Coursera Machine Learning

[Welcome lecture]

- ML as a field grew out of AI, when it became clear that for some problems you can't write a program by hand. (eg. autonomous helicopters)
- Examples of everyday ways you use ML
  - search ranking
  - automatic photo tagging
  - spam filtering
  - database mining (eg. web click data)
  - self-customising programs

What is Machine Learning?

Defn according to Arthur Samuel:
Field of study that gives computers the ability to learn without being explicitly programmed

Later defn:

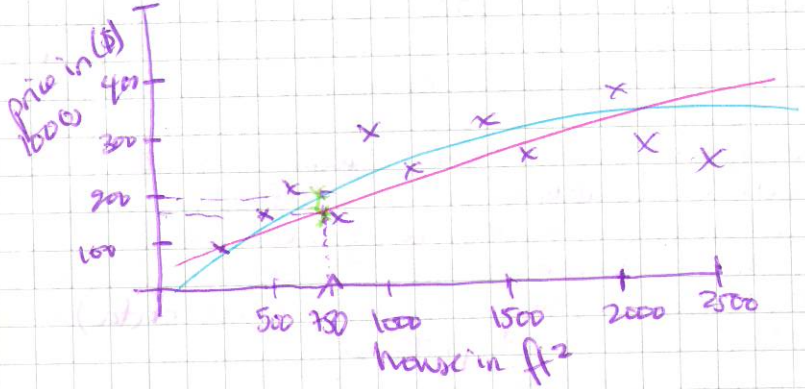
A well Posed Learning Problem: A computer program is said to learn from experience E wrt some task T and some performance measure P if its performance on T, as measured by P, improves with experience E.

Leigh-Anne Mathieson · LMATHIE@GMAIL.COM
· 07531373984

What is Supervised Learning?

Let's start with an example

e.g. Housing Price Prediction



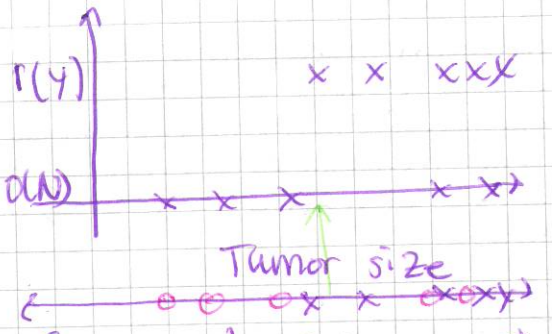
If given data on how much houses (by size) have sold for in the past, can we predict how much a 750 sq ft house will sell for?

This problem is an example of supervised learning because there are some "right answers" to work with (ie. existing data)

This is also known as a regression problem because it is predicting continuous valued output (the price)

What about a ~~more complicated~~ different example?

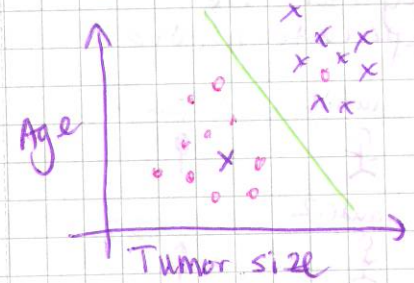
Breast Cancer Classification



Say you have some past patient data (their tumor size and whether benign or malignant). Now you have a new patient & corresponding tumor size. Can we predict if benign or malignant?

This is an example of classification (predicting discrete valued output)

But many real world problems have many features. We may still want to use classification.



In breast cancer research, also interested in
- clump thickness
- uniformity of cell size
- uniformity of cell shape
- etc.

Support Vector Machines allow an infinite number of features.

What is unsupervised learning?

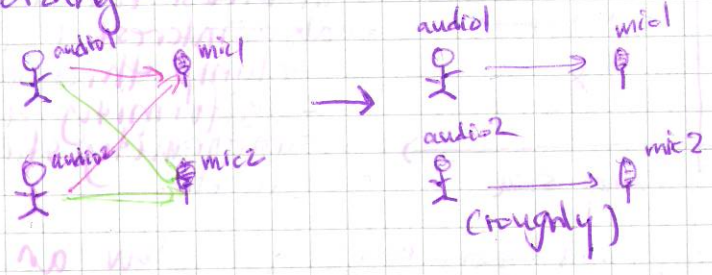
- No historical data with right/wrong classification
- Given ~~unlabeled~~ data, algorithm may seek to find structure e.g. clustering
- Everyday example: Google news & other news aggregators

clustering algorithm

~~research example~~
~~DNA individuals~~
- social network analysis
- market segmentation
- astronomical data analysis

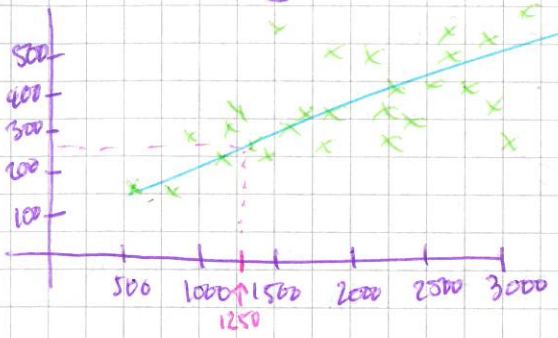
→ we don't know in advance what the segments are!

Research example:
Cocktail Party Problem
Algorithm that tries to find structure in overlapping sounds/voices ~~(how?)~~ so that they can be separated out into separate audio recording.



Model representation: Linear Regression

Back to housing prices in Portland...



More formally, we have a training set

Size in ft^2 (x)	price in 1000's ($) ($y$)
2104	460
1416	232
1534	315
852	178
...	...

m = number of training examples
x's = "input" variable/features
y's = "output" variable/features

$m = 47$

(x, y) - one training example
$(x^{(i)}, y^{(i)})$ - i^{th} training example

$x^{(1)} = 2104$
$x^{(2)} = 1416$
$y^{(1)} = 460$

Model:

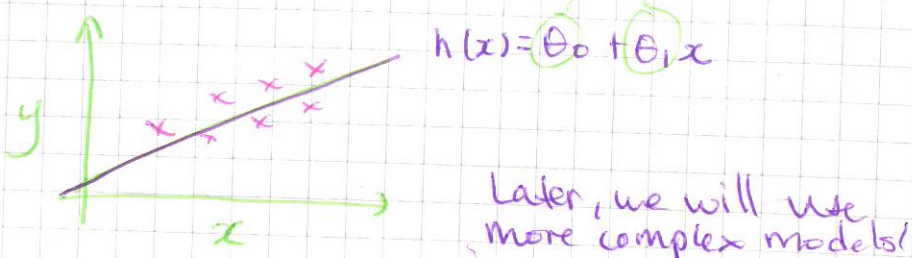
h maps from x's to y's.

How do we represent h?

Here, for regression

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

parameters

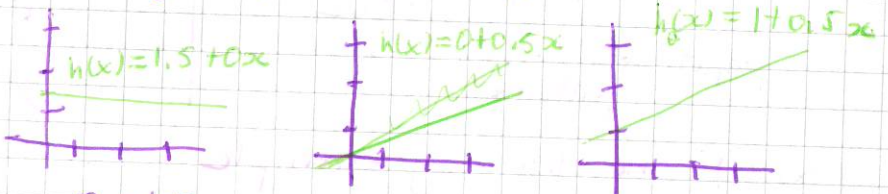


Later, we will use more complex models!

This one is linear regression with one variable aka univariate linear regression.

How to choose our 2 parameters?

Let's try a few values...



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

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

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

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

More formally:

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

As a cost function:

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

$$\underset{\theta_0, \theta_1}{\text{minimize}} \ J(\theta_0, \theta_1)$$

$J(\theta_0, \theta_1)$ is aka squared error function

Recap: we want to fit a straight line through our data and

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

Parameters: θ_0, θ_1

Cost function: ~~$J(\theta_0, \theta_1) = (h_\theta(x^{(i)}) - y^{(i)})^2$~~

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

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}} \ J(\theta_0, \theta_1)$

Let's consider a simplified version

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

parameter: θ_1

$\underset{\theta_1}{\text{minimize}} \ J(\theta_1)$

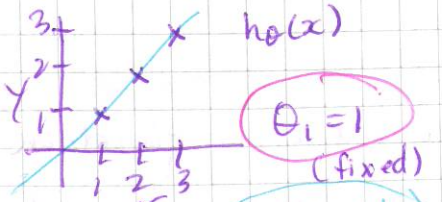
$\theta_0 = 0$

only consider hypothesis functions that pass through the origin

Two functions to understand

$h_\theta(x)$

(for fixed θ_1, this is a function of x)



training set

x	y
1	1
2	2
3	3

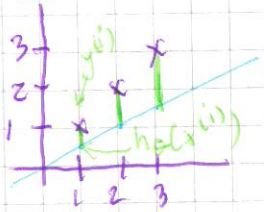
when $\theta_1 = 1$

$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\theta_1 x^{(i)} - y^{(i)})^2$$

$$= \frac{1}{2m}(0^2+0^2+0^2) = 0$$

So $J(1) = 0$

$\theta_1 = 0.5$



$$J(\theta_1) = J(0.5)$$

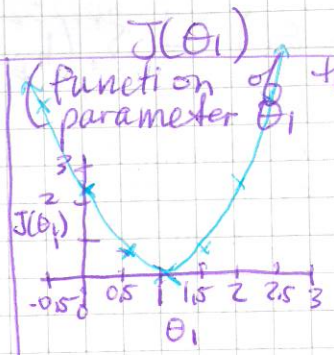
$$= \frac{1}{2m}\sum_{i=1}^{m}(\theta_1 x^{(i)} - y^{(i)})^2$$

$$= \frac{1}{2m}(0.5^2 + 1^2 + 1.5^2)$$

$$= \frac{3.5}{6} \approx 0.58$$

$J(\theta_1)$

(function of the parameter θ_1)



Each value of θ_1 corresponds to a different hypothesis function

We want to pick the θ_1 where

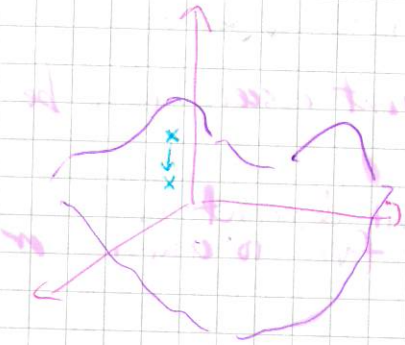
minimize $J(\theta_1)$
θ_1

Gradient Descent

Idea:
We have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:
- start with some θ_0, θ_1 (say, $\theta_0 = 0$, $\theta_1 = 0$) but it doesn't really matter
- keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum



Idea: Imagine a grassy park where you are standing on a hill and want to decrease your elevation ASAP. So you look around 360° and decide where to take your next step. Repeat.

Gradient descent does not guarantee global minimum. Only a local min!

Gradient descent algorithm

repeat until convergence {

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

learning rate = size of step

}

(means update both θ_0 and θ_1 simultaneously)

Correct implementation

In impl: simultaneous update

$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

$\theta_0 := temp0$

$\theta_1 := temp1$

Picking α:

- If α is too small, gradient descent can be slow
- If α is too large, gradient descent can overshoot the minimum, fail to converge or even diverge!

Gradient Descent For Linear Regression

To apply to our linear regression model we need to know:

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

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

Need to know for

θ_0 $j=0$: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

θ_1 $j=1$: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$

Now we have gradient descent for linear regression

repeat until convergence {

$\theta_0 := \theta_0 - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)\right]$ $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\theta_1 := \theta_1 - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}\right]$ $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

}

Note:

Though gradient descent generally doesn't guarantee a global optimum, linear regression always generates a convex (bowl-shaped) function and thus there will always be a global optimum

"Batch" Gradient Descent: each step of gradient descent uses all the training examples.

Linear Regression with Multiple Features

Let's return to our question about housing price prediction.

Size (ft^2) x_1	# of bedrooms x_2	# of floors x_3	Age of home (yrs) x_4	Price (Cin $1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$m = 47$

More notation:

n = number of features
$x^{(i)}$ = input (features) of i^{th} training example
$x_j^{(i)}$ = value of feature j in i^{th} training example

e.g. $x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix} \in \mathbb{R}^4$ $\quad x_3^{(2)} = 2$

How do multiple features affect the hypothesis?

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

Now: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

e.g. $h_\theta(x) = 80 + 0.1x + 0.01x_2 + 3x_3 - 2x_4$

More generally:

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

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

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$ $\quad x_0 = 1$ (for convenience of notation)

$= \theta^T x$ $\quad \left[= \begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \right]$ $\quad \theta^T$

We can use the fancy term multivariate linear regression.

Gradient Descent For Multiple Variables

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

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$ / θ where θ is a $n+1$ dimensional vector

Cost function:

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

$J(\theta)$

Gradient Descent

Repeat until convergence {

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

} (simultaneously update for every $j = 0, \ldots, n$)

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

New Gradient Descent 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 $j = 0, \ldots, n$)

}

So...

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

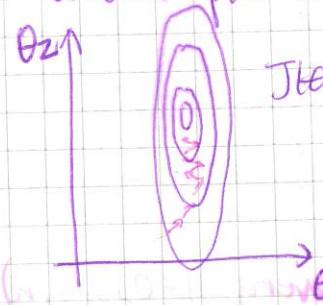
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

Feature Scaling

Idea: If you make sure features are on a similar scale, gradient descent will converge faster.

ex: x_1 = size (0-2000 ft^2)
x_2 = number of bedrooms (1-5)

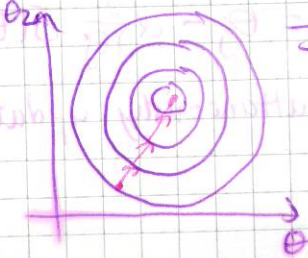
Contour plot might look like



$x_1 = \frac{\text{size } (ft^2)}{2000}$

$x_2 = \frac{\#\text{ of bedrooms}}{5}$

so then $0 \leq x_1 \leq 1$, $0 \leq x_2 \leq 1$



Goal of feature scaling:

Get every feature into approximately a a bit $-1 \leq x_i \leq 1$ range but this is arbitrary

$0 \leq x_1 \leq 3$ ✓
$-2 \leq x_2 \leq 0.5$ ✓
$-100 \leq x_3 \leq 100$ ✗
$-0.0001 \leq x_4 \leq 0.0001$ ✗

different rules of thumb, but $3 \leq x_i \leq -3$ def reasonable

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features approx zero mean

e.g. $x_1 = \frac{\text{size} - 1000}{2000}$ $\quad x_2 = \frac{\#\text{ bedrooms} - 2}{5}$

$-0.5 \leq x_1 \leq 0.5$ $\quad -0.5 \leq x_2 \leq 0.5$

Generally:

$$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$$

μ_1: avg value of x in training set

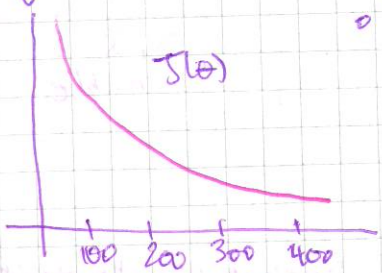
s_1: range (max-min) or standard deviation

Learning Rate

Making sure gradient descent is working

GDCP

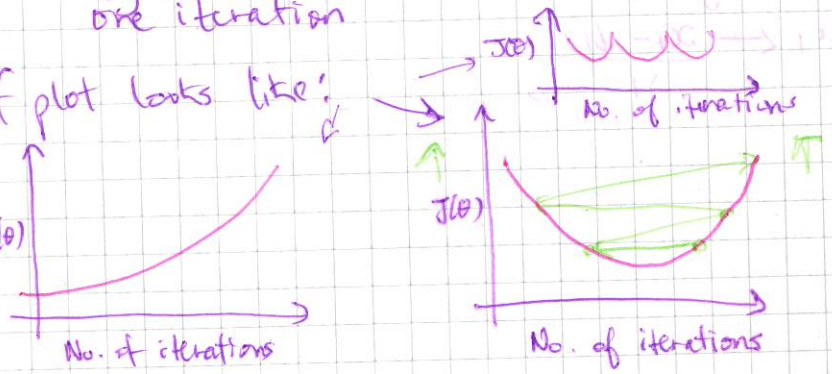
$\min_\theta J(\theta)$



- value of cost function after each iteration of gradient descent
- $J(\theta)$ should decrease after every iteration.
- No. of iterations can vary depending on application but this trend should hold.

Example of automatic convergence test:
Declare convergence if $J(\theta)$ decreases by less than some ε [e.g. 10^{-3}] in one iteration

If plot looks like:



Gradient descent is not working!

Use a smaller α. Choice of α is probably causing us to overshoot the minimum.

- For sufficiently small α, $J(\theta)$ should decrease on every iteration
- But if α is too small, gdvs gradient descent will be slow.

To choose α, try

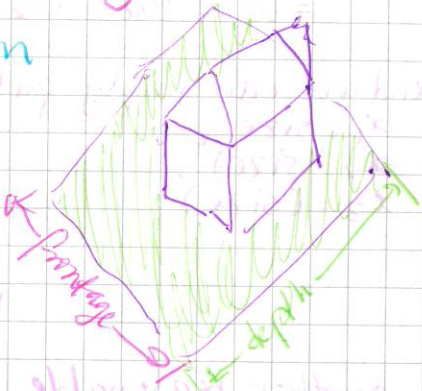
..., 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, ...

~3x ~3x ~3x ...

until you find which in the range is too large and which is too small and then try to pick largest value possible that will cause convergence in that range.

Features and Polynomial Regression

Housing Prices Prediction



$h_\theta(x) = \theta_0 + \theta_1 \times \underbrace{\text{frontage}}_{x_1} + \theta_2 \times \underbrace{\text{depth}}_{x_2}$

May want to instead define new feature

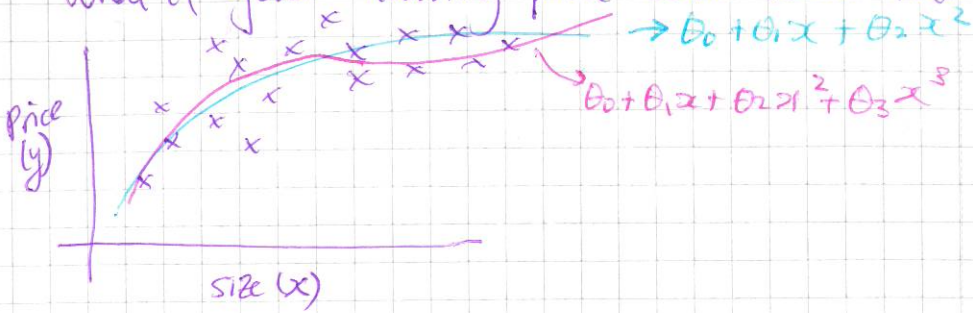
Area: $x = \text{frontage} \times \text{depth}$

$h_\theta(x) = \theta_0 + \theta_1 x$ ← land area.

No need to stick with the features you started out with!

Polynomial Regression

What if your housing price data looks like



It may be better fit by a polynomial.

We're familiar with hypotheses of the form

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

$$= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3$$

and we can simply pick

$x_1 = \text{size}$ — size: 1 - 1000

$x_2 = (\text{size})^2$ — $(\text{size})^2$: 1 - 100,000

$x_3 = (\text{size})^3$ — $(\text{size})^3$: 1 - 10^9

feature scaling even more important in this case!

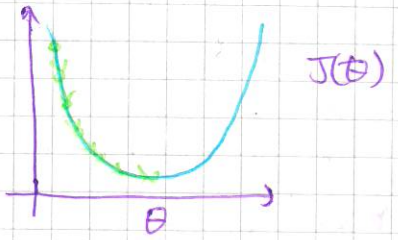
another reasonable hypothesis might be

$$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{\text{size}}$$

Later, we'll cover algorithms that will automatically decide which features to use.

Normal Equation

Gradient descent



Normal equation: Method to solve for θ analytically.

Some intuition: If 1D ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$

$$\frac{d}{d\theta} J(\theta) = \cdots \overset{\text{set to}}{=} 0$$

Solve for θ

But in our case, θ is not simply a real number. $\theta \in \mathbb{R}^{n+1}$

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

$$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots \overset{\text{set}}{=} 0 \quad \text{(for every } j\text{)}$$

Solve for $\theta_0, \theta_1, \ldots \theta_n$

But derivation is somewhat involved.

Examples $m = 4$

x_0	size (feet^2) x_1	# of bedrooms x_2	# floors x_3	Age of home (years) 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

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

dim: $m \times (n+1)$ $\qquad$ m (vector)

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

More generally,

m examples $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$;

n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \qquad X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

(design matrix)

$m \times (n+1)$ dim ~~vector~~ matrix.

simple example: If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_2^{(1)} \\ \vdots & \vdots \\ 1 & x_m^{(1)} \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

dim: $m \times 2$

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

$A = X^T X$

$(X^T X)^{-1} = A^{-1}$

in octave: `pinv(x'*x)*x'*y`

If using normal equation, feature scaling is not necessary.

Comparing to gradient descent:

Gradient descent	Normal Equation
• Need to choose α	• No need to choose α
• Needs many iterations	• Don't need to iterate
• Works well even when n is large	• Need to compute $(X^T X)^{-1}$ ($n \times n$ matrix)
	Slow if n is very large! Costs $\sim O(n^3)$

$n = 100$

$n = 1000$

$n = 10{,}000$ (maybe).

Normal Equation Noninvertability

When computing

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

what if $X^T X$ is non-invertible (singular/ degenerate)

if you use pinv (pseudo-inverse) (X' * X), you will still get the right answer

How a non-invertible $X^T X$ comes about:

- Redundant features (linearly dependent)
  e.g. x_1 = size in feet^2
  x_2 = size in m^2

  $x_1 = (3.28)^2 x_2$

  When you have linearly dependent features, your matrix will be non-invertible
- Too many features ($m \leq n$)
  - Delete some features, or use regularization

end of week 2

Octave Tutorial

- size(A) gives 1×2, [#rows #cols]
  length(v) gives longest dimension
- load data into octave
  e.g. load('features x*.dat')
  load features x.dat
- who gives variables in current scope
  whos is more detailed
- clear remove variable from scope
- save variables into file
  eg. save hello.mat v
  save hello.txt v -ascii
  human-readable

how to:
append column
e.g. A = [A, [COL VECTOR]];
flatten: A(:)

Multiplication:

A * B → matrix multiplication
A .* B → element wise multiplication

max [val, ind] = max(A)

returns value that is max & its index

sum sum(a) sums all element of vector

prod(a) as expected

sum(A,1): sum of each column

sum(A,2): sum of each row.

pinv pinv(A) gives inverse of A.

Plotting

eg. t = [0:0.1:0.98]

y1 = sin(2*pi*4*t)

y2 = cos(2*pi*4*t)

[plot both on same graph, one in red

```
plot(t,y1);
hold on
plot(t,y2,'r');
xlabel('time');
ylabel('value');
legend('sin','cos')
title('my plot')
```

Save: print -dpng 'myplot.png'.

```
hold off
close (close figure)
```

can specify figure numbers
figure(1) ...

subplot(1,2,1) % divides plot into 1×2 grid & access first elem

change scale of axes with axis
eg. axis([0.5 1 -1 1])

clf clear figure

imagesc(A) plots A matrix as a variation of colour depending on value

grayscale heatmap:
imagesc(A); colorbar; colormap gray;

" comma chaining commands "
eg a=1, a=2, a=3 will execute all 3,
a=1; a=2; a=3

for i=1:10
while i<=5 } all need an end.
if v(1)==1

functions

create a new file e.g. squarethis.m

takes one param

```
function y = squareThisNumber(x)
y = x^2
```

will return one value saved in y

to access the function, it needs to be in working directory.

to add directory to Octave's search path, use search path

to return multiple arguments, e.g.

```
function [y1, y2] = squareAndCube(x)
y1 = x^2
y2 = x^3
```

to call:

```
[a, b] = squareAndCube(x)
```

Vectorization

e.g.

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

$$= \theta^T x \quad \text{where } \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

unvectorised implementation

```
prediction = 0.0
for j = 1:n+1,
  prediction = prediction + theta(j) * x(j)
end;
```

vectorized implementation

```
prediction = theta' * x
```

eg. gradient descent

$$\theta := \theta - \alpha \delta \qquad \theta, \delta \in \mathbb{R}^{n+1}, \ \alpha \in \mathbb{R}$$

$$\text{where } \delta = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

$h_\theta(x^{(i)}) - y^{(i)} \in \mathbb{R}$, $x^{(i)} \in \mathbb{R}^{n+1}$

$$\delta = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix} \qquad \delta_0 = \frac{1}{m} \sum \left(h_\theta(x)^{(i)} - y^{(i)} \right) x_0^{(i)}$$

Classification: Logistic Regression

onto the discrete values!

For now, y can take on 2 values

$y \in \{0, 1\}$

- 0: "Negative class" (eg. benign tumor)
- 1: "Positive class" (eg. malignant tumor)

We can't use linear regression as data points like on far right are not outliers but still influence fit

also: Classification $y=0$ or 1

but $h_\theta(x)$ can be >1 or <0

Hypothesis representation

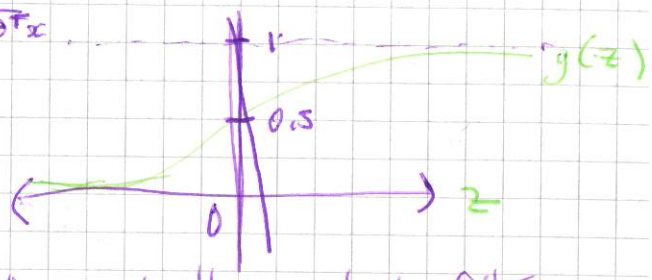
Logistic Regression Model

Want $0 \leq h_\theta(x) \leq 1$

$h_\theta(x) = g(\theta^T x)$

where $g(z) = \frac{1}{1+e^{-z}}$ } sigmoid function / logistic function

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



As before, will eventually need to fit parameters θ

Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that $y=1$ on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorsize} \end{bmatrix}$

$h_\theta(x) = 0.7$

Tell patient 70% chance of tumor being malignant

Or more formally, $h_\theta(x) = P(y=1 \mid x; \theta)$

"probability that $y=1$, given x, parameterised by θ"

We can compute the probability for $y=1$ or 1.

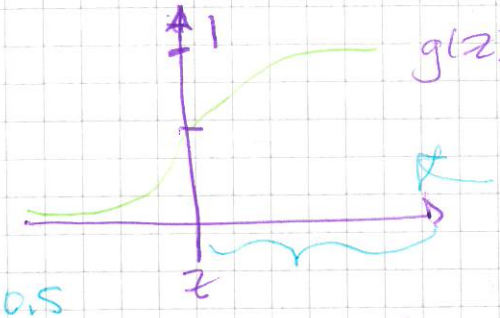
$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$$

The Decision Boundary

but when to pick $y=0$ / $y=1$

Suppose predict "$y=1$" if $h_\theta(x) \geq 0.5$

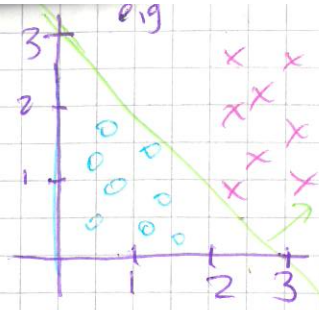
"$y=0$" if $h_\theta(x) < 0.5$



$g(z) \geq 0.5$ when $z \geq 0$

So, since $h_\theta(x) = g(\theta^T x) \geq 0.5$ whenever $\theta^T x \geq 0$

likewise $y=0$ predicted when $\theta^T x < 0$



$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

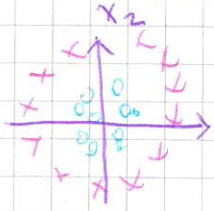
Decision boundary where $h_\theta(x) = 0$

When would this predict $y=1$?

If $\underbrace{-3 + x_1 + x_2}_{\theta^T x} \geq 0$

$x_1 + x_2 \geq 3$

Non-linear decision boundaries



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

Predict $y=1$ if $-1 + x_1^2 + x_2^2 \geq 1$

Logistic Regression: Cost Function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})\}$

with m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$ $x_0 = 1$ $y = \{0, 1\}$

$\mathbb{R}^{n+1}$

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

So how do we choose parameters θ

For linear regression

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

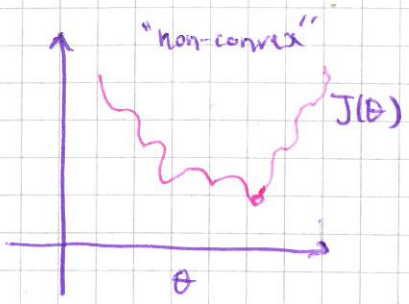
Can change notation

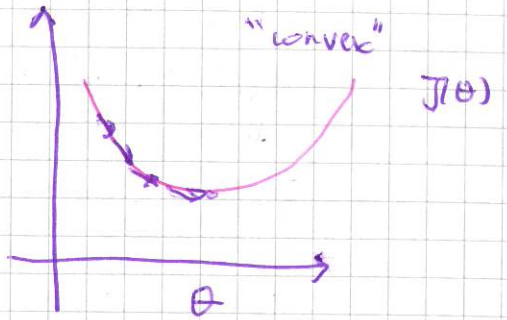
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x^{(i)}), y)$$

where $\text{Cost}(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$

called the cost because it's the price paid by the algorithm (wrt error) for outputting this value

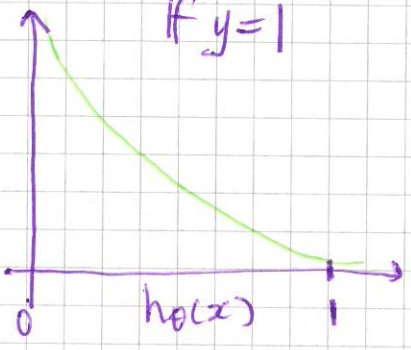
This can't be directly applied to logistic regression because then $J(\theta)$ is not convex and it's more complicated to find a global minimum

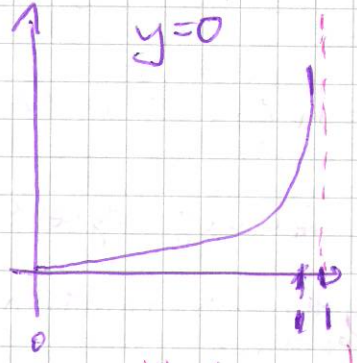




Logistic Regression Cost Function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$





likewise, as $h_\theta(x) \to 1$ cost $\to \infty$

recall log z:

cost = 0 if $y = 1$, $h_\theta(x) = 1$

as $h_\theta(x) \to 0$ cost $\to \infty$

if $h_\theta(x) = 0$ but $y = 1$ we'll penalize learning alg.

Logistic Regression: Simplified Cost Function

Recall: $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$

where $Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}$

$y \in \{0,1\}$

Can rewrite as

$cost(h_\theta(x), y) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$

So, Logistic Regression Cost Function

$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$

$= -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$

Why this cost function

- convex!
- determined through maximum likelihood estimation

To fit parameters θ:

$\min_\theta J(\theta)$

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ $\rightarrow p(y=1|x;\theta)$

So how do we get $\min_\theta J(\theta)$?

Gradient Descent!

We want $\min_\theta J(\theta)$:

Repeat{

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

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

(simultaneously update all θ_j)

}

Algorithm looks identical to linear regression!

The difference is in $h_\theta(x)$.

Logistic Regression: Optimisation

Optimisation Algorithm

We have cost function $J(\theta)$.

Want $\min_\theta J(\theta)$

Given θ, we have code that can compute

$$\begin{bmatrix} -J(\theta) \\ -\frac{\partial}{\partial \theta_j} J(\theta) \leftarrow \text{for } (j = 0, 1, \ldots, n) \end{bmatrix}$$

Which we then use for gradient descent

Repeat {

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

}

But there are algorithms
eg. - Conjugate gradient
- BFGS
- L-BFGS

Advantages
- no need to manually pick α
- often faster than gradient descent

Disadvantages
- more complex

But don't try to implement these algorithms yourself! Beyond the scope of this course. It's worth investigating and finding a good library.

Example

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$

$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

Would implement a cost function:

```
function [jVal, gradient]
        = costFunction (theta)

  jVal = theta(1) -5^2 + ...
         + theta(2) - 5^2

  gradient(1) = 2* theta(1) - 5
  gradient(2) = 2* theta(2) - 5
```

← code to compute $J(\theta)$

← code to compute gradients

Which then leaves you with what you need to use matlab library

```
options = optimset ('gradObj', 'on', 'maxIter', '100')
initialTheta = zeros(2,1)
[optTheta, functionalVal, exitFlag] ...
    = fminunc(@costFunction, initialTheta, options)
```

functionalVal: ideally close to zero.

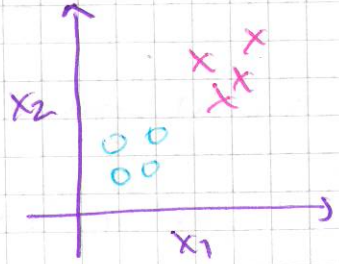
fminunc: function minimisation unconstrained.

@costFunction: pointer to costfunction

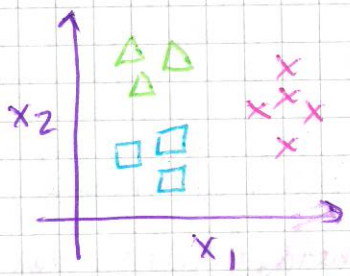
Multiclass Classification

examples:
- autotagging emails
  - work $y=0$
  - friends $y=1$
  - family $y=2$
  - hobby $y=3$
- medical diagnosis
  - not ill
  - cold
  - flu
- weather
  - sunny
  - cloudy
  - rain
  - snow

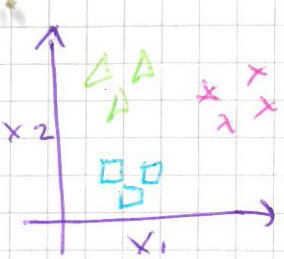
Previously, binary classification



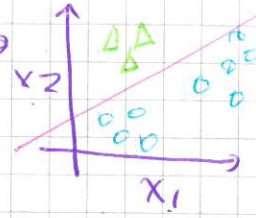
Now, multi-class



One-vs-all (one-vs-rest)

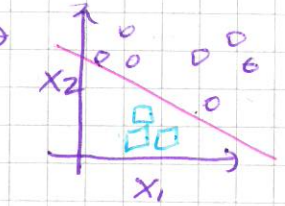


class 1: △
class 2: □
class 3: ×

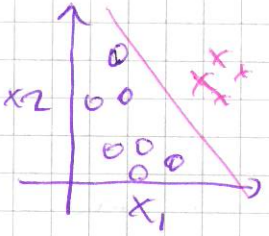


if $h_\theta^{(1)}(x)$

$P(y=1|x;\theta)$



$h_\theta^{(2)}(x)$



$h_\theta^{(3)}(x)$

$h_\theta^{(i)}(x) = P(y=i|x;\theta) \quad i=1,2,3$

Approach:

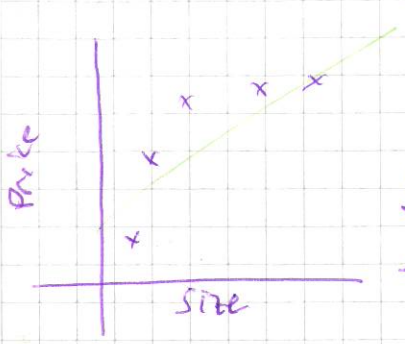
- Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y=i$
- On a new input x, to make a prediction, pick the class i that maximises

$$\max_i h_\theta^{(i)}(x)$$

(check all 3, pick the one that is most confident)

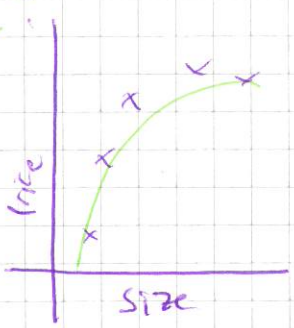
The Problem of Overfitting

What is overfitting?

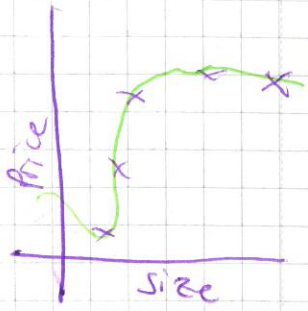


$\theta_0 + \theta_1 x$

"underfit"
"high bias"
(he's already assumed data is linear)



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

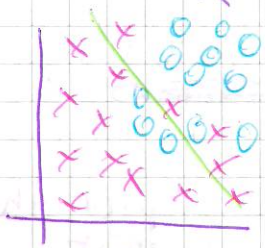


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^2 + \theta_4 x^4$

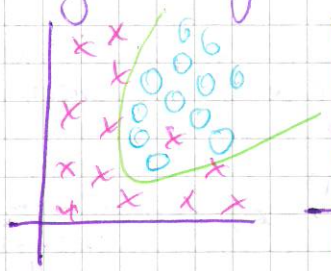
"overfit"
"high variance"

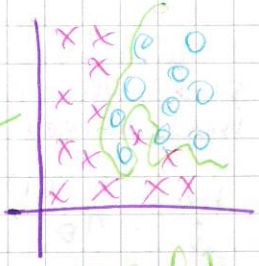
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well, but fail to generalise to new examples

Can overfit for logistic regression as well



underfit





overfit

How to address overfitting?

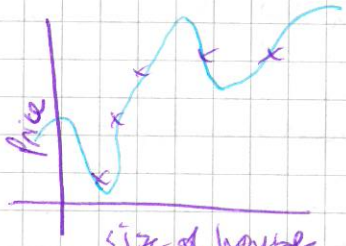
- can only plot hypothesis if you have 1-2 features

Options:

1. Reduce number of features
   - Manually select which features to keep
   - Model selection algorithm (later in course)
2. Regularization
   - 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.

Intuition

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



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

Suppose we penalise and make θ_3, θ_4 really small

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

then $\theta_3 \approx 0$ $\theta_4 \approx 4$

Idea behind Regularisation

- If you have small values for parameters $\theta_0, \theta_1, \dots, \theta_n$
  - "simpler" hypothesis
  - less prone to overfitting

eg. housing

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

harder to pick which parameters are most relevant!

So, we'll modify our cost function

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

But there's a trick to picking λ!

Setting λ to an extremely large value makes it that $\theta_1, \theta_2, \theta_3, \dots \approx 0$

and suggests $h_\theta(x) = \theta_0$, a flat line!

underfit!

Regularised Linear Regression

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

$\min_\theta J(\theta)$

Now: Gradient Descent

Repeat {

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

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

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

}

Can rewrite θ_j update as

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

Normal Equation

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

- Regularisation also takes care of non-~~invertible~~ invertability issue.

Regularised Logistic Regression

Cost function:

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

Gradient descent, same as for linear regression but remember $h_\theta(x)$ has changed!

Regularised Logistic Regression with more Advanced Optimisation

Recall: you need to provide a cost function to fminunc

```
function [jVal, gradient] = costFunction, theta
  jVal = code to compute J(θ)
    J(θ) = new cost function w/ λ term
  gradient(1) = code to compute ∂/∂θ0 J(θ)
      (same as before)
  gradient(2) = code to compute ∂/∂θ1 J(θ)
    1/m Σ_{i=1}^{m} (hθ(x^(i)) - y^(i)) x1^(i)  + λ/m θ1
  :
  etc
```