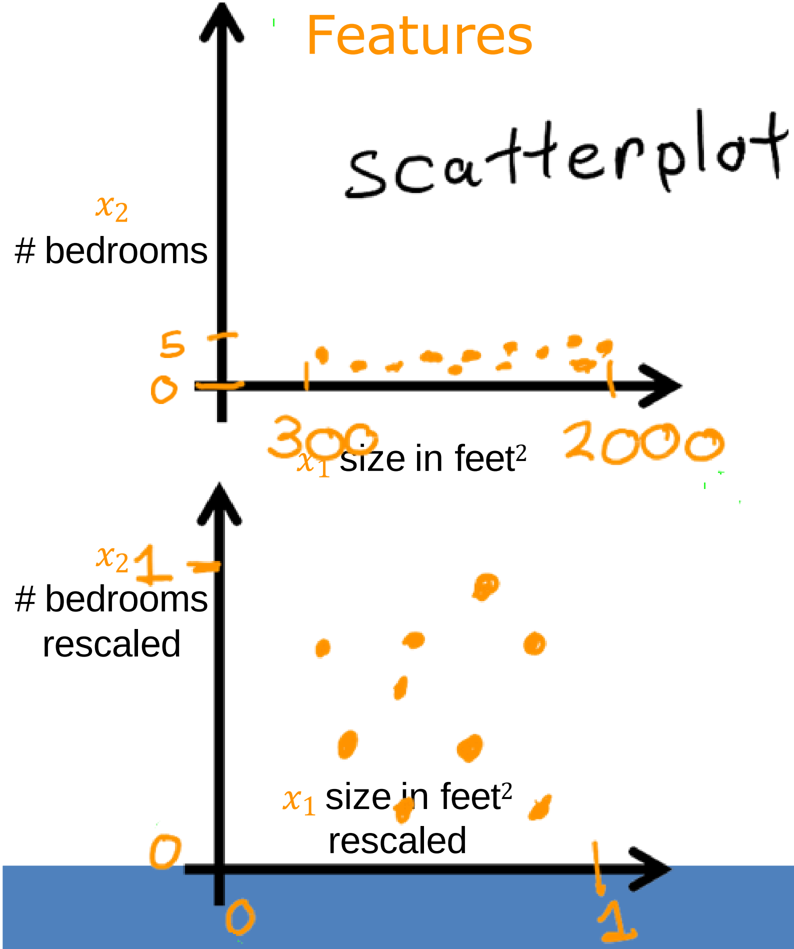


Parameters

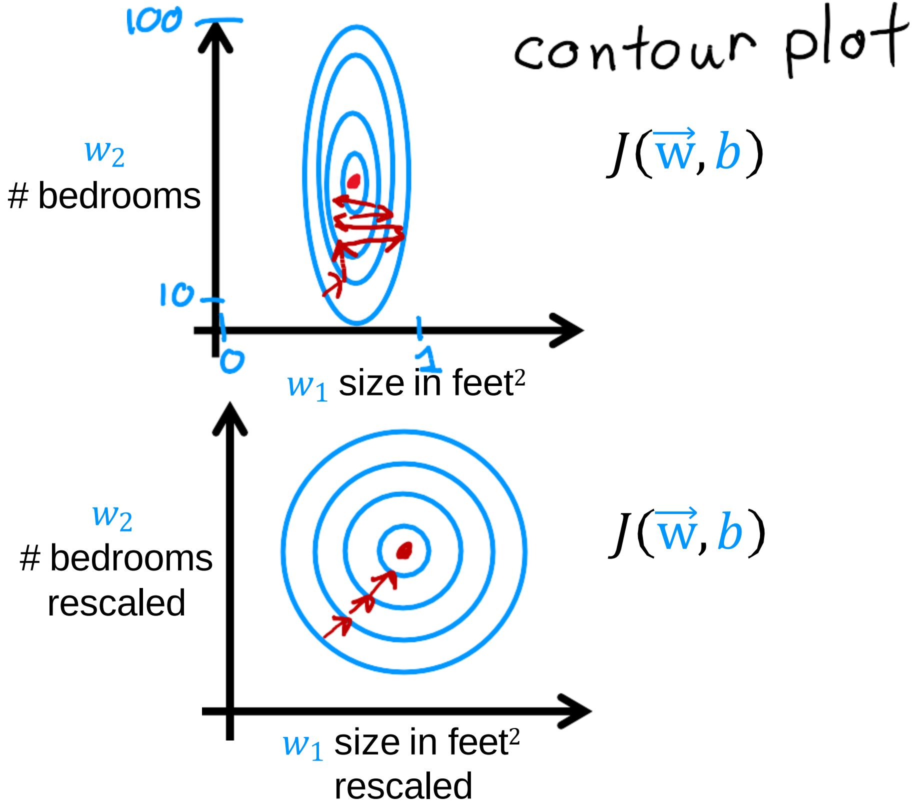


Feature size and gradient descent

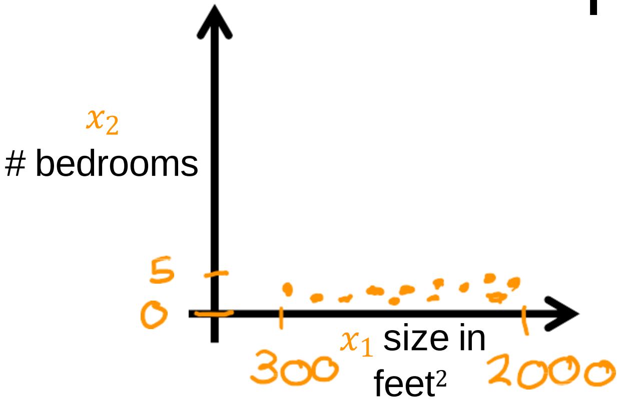
Features



Parameters



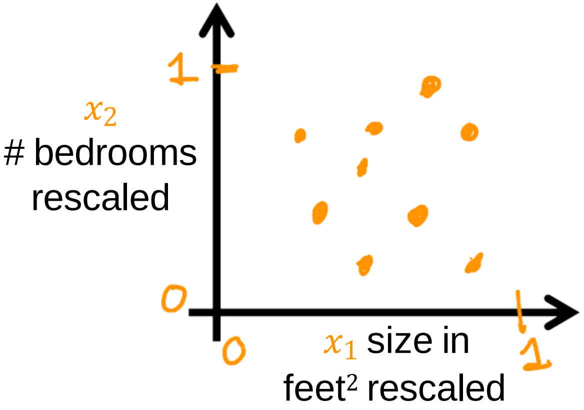
Feature scaling



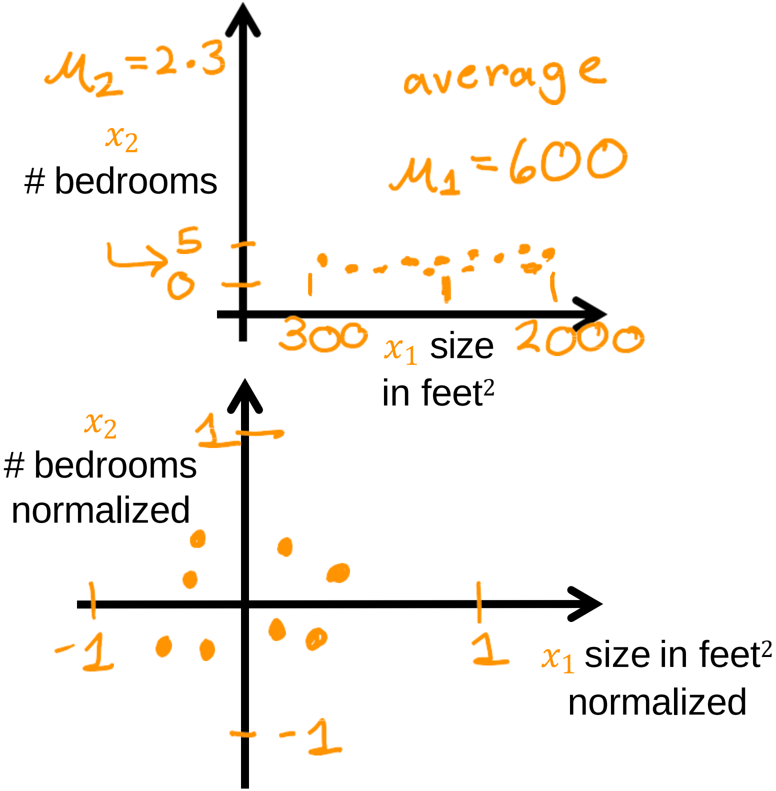
$$300 \le x_1 \le 2000 \qquad 0 \le x_2 \le 5$$

$$x_{1,scaled} = \frac{x_1}{\underset{\text{max}}{2000}} \qquad x_{2,scaled} = \frac{x_2}{\underset{\text{max}}{5}}$$

$$0.15 \le x_{1,scaled} \le 1 \qquad 0 \le x_{2,scaled} \le 1$$



Mean normalization



$300 \leq x_1 \leq 2000$ $\qquad$ $0 \leq x_2 \leq 5$

$$x_1 = \frac{x_1 - \mu_1}{2000 - 300} \qquad x_2 = \frac{x_2 - \mu_2}{5 - 0}$$

max-min $\qquad$ max-min

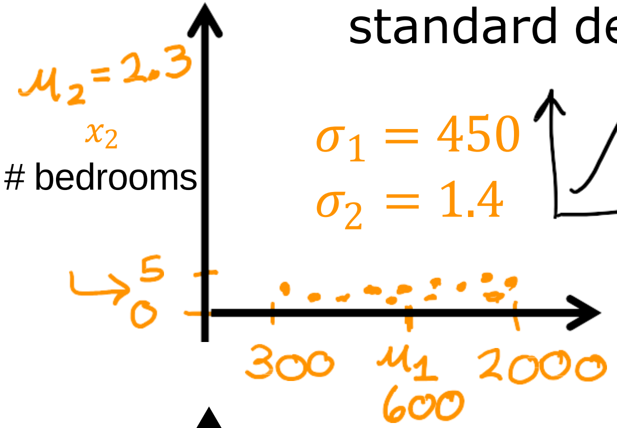
$-0.18 \leq x_1 \leq 0.82$ $\qquad$ $-0.46 \leq x_2 \leq 0.54$

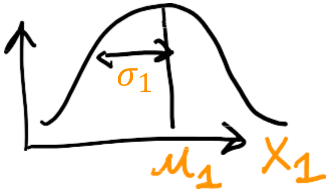
Z-score normalization

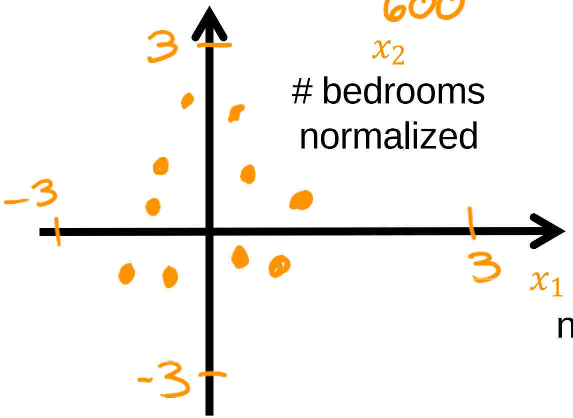
standard deviation σ

$\sigma_1 = 450$

$\sigma_2 = 1.4$







$300 \le x_1 \le 2000$ $\qquad$ $0 \le x_2 \le 5$

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1} \qquad x_2 = \frac{x_2 - \mu_2}{\sigma_2}$$

$-0.67 \le x_1 \le 3.1$ $\qquad$ $-1.6 \le x_2 \le 1.9$

Feature scaling

aim for about $-1 \le x_j \le 1$ for each feature x_j

$$-3 \le x_j \le 3$$

$$-0.3 \le x_j \le 0.3$$

$$0 \le x_1 \le 3$$

$$-2 \le x_2 \le 0.5$$

$$-100 \le x_3 \le 100$$

$$-0.001 \le x_4 \le 0.001$$

$$98.6 \le x_5 \le 105$$

Feature scaling

aim for about $-1 \leq x_j \leq 1$ for each feature x_j

$$-3 \leq x_j \leq 3$$

$$-0.3 \leq x_j \leq 0.3$$

acceptable ranges

$0 \leq x_1 \leq 3$ okay, no rescaling

$-2 \leq x_2 \leq 0.5$ okay, no rescaling

$-100 \leq x_3 \leq 100$ too large → rescale

$-0.001 \leq x_4 \leq 0.001$ too small → rescale

$98.6 \leq x_5 \leq 105$ too large → rescale

Practical Tips for Linear Regression

Checking Gradient Descent for Convergence

Gradient descent

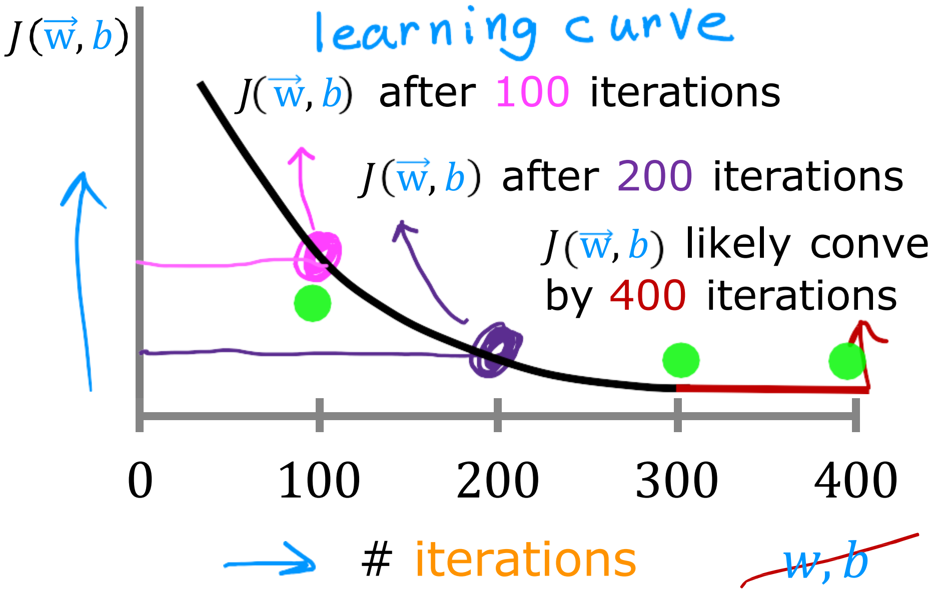
$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$$

Make sure gradient descent is working correctly

objective: $\min_{\vec{w},b} J(\vec{w}, b)$

$J(\vec{w}, b)$ should decrease after every iteration



iterations needed varies 30 1,000 100,000

Automatic convergence test

Let ε "epsilon" be 10^{-3}.
0.001

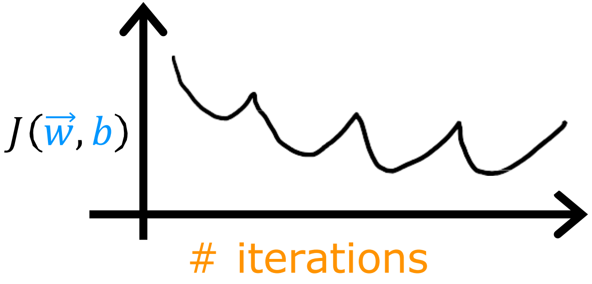
If $J(\vec{w}, b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence.

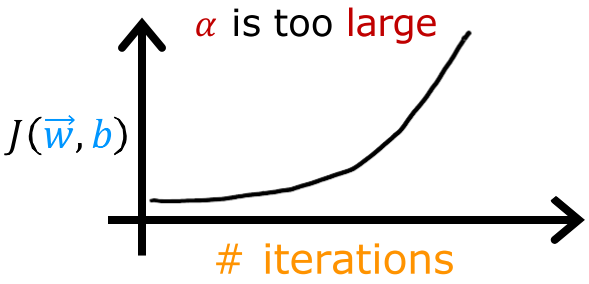
(found parameters $\vec{w}, b$ to get close to global minimum)

Practical Tips for Linear Regression

Choosing the Learning Rate

Identify problem with gradient descent





or learning rate is too large

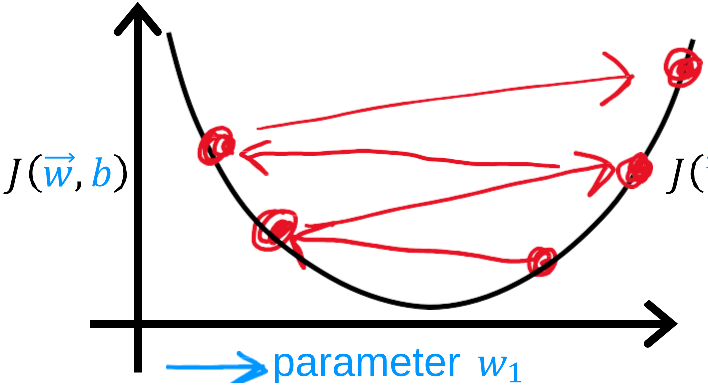
$w_1 = w_1 + \alpha d_1$

use a minus sign

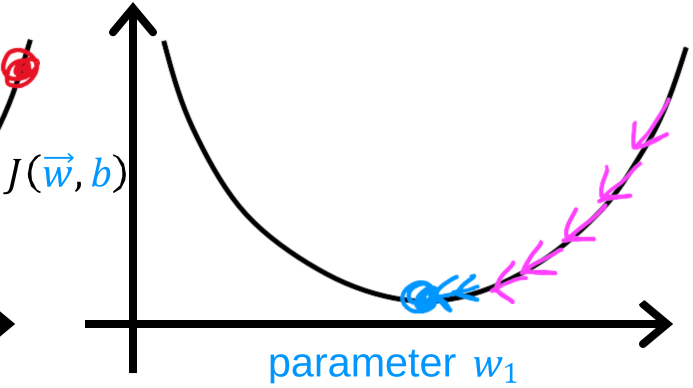
$w_1 = w_1 - \alpha d_1$

Adjust learning rate

α is too big



Use smaller α



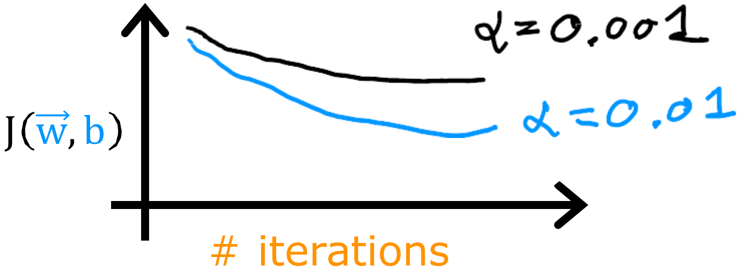
With a small enough α, $J(\vec{w}, b)$ should decrease on every iteration

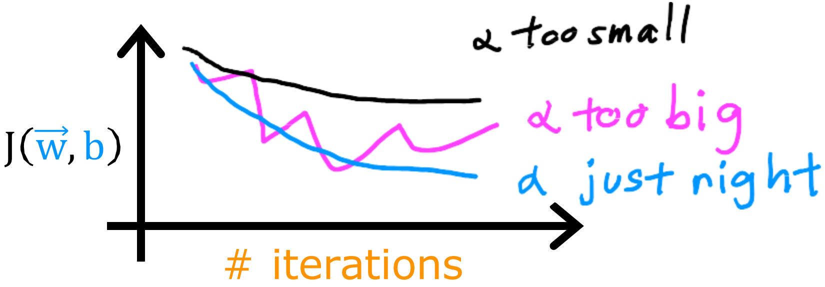
If α is too small, gradient descent takes a lot more iterations to converge

Values of α to try:

… 0.001 0.003 0.01 0.03 0.1 0.3 1 …

3X ≈3X 3X ≈3X 3X ≈3X





Practical Tips for Linear Regression

Feature Engineering

Feature engineering

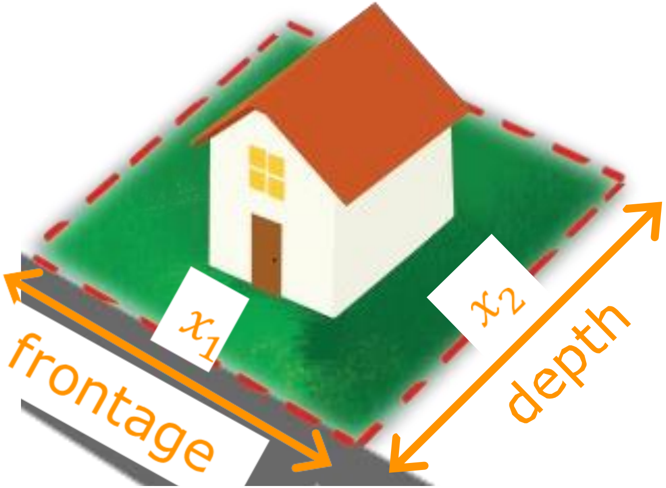
$$f_{\vec{w},b}(\vec{x}) = w_1 \underbrace{x_1}_{\text{frontage}} + w_2 \underbrace{x_2}_{\text{depth}} + b$$

$$area = frontage \times depth$$

$$x_3 = x_1 x_2$$

new feature

$$f_{\vec{w},b}(\vec{x}) = \underbrace{w_1}x_1 + \underbrace{w_2}x_2 + \underbrace{w_3}x_3 + b$$



Feature engineering:
Using intuition to design new features, by transforming or combining original features.

Important notes

- Sci-kit learn library `LinearRegression()` model uses Ordinary Least Squares (OLS) Method with no involvement of optimization
- Stochastic Gradient Descent (SGD) regressor in sci-kit learning uses a stochastic gradient descent algorithm to optimize the cost function.

Ordinary Least Squares (OLS) Method

The ordinary Least Squares (OLS) estimator for linear regression is derived by minimizing the sum of squared errors between the observed values and the predicted values of a linear model.

The goal of OLS is to minimize the sum of the squared residuals (errors) between the actual values $\mathbf{y}$ and the predicted values $\hat{\mathbf{y}}$:

$$\mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\beta$$

We define the sum of squared errors (SSE) as:

$$S(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta)$$

This function, $S(\beta)$, is the **objective function** we aim to minimize with respect to β.

Expanding the Objective Function:

Let's expand the SSE to make it easier to differentiate:

$$S(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta)$$

Using the distributive property of the transpose operation, we get:

$$S(\beta) = \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\beta + \beta^\top \mathbf{X}^\top \mathbf{X}\beta$$

This expression represents the sum of squared errors that we need to minimize with respect to β.

Minimization by Taking the Derivative:

To find the value of $\boldsymbol{\beta}$ that minimizes $S(\boldsymbol{\beta})$, we take the derivative of $S(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ and set it equal to zero.

The derivative of $S(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ is:

$$\frac{\partial S(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\beta$$

Now, set the derivative equal to zero to find the minimum:

$$-2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\beta = 0$$

Simplifying the equation:

$$\mathbf{X}^\top \mathbf{X}\beta = \mathbf{X}^\top \mathbf{y}$$

OLS Estimator

To solve for $\boldsymbol{\beta}$, we multiply both sides of the equation by $(\mathbf{X}^\top \mathbf{X})^{-1}$, assuming $\mathbf{X}^\top \mathbf{X}$ is invertible:

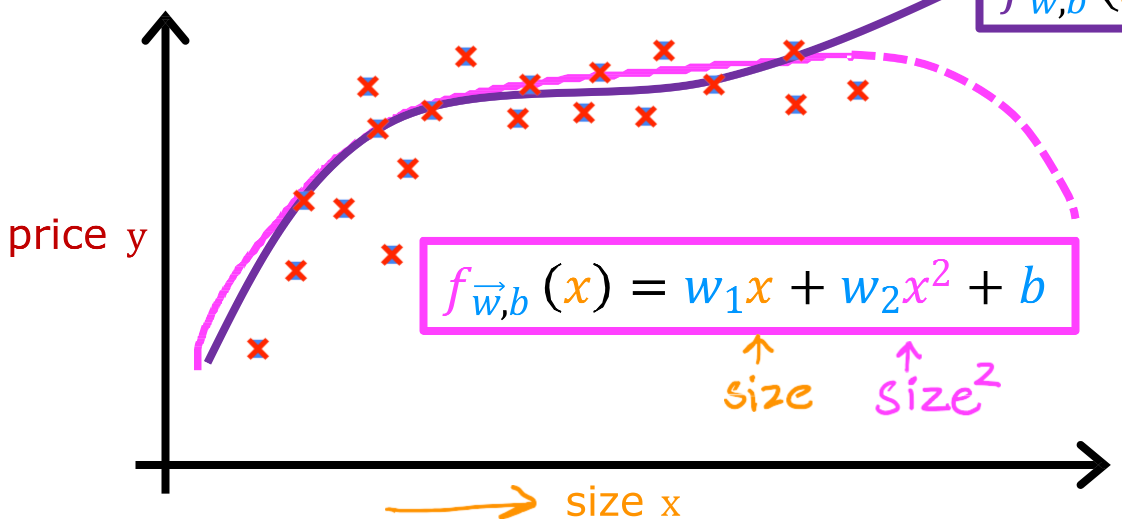
$$\boldsymbol{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

The above expression is the **OLS estimator** for $\boldsymbol{\beta}$, which gives the coefficients that minimize the sum of squared residuals:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

This is the closed-form solution for the OLS estimator.

Polynomial regression



$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + b$$

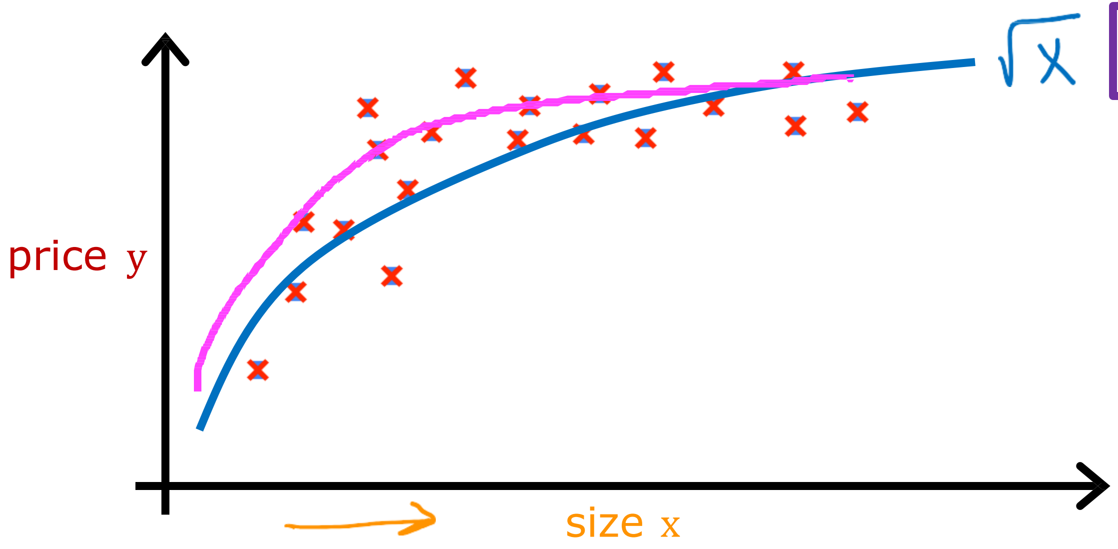
size
$1-10^3$

size²
$1-10^6$

size³
$1-10^9$

feature scaling

Choice of features



$$\boxed{f_{\vec{w},b}(x) = w_1 x + w_2\sqrt{x} + b}$$

size

$\sqrt{\text{size}}$

what features to use?

Assumptions for Lines Regression

1. **Linear relationship**:

1. **Little or no multi-collinearity**:
   - It is assumed that there is little or no multicollinearity in the data.
   - Multicollinearity occurs when the features (or independent variables) are not independent of each other.

2. **Little or no autocorrelation**:
   - Another assumption is that there is little or no autocorrelation in the data.
   - Autocorrelation occurs when the residual errors are not independent of each other.

Assumptions for LR

- **No outliers:**
  - We assume that there are no outliers in the data. Outliers are data points that are far away from the rest of the data. Outliers can affect the results of the analysis.
- **Homoscedasticity**:
  - Homoscedasticity describes a situation in which the error term (that is, the “noise” or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables.