3. LINEAR REGRESSION WITH MULTIPLE VARIABLES

CSC 462[HMLSKT:2,4]

MULTIPLE FEATURES (VARIABLES)

Size (feet^2) x_1	# of bedrooms x_2	# of floors x_3	Home age (years) x_4	Price ($1000) y
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

$x^{(i)}$: input features of ith example

$x_j^{(i)}$: value of feature j in ith example

HYPOTHESIS FOR MULTIVARIATE LR

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$

$$h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\underbrace{[\theta_0 \quad \theta_1 \quad \theta_2 \quad \ldots \quad \theta_n]}_{\theta^T} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{\text{x}} = \boldsymbol{\theta}^T \mathbf{x}$$

GRADIENT DESCENT FOR MULTIPLE VARIABLES

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function: $J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:

Repeat $\{$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$ simultaneously update for every $j = 0, \ldots, n$

$\}$

GRADIENT DESCENT

Previously ($n = 1$):

Repeat {

$$\underbrace{\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

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

(simultaneously update θ_0, θ_1)

}

New algorithm ($n \geq 1$):

Repeat {

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

simultaneously update θ_j for $j = 0, \ldots, n$

}

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$\ldots$

POLYNOMIAL REGRESSION

HOUSING PRICES PREDICTION

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

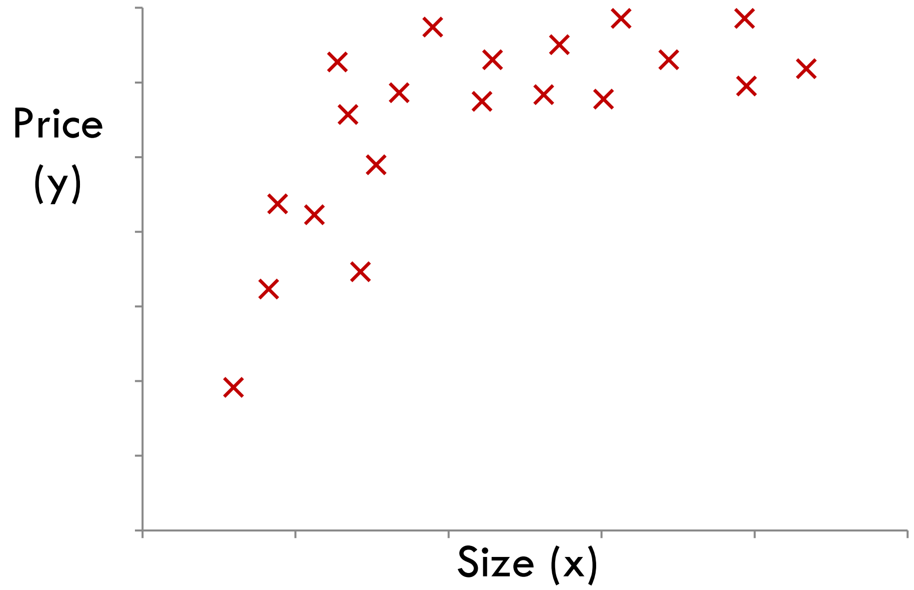
Area:

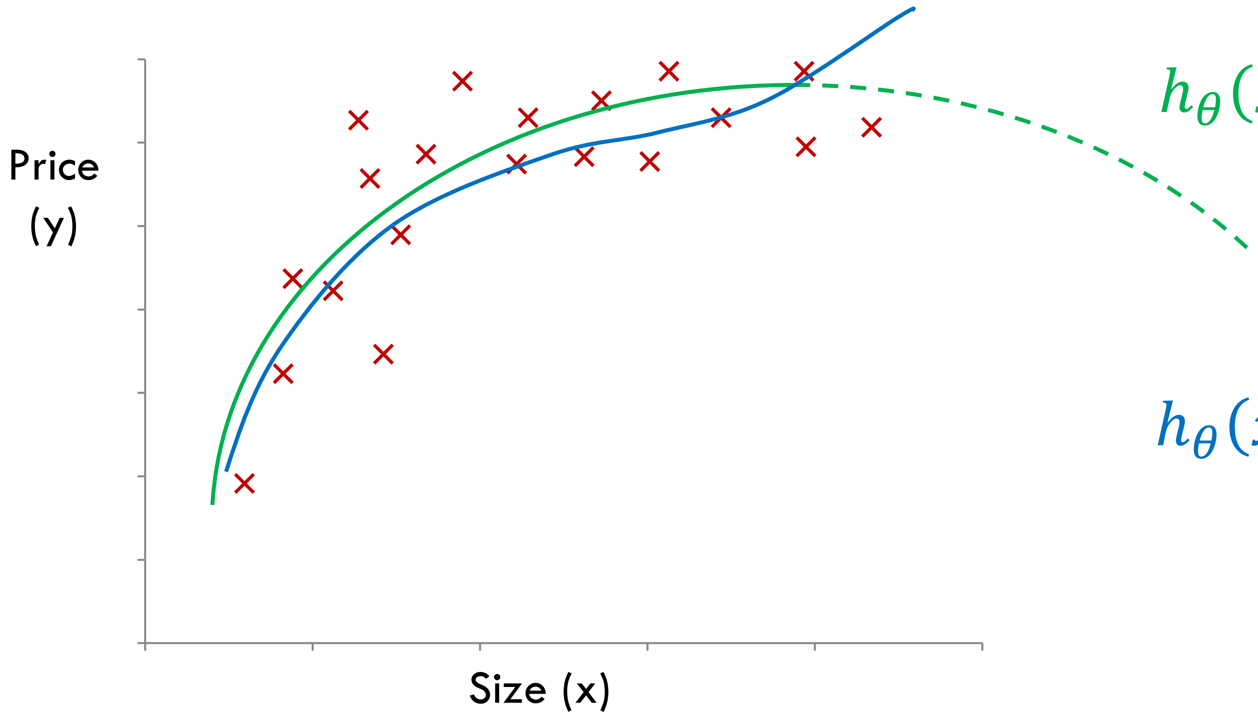
$$x = frontage * depth$$

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

POLYNOMIAL REGRESSION

- Allows you to use the machinery of linear regression to fit complicated and nonlinear functions.
- For these prices, a quadratic function might be a better fit





$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

← Multivariate Linear Regression hypothesis

So the model is *linear in the parameters θ*, even though it is nonlinear in the original feature (size).

$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

Feature scaling is important since the range becomes huge

WHY IS SCALING IMPORTANT

When you create polynomial features:

- $x_1 = \text{size}$
- $x_2 = (\text{size})^2$
- $x_3 = (\text{size})^3$

If "size" is in the hundreds:

- $x_1 \approx 10^2$
- $x_2 \approx 10^4$
- $x_3 \approx 10^6$

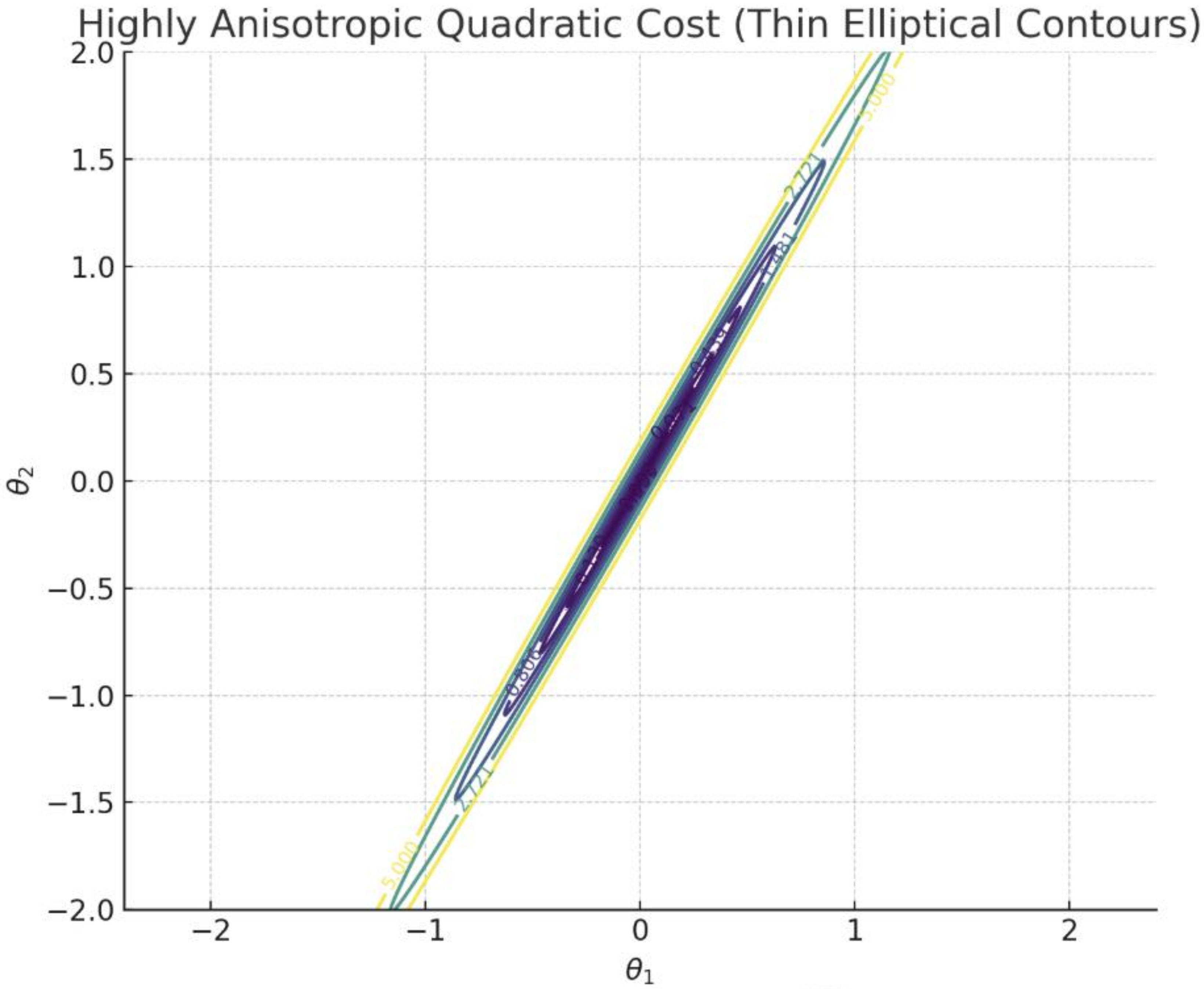
Now your feature matrix XXX has columns with **vastly different ranges.**
This causes two issues:

1. ELONGATED COST CONTOURS

The cost function $J(\theta)$ becomes highly stretched (like a very thin ellipse).

- For small-scale features, the cost is sensitive to changes in θ.
- For large-scale features, the cost changes slowly with θ.

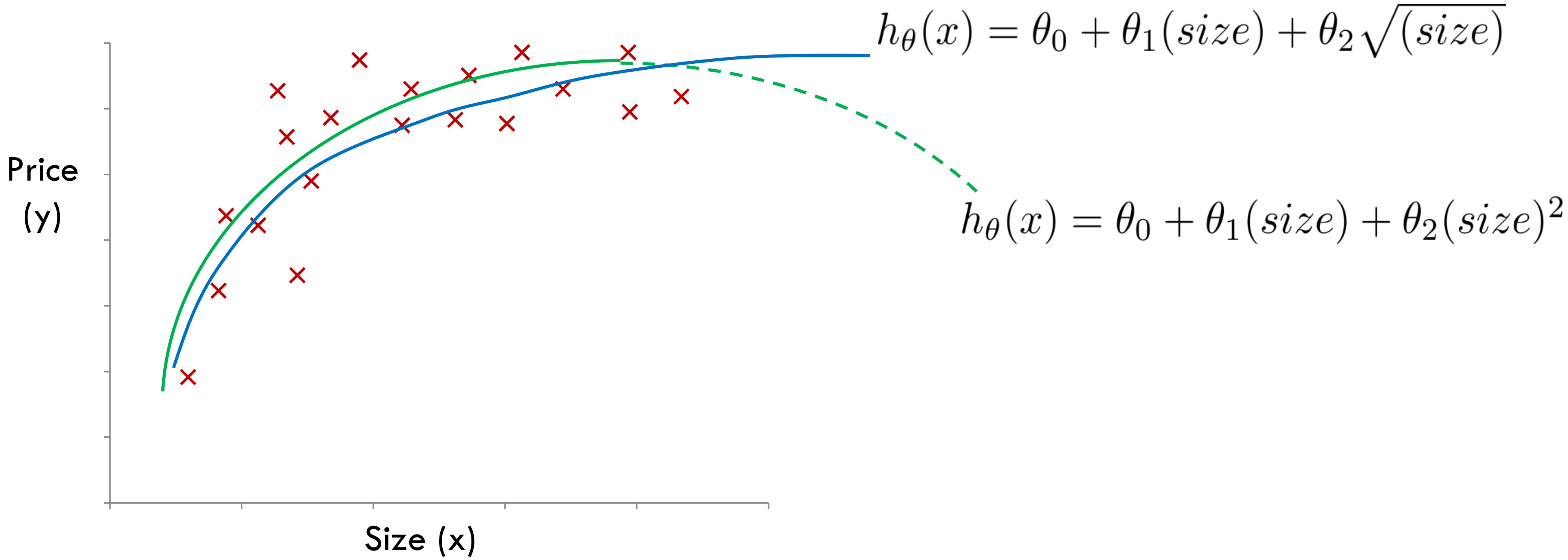
Gradient descent updates move in small zig-zag paths across these elongated ellipses → **slow convergence**.


Highly Anisotropic Quadratic Cost (Thin Elliptical Contours)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
θ_1
θ_2
5.000
2.721
1.481
0.806

2. LEARNING RATE IMBALANCE

- A single learning rate α work well for all features:
- If α is large enough for small-scale features, it overshoots for large-scale features.
- If α is safe (small enough) for large-scale features, it makes tiny, painfully slow updates for small-scale ones.

CHOICE OF FEATURES



EXAMPLE PYTHON IMPLEMENTATION

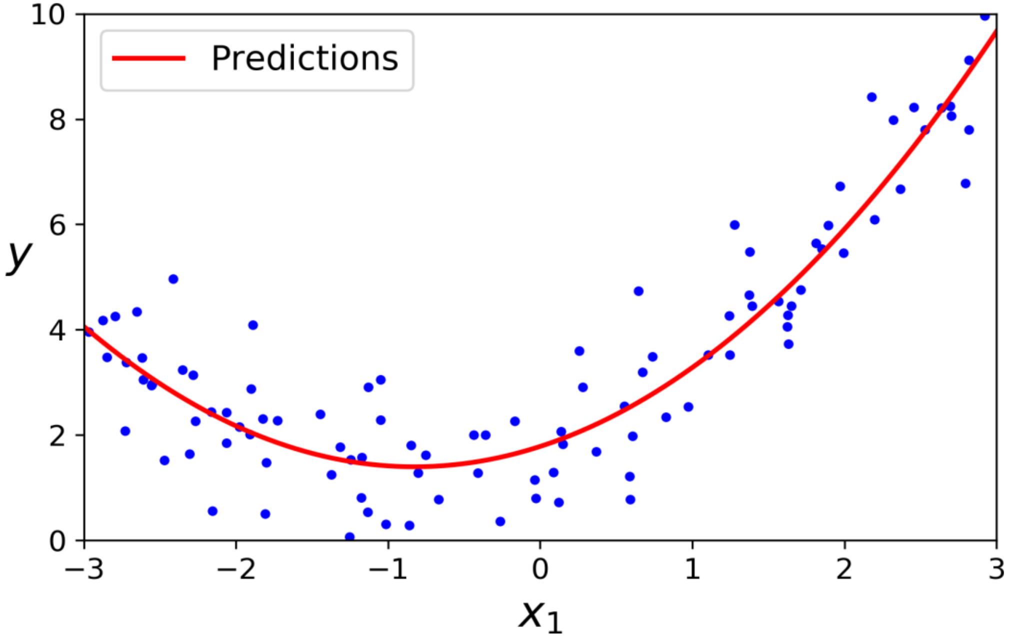
- Sklearn PolynomialFeatures() generates polynomial and interaction features.
- Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree.
- For example, if an input sample is two dimensional and of the form [a, b],
  - the degree-2 polynomial features are $[1, a, b, a^2, ab, b^2]$.
  - the degree-3 polynomial features are $[1, a, b, ab, a^2, b^2, ab^2, a^2b, a^3, b^3]$.

EXAMPLE

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2, include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929, 0.56664654])
```

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```



NORMAL EQUATION METHOD

NORMAL EQUATION

- So far we've been using gradient descent
  - Iterative algorithm which takes steps to converge
- For some linear regression problems the normal equation provides a better solution
- Normal equation solves θ analytically:
  - a closed-form solution used to find the value of θ that minimizes the cost function.
  - i.e., solve for the optimum value of theta in one step
- It has some advantages and disadvantages

EXAMPLE

$m = 4$

x_0	Size (feet²) x_1	# of bedrooms x_2	# of floors x_3	Home age (yrs) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$m \times (n+1)$ $\quad$ $m \times 1$

- The value of theta that minimizes the cost function is $\theta = (X^T X)^{-1} X^T y$

Note: The derivation of the normal equation is beyond the scope of this course.

For interested students, this site provides a simple explanation: https://prutor.ai/normal-equation-in-linear-regression/

MORE GENERALLY ..

- m examples $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$, and n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} \dots & x_n^{(2)} \\ x_0^{(3)} & x_1^{(3)} \dots & x_n^{(3)} \\ \dots & & \\ x_0^{(m)} & x_1^{(m)} \dots & x_n^{(m)} \end{bmatrix}$$

← Design matrix

$$\theta = (X^T X)^{-1} X^T y$$

NORMAL EQUATION

$$\theta = (X^T X)^{-1} X^T y$$

- $(X^T X)^{-1}$ is inverse of matrix $X^T X$
- You can use the inv() function from NumPy's Linear Algebra module (np.linalg) to compute the inverse of a matrix, and the dot() method for matrix multiplication:

```
X_b = np.c_[np.ones((100, 1)), X]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

- Feature scaling: is not necessary when using the normal equation method
- $0 < x_1 < 1$
- $0 < x_2 < 10000$

OK

ADVANTAGES AND DISADVANTAGES

Gradient Descent

- Need to choose α
- Needs many iterations
- Works well even when the number of features is large

Normal Equation

- No need to choose α
- Don't need to iterate
- Need to compute $(X^T X)^{-1}$
  - $X^T X$ is an $n \times n$ matrix, computing its inverse is $O(n^3)$
- Slow if the number of features is very large

$n = 100$, ok

$n = 1000$, ok

$n = 10000$, meh, not so good

NORMAL EQUATION AND NON-INVERTIBILITY

Normal equation $\theta = (X^T X)^{-1} X^T y$

- What if $X^T X$ is non-invertible? (singular/ degenerate matrices)
- The *pseudoinverse* of **X** (specifically the Moore-Penrose inverse). You can use np.linalg.pinv()

NORMAL EQUATION AND NON-INVERTIBILITY

What if $X^T X$ is non-invertible?

1. Check for Redundant features (linearly dependent).

   - E.g. $x_1 =$ size in feet^2

     $x_2 =$ size in m^2

2. Too many features (e.g. $m < n$).

   - Delete some features, or use regularization.