

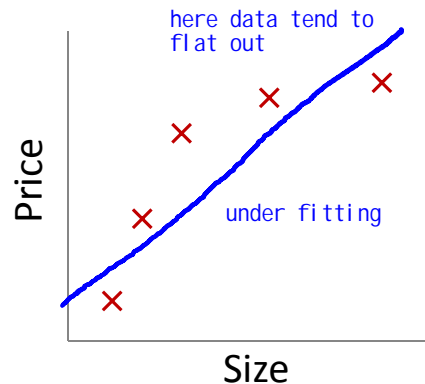


Machine Learning

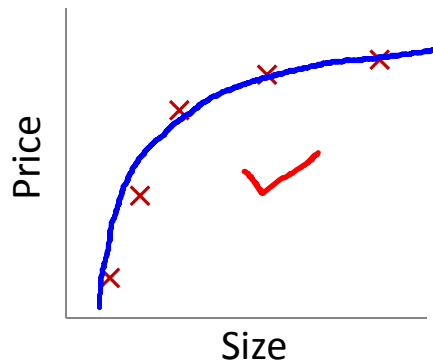
Regularization

The problem of overfitting

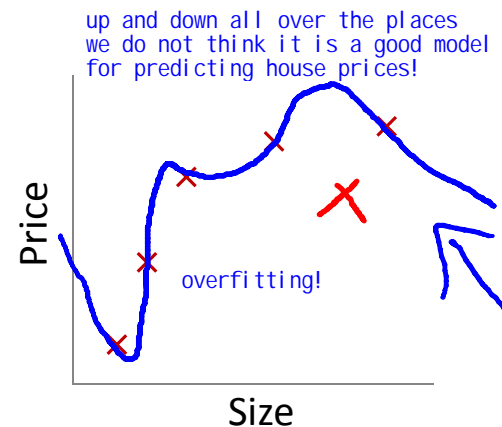
Example: Linear regression (housing prices)



→ $\theta_0 + \theta_1 x$
 "Underfit" "High bias"



→ $\theta_0 + \theta_1 x + \theta_2 x^2$
 "Just right"



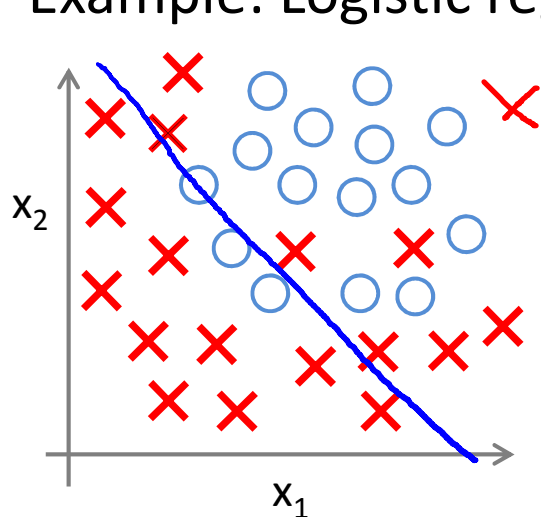
→ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$
 "Overfit" "High variance"

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

→ predict!

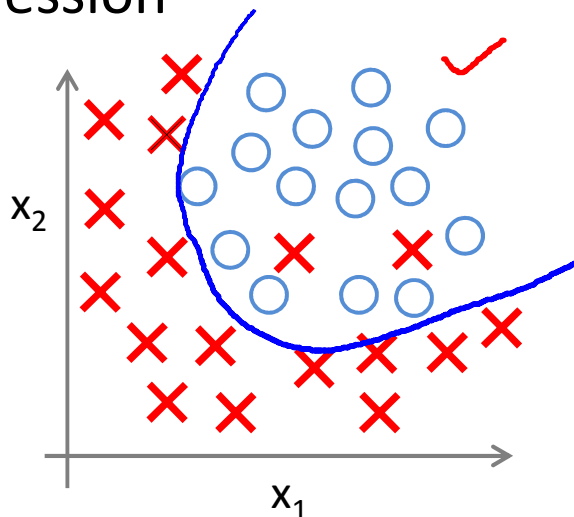
overfitting examples in logistic regression!

Example: Logistic regression



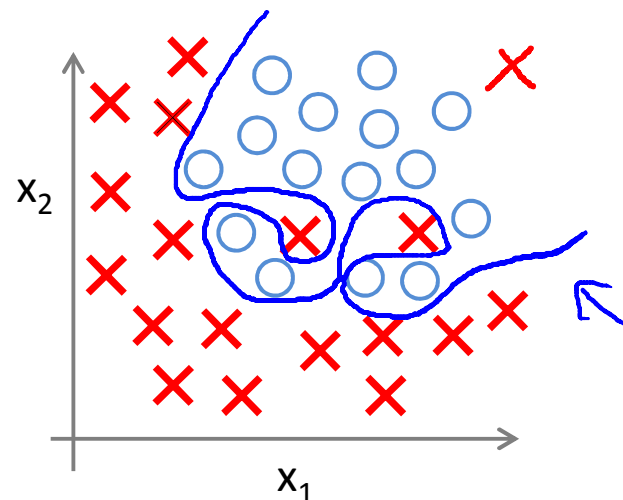
$\rightarrow h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
(g = sigmoid function)

"Underfit"



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2$
 $+ \theta_3 x_1^2 + \theta_4 x_2^2$
 $+ \theta_5 \underline{x_1 x_2})$

this is just right



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2$
 $+ \theta_3 \underline{x_1^2 x_2} + \theta_4 \underline{x_1^2 x_2^2}$
 $+ \theta_5 \underline{x_1^2 x_2^3} + \theta_6 \underline{x_1^3 x_2} + \dots)$

"Overfit"

hypothesis has high variance!

Addressing overfitting: what do we do when overfitting happens

x_1 = size of house

x_2 = no. of bedrooms

x_3 = no. of floors

x_4 = age of house

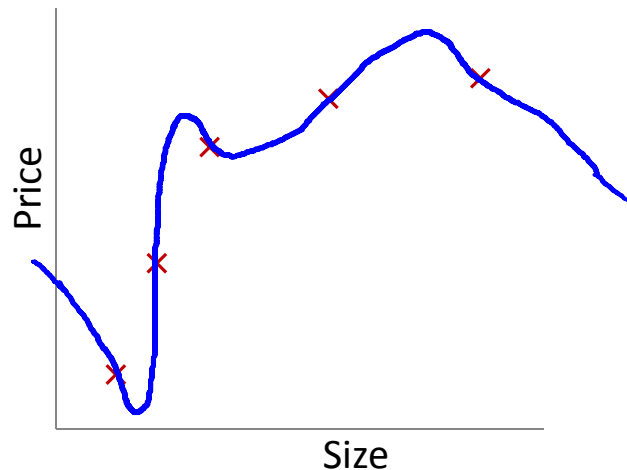
x_5 = average income in neighborhood

x_6 = kitchen size

⋮

However, when we have so many features,
it becomes harder to plot the data and
visualize them!

x_{100}



one way to address the overfitting
is to plot the hypothesis function
and look at its shape!

Addressing overfitting: two main options to address overfitting problem

Options:

1. Reduce number of features.

→ — Manually select which features to keep.

→ — Model selection algorithm (later in course).

2. Regularization. 正则化

→ — Keep all the features, but reduce magnitude/values of parameters θ_j .

— Works well when we have a lot of features, each of which contributes a bit to predicting y .



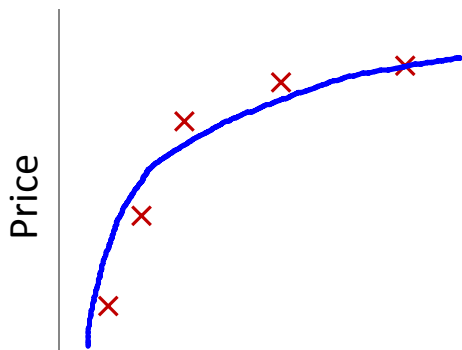
Machine Learning

for dealing with overfitting!

Regularization

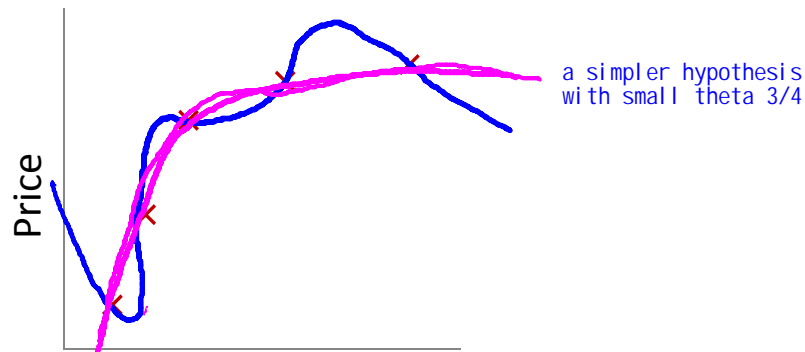
Cost function

Intuition



Size of house

$$\theta_0 + \theta_1 x + \theta_2 x^2$$



Size of house

a simpler hypothesis
with small theta 3/4

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$

Suppose we **penalize** and make θ_3, θ_4 really small.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + 1000 \theta_3^2 + 1000 \theta_4^2$$

we minimize this new cost function
so that we can make theta 3,4 very small

$$\theta_3 \approx 0$$

$$\theta_4 \approx 0$$

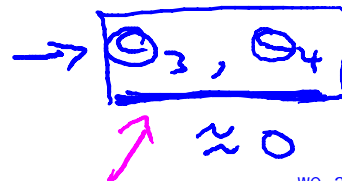
Regularization.

main ideas behind regularization!

Small values for parameters $\theta_0, \theta_1, \dots, \theta_n$

- “Simpler” hypothesis
- Less prone to overfitting

e.g. we penalize these two parameters to make them very small!



an example!

Housing:

- Features: x_1, x_2, \dots, x_{100}
- Parameters: $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

we do not know which to pick to make small parameters

we add extra regularization term to shrink all the parameters

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

Diagram showing the regularization term $\lambda \sum_{j=1}^n \theta_j^2$ in a red box, with a pink arrow pointing to it and the text ≈ 0 below.

Diagram showing parameters $\theta_1, \theta_2, \theta_3, \dots, \theta_{100}$ in a red box, with a pink arrow pointing to them and the text ≈ 0 below.

by convention we do not regularize θ_0

Regularization.

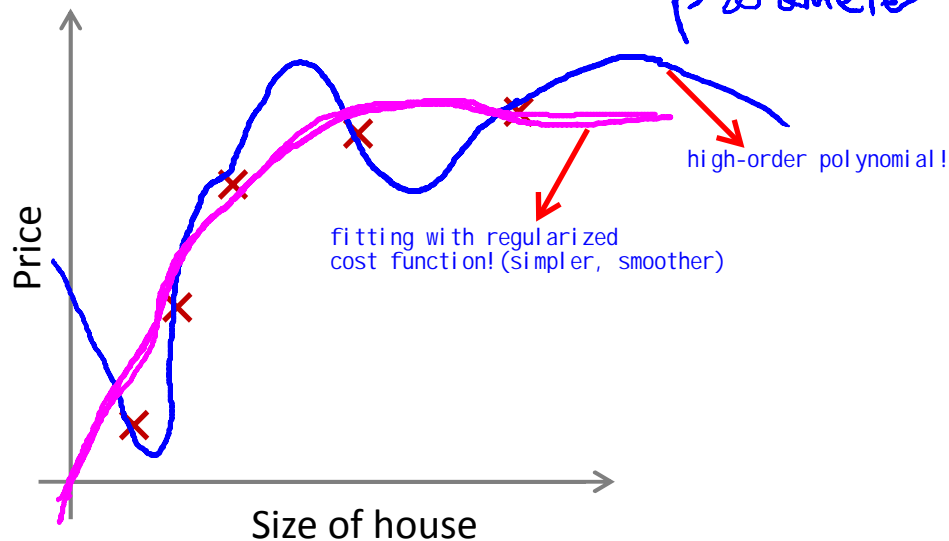
regularized cost function

$$\rightarrow J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

controls a tradeoff b/c 2 diff goals:
1. fitting the training data well
2. want to keep parameter small, therefore keep the hypothesis simple to avoid overfitting!

$$\min_{\theta} J(\theta)$$





regularization parameter



In regularized linear regression, we choose θ to minimize

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

What if λ is set to **an extremely large value** (perhaps far too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm works fine; setting λ to be very large can't hurt it 
- Algorithm fails to eliminate overfitting. 
- Algorithm results in underfitting. (Fails to fit even training data well). 
- Gradient descent will fail to converge. 

In regularized linear regression, we choose θ to minimize

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

What if λ is set to an extremely large value (perhaps far too large for our problem, say $\lambda = 10^{10}$)?

then we will have

$$\theta_1, \theta_2, \theta_3, \theta_4$$

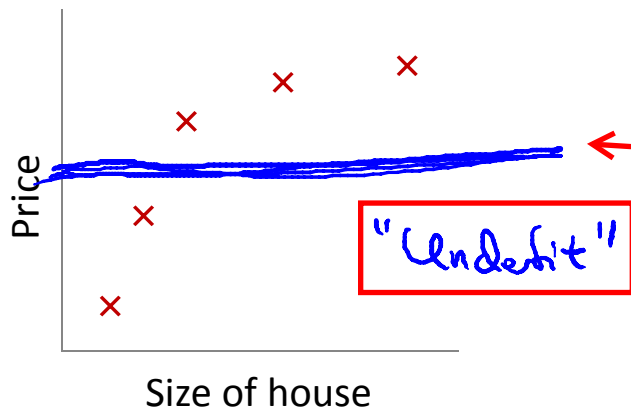
$$\theta_1 \approx 0, \theta_2 \approx 0$$

$$\theta_3 \approx 0, \theta_4 \approx 0$$

as a result

$$h_{\theta}(x) \approx \theta_0$$

almost flat line!



$h_{\theta}(x)$

$$\theta_0 + \cancel{\theta_1 x} + \cancel{\theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$



Machine Learning

Regularization

Regularized linear regression

two algorithms:
1. gradient descent
2. normal equation

Regularized linear regression

objective! $\left\{ \right.$

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \underbrace{\lambda \sum_{j=1}^n \theta_j^2}_{\text{regularization parameter}} \right]$$

$\min_{\theta} \underline{J(\theta)}$

↑

Gradient descent

we do not penalize θ_0 in regularization

$$\theta_0$$

$$\theta_1, \theta_2, \dots, \theta_n$$

we write θ_0 separately from θ_1, \dots since in the regularization term we do not include θ_0

Repeat {

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

$$\frac{\partial}{\partial \theta_0} J(\theta)$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

$(j = 1, 2, 3, \dots, n)$

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

$$1 - \alpha \frac{\lambda}{m} < 1$$

$$0.99$$

$$\theta_j \times 0.99$$

same as the original gradient descent algorithm

$$\theta_j^2$$

Normal equation

m training samples

$$\underline{X} = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

m x (n+1)

$$\underline{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

\mathbb{R}^m

$$\rightarrow \min_{\theta} \underline{J(\theta)}$$

$$\frac{\partial}{\partial \theta_j} J(\theta) \stackrel{\text{set}}{=} 0$$

use this to derive this formula

$$\Rightarrow \underline{\Theta} = \left(X^T X + \lambda \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{(n+1) \times (n+1)} \right)^{-1} X^T y$$

E.g. n=2 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Non-invertibility (optional/advanced).

Suppose $m \leq n$, \leftarrow
(#examples) (#features)

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

$\underbrace{\hspace{10em}}_{\text{non-invertible / singular}}$

pinv

inv
 \nwarrow

we do not penalize θ_0

Can and has be easily proved!

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

invertible

. This can be proved as always invertible! Refer to notes



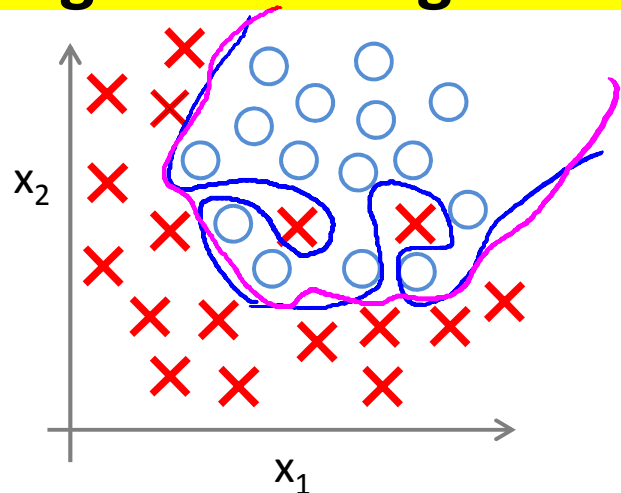
Machine Learning

Regularization

Regularized

logistic regression

Regularized logistic regression.



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Cost function:

$$\rightarrow J(\theta) = - \left[\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

add a regularization term!

$\theta_1, \theta_2, \dots, \theta_n$

Gradient descent

even though the algorithm looks identical as in the linear regression case the hypothesis function is different in the logistic regression case!

Repeat {

$$\begin{aligned} \rightarrow \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \\ \rightarrow \theta_j &:= \theta_j - \alpha \left[\underbrace{\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}}_{(j = \cancel{0}, 1, 2, 3, \dots, n)} + \frac{1}{n} \theta_j \right] \end{aligned}$$

Annotations:

- Red arrow from $x_0^{(i)}$ to "update for theta_0"
- Red arrow from $\frac{1}{n} \theta_j$ to "new term!"
- Red box around the entire update for θ_j
- Blue arrow pointing to the left of the bracket in the θ_j update
- Red arrow from the closing brace of the repeat loop to "update for theta_1, ... n"
- Blue bracket under the summation term in the θ_j update, labeled $\theta_1, \dots, \theta_n$

$$\frac{\partial}{\partial \theta_j} J(\theta)$$

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Advanced optimization

minimize (cost function)

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$
 $\theta_0 \leftarrow \text{theta}(1)$
 $\theta_1 \leftarrow \text{theta}(2)$
 $\theta_n \leftarrow \text{theta}(n+1)$

function [jVal, ^{cost value} gradient] = costFunction(^{input} theta)

jVal = [code to compute $J(\theta)$];

need to return the cost function and gradient!

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \boxed{\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2}$$

regularization term!

gradient(1) = [code to compute $\frac{\partial}{\partial \theta_0} J(\theta)$];

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

gradient(2) = [code to compute $\frac{\partial}{\partial \theta_1} J(\theta)$];

$$\left(\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)} \right) + \frac{\lambda}{m} \theta_1$$

gradient(3) = [code to compute $\frac{\partial}{\partial \theta_2} J(\theta)$];

$$\vdots \left(\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)} \right) + \frac{\lambda}{m} \theta_2$$

gradient(n+1) = [code to compute $\frac{\partial}{\partial \theta_n} J(\theta)$];

$J(\theta)$

