Subject: Supervised ML Date:

ML: The science of getting computers to learn without being explicitly programmed

Supervised Learning: Algorithm that learn x to Y (input to output mapping)

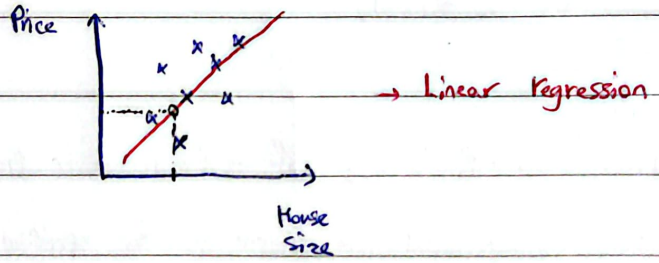
*Learns from being given right answers

e.g:

Input (X)	Output (Y)	Application
email	Spam (0/1)	Spam filtering
audio	text transcript	Speech recognition
ad, user info	click? (0/1)	online advertising
Image, radar info	position of other cars	self-driving car

Regression: a particular type of supervised learning which predicts a number from infinitely many possible outputs

e.g:

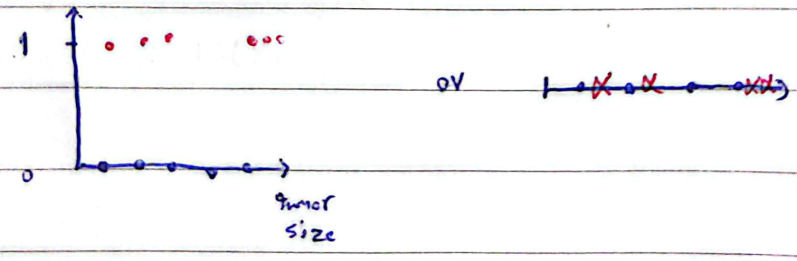


Classification: a particular type of supervised learning which predict categories.

*classification predicts small finite limited set of possible output categories

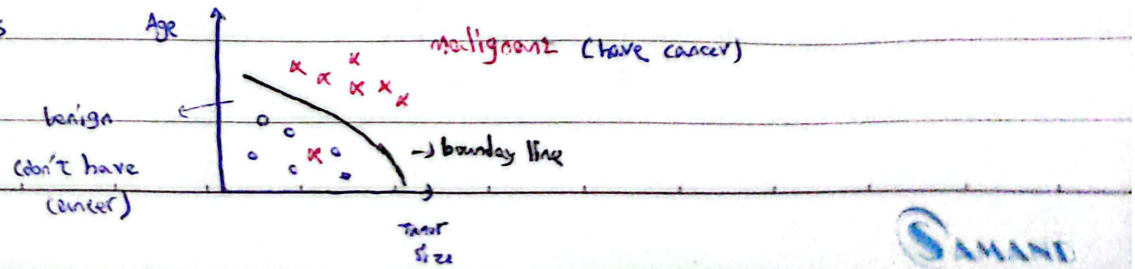
e.g:

classify breast cancer base on tumor size



*Class = Category *Categories can be non-numeric (cat or dog)

e.g: Two inputs

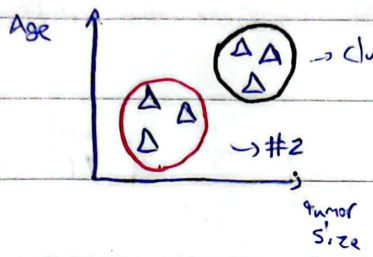


SAMANI

* learning algorithm has to decide how to fit a boundary line through data

Unsupervised Learning: We're given data that isn't associated with any output label y and our job is to find some structure or pattern or just something interesting

e.g1: Clustering: Places the unlabeled data into different clusters



e.g2: Anomaly Detection: Find unusual data points (events) → fraud detection in financial system

e.g3: Dimensionality reduction: Compress data using fewer numbers (while losing as little info as possible)

Terminology and Process of supervised learning

Training set: Data used to train the model

e.g →

	x: size in feet	y: Price in 1000$'s
(1)	2104	400
(2)	1416	232
⋮	⋮	⋮
(47)	3210	870

m=47

Notation

x = 'input' variable or 'feature'

y = 'output' variable or 'target'

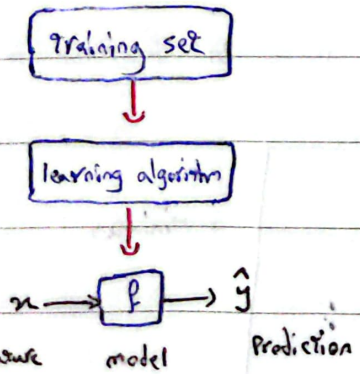
m = number of training examples

$(x^{(i)}, y^{(i)})$ = ith training example

(x, y) = single training example

y-hat: estimate or the prediction for y ($\hat{y}$)

* $\hat{y}$ may or may not be the actual true value



e.g: Size → [f] → Price

Subject: Date:

How to represent f? $f_{w,b}(x) = wx + b$ → Linear regression with one variable

or

$f(x)$

★ this is also called univariate (one variable) linear regression

Cost function: a function that would compute the difference between y and $\hat{y}$

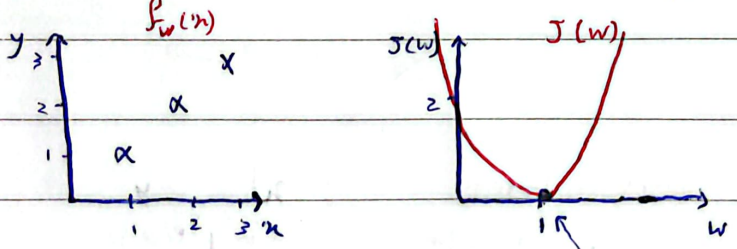
Squared error cost function:

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

$$\hat{y}^{(i)} = f_{w,b}(x^{(i)}) \rightarrow J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

★ Imagine $b = 0$ → $f_w(x) = wx$ and $J(w) = \frac{1}{2m} \sum_{i=1}^{m} (f_w(x^{(i)}) - y^{(i)})^2$

e.g:

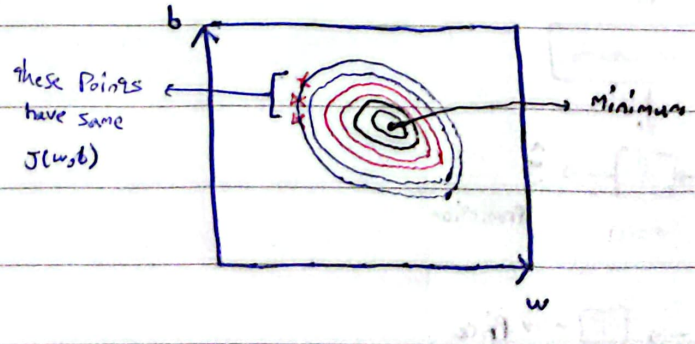


★ Our goal is to minimize cost func or $J(w)$ → so this point is the min point

→ $w = 1$ → $f(x) = x$ is the line that would have least cost and would fit the data pretty well.

In general: if $b \neq 0$ → $f_{w,b}(x) = wx + b$ and $J(w,b)$ is a 3D function with axis w, b and $J(w,b)$.

Contour plot of $J(w,b)$:



Date: Subject:

Gradient descent: an algorithm that we can use to minimize cost func ($J(w_1, \dots, w_n, b)$)

Repeat until convergence (until you reach the a point at a local minimum where w and b no longer change much with each iteration)

{

$tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$

$tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$

$w = tmp\_w$

$b = tmp\_b$

}

α: learning rate

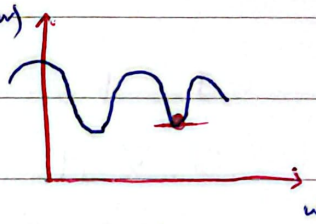
★ simultaneously update w and b

★ if α is too small → Gradient descent may be slow

★ if α is too large → Gradient descent may: a) overshoot, never reach minimum
b) Fail to converge, diverge

★ If we are already at a local minimum, since the tangent line has slope = 0 → then $w = w - \alpha \cdot 0$

→ $w = w$. so gradient descent won't do anything!

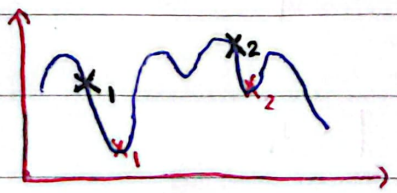


★ as we get near a local minimum, derivative becomes smaller (moving towards 0), and the update steps become smaller → so we can reach min without decreasing α

★ Depending on where you initialize the parameters w and b, you can end up at different local minima

X_i: starting point i

x_i: ending point i after we run algorithm



★ squared error cost func is a convex func. Convex funcs are bowl-shape funcs and have single a local min (which is also global min as well.

e.g: Linear regression model: $f_{w,b}(x) = wx + b$ Cost func: $J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$

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

Subject: Date:

★ This version of gradient descent is called "Batch gradient descent"

★ Batch: Each step of gradient descent uses all of the training examples (there are versions that with only a subset of them)

Multiple features (variables)

x_j = j^{th} feature

n = number of features

$\vec{x}^{(i)}$ → feature of i^{th} training example

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

x_1	x_2	x_3	...	x_n	y
1	2	3		5	20
6	7	8		1	30

$X^{(1)} = [1, 2, 3, \dots, 5]$

Model:

$$f_{w,b}(x) = w_1 x_1 + \dots + w_n x_n + b$$

or

$\vec{w} = [w_1\ w_2\ \dots\ w_n]$ b is a number $\vec{x} = [x_1\ x_2\ \dots\ x_n]$

→ $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$ → multiple linear regression

↳ dot Product

Vectorization:

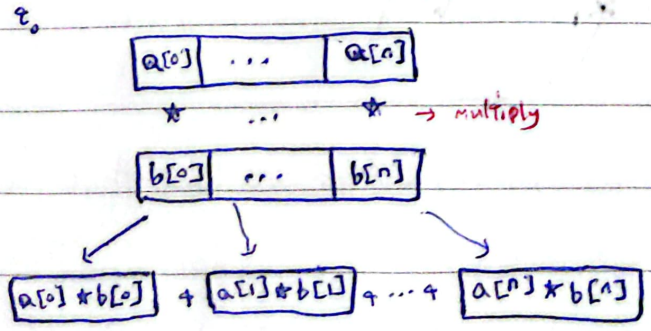
★ Use np.dot(a,b) over for-loop.

benefits

1- Shorter code

2- faster (cause it uses Parallel hardware) (!)

(!): e.g.



★ efficient → scale to large datasets

Gradient descent with Vector notation (multiple features)

repeat {

$j=1 \quad w_1 = w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_1^{(i)}$

$\vdots$

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

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

}

An alternative to gradient descent: Normal equation

Normal equation:

a) Only for linear regression

b) Solve for w, b without iterations

★ Gradient descent is still the recommended method

Disadvantages:

1- Doesn't generalize to other learning algos

2- Slow when number of features is large ($>10^4$)

Feature and Parameter Values

Imagine: $y = w_1 x_1 + w_2 x_2 + b$ (Price)

x_1 range: 300 - 2000 (large)

x_2 range: 0 - 5 (small)

Training example: $x_1 = 2000 \quad x_2 = 5 \quad$ Price (y) = 500

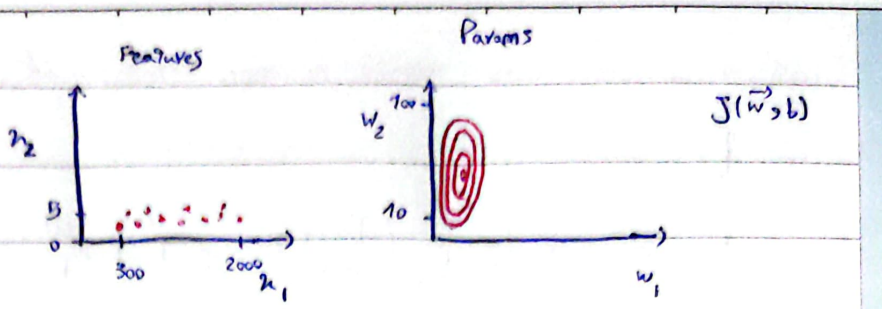
Possible Params → $w_1 = 0.1$ (small) $\quad w_2 = 50$ (large) $\quad b = 50$

★ when a possible range of values of a feature is large, a good model will learn to choose small parameter value

★ when a possible range of values of a feature is small, a good model will learn to choose large parameter value

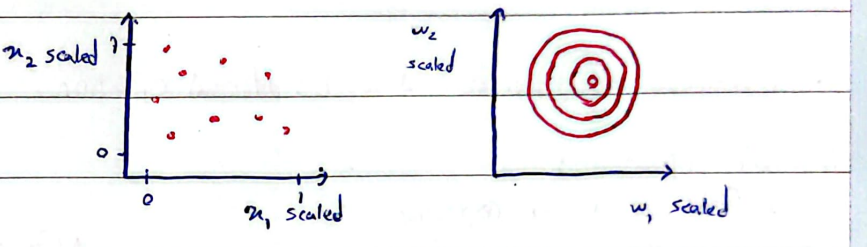
Subject: Date:

effect of this on gradient descent:



★ If we run gradient descent with this, gd will end up bouncing back and forth for a long time before it can finally find its way to the min

→ What to do? Scale features!



Feature scaling:

1- Divide by max: $300 \le x_1 \le 2000 \rightarrow x_{1,scaled} = \frac{x_1}{2000} \rightarrow 0.15 \le x_{1,scaled} \le 1$

2- Mean normalization: $x_{1,s} = \frac{x_1 - \mu_1}{max - min} \rightarrow -0.18 \le x_{1,s} \le 0.82$

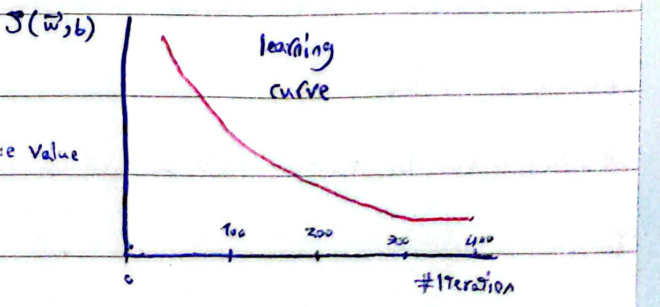
3- Z-score normalization: $x_{1,s} = \frac{x_1 - \mu_1}{\sigma_1} \rightarrow -0.67 \le x_1 \le 3.1$

★ aim for a okay range → $-1 \le x \le 1$ ✓ $-3 \le x \le 3$ ✓ $-100 \le x \le 100$ → too large

$-0.001 \le x \le 0.001$ → Too small $98.6 \le x \le 104$ → too large

Make sure gd is working correctly

- $J(\vec{w},b)$ should decrease after every iteration
- After x iterations, learning curve should converge toward some value
- # iterations needed varies



Another way: Automatic convergence test:

Let ε be a small number. If $J(\vec{w},b)$ decreases by $\leq \varepsilon$ in one iteration → declare convergence
(like 10^{-3})

★learning curve method is better due to difficulty of finding a good ε

Chosing the learning rate

We can make decisions about our learning rate (whether it's too large or too small) by looking at the learning curve or # iterations.

★ a mistake in the learning curve might also be caused by a bug in our program. We can identify that by using a small enough α and if $J(\vec{w},b)$ increases in one iteration → there is a bug!

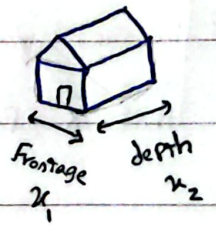
- Values of α to try: 0.001 0.003 0.01 0.03 0.1 0.3 1
($\times 3$ between 0.001 and 0.003, $\times 3$ between 0.003 and 0.01)

★ Andrew tend to use the largest possible α that is reasonable and get runs well for it

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

e.g: $f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + b$

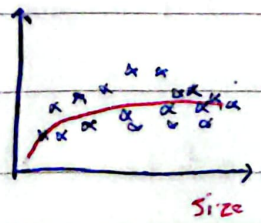
$\text{area} = x_1 \times x_2 \rightarrow x_3 = x_1 x_2$



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

Polynomial regression: Sometimes a straight line doesn't fit data-set very well so we need to use a non-linear one.

★ feature scaling is important here



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

Subject: Date:

Classification:

- Binary classification: There are only two possible outputs (two possible classes) or categories

e.g:

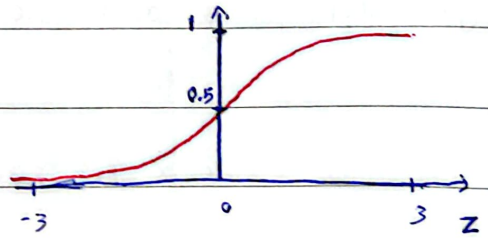
No	Yes
False	True
0	1
Negative class	Positive class

★ What happens if we use linear regression for classification? Sometimes you might get lucky and it may work but often it will not work well

Logistic regression:

- Sigmoid function (logistic function):
  - outputs between 0 and 1

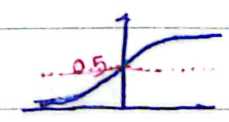
$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$



$$z = \vec{w}\cdot\vec{x} + b \rightarrow f_{\vec{w},b}(\vec{x}) = g(\vec{w}\cdot\vec{x}+b) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}} \rightarrow \text{logistic regression} \rightarrow \text{used for classification}$$

Interpretation of logistic regression output: $f_{\vec{w},b}(\vec{x}) = P(y=1 \mid \vec{x}; \vec{w}, b)$ → Probability that y is 1, given input $\vec{x}$, parameters $\vec{w}, b$

Decision boundary: We can set a threshold above which you predict $\hat{y}=1$, and below which you might say $\hat{y}=0$. a common choice → 0.5

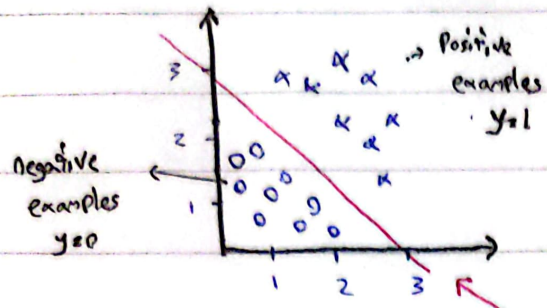


$f_{\vec{w},b}(\vec{x}) \geq 0.5$ or $g(z) \geq 0.5$ or

$z \geq 0$ or $\vec{w}\cdot\vec{x}+b \geq 0 \rightarrow \hat{Y}=1$ ow $\rightarrow \hat{Y}=0$

e.g: a classification problem with 2 features

Date: Subject:



$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1 x_1 + w_2 x_2 + b)$

(Imagine $w_1 = 1$ $w_2 = 1$ $b = -3$)

Decision boundary : $z = \vec{w} \cdot \vec{x} + b = 0$

$\Rightarrow x_1 + x_2 - 3 = 0 \rightarrow x_1 + x_2 = 3$

this line!

Non-linear decision boundaries :

use Polynomials

$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1 x_1^2 + w_2 x_2^2 + b)$

(Imagine $w_1 = 1$ $w_2 = 1$ $b = -1$)

$z = x_1^2 + x_2^2 - 1 = 0 \rightarrow x_1^2 + x_2^2 = 1$

this circle

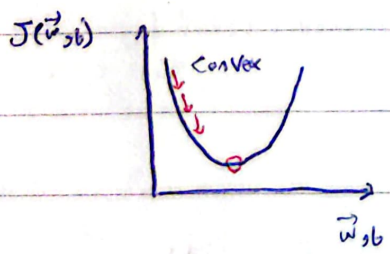
Cost function for logistic regression :

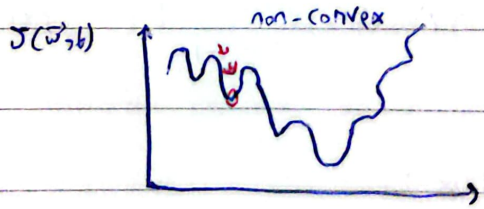
Squared error cost function is not good for logistic regression cause there are alot of local minimas in the plot → gd will do poorly . We need a func that would make the plot Convex again.

$$J(\vec{w},b) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2$$ → squared error

↳ linear regression

logistic regression





★ Loss func for squared error : $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \frac{1}{2} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2$

Subject: Date:

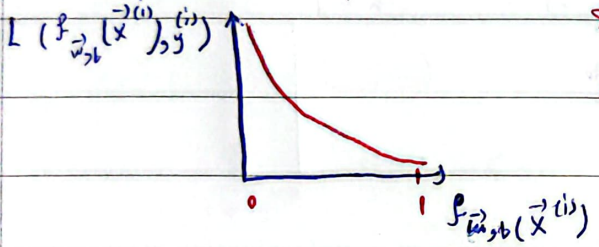
★ loss func → measure how well you're doing on 1 training example

★ cost func = $\sum$ loss func → for all training examples

Logistic loss func:

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=1 \\ -\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=0 \end{cases}$$

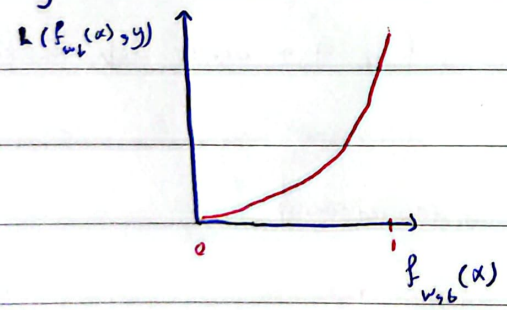
if $y^{(i)}=1$



★ As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to 0$

★ As $z \to 0$ then loss $\to \infty$

if $y^{(i)}=0$



★ As $f_{w,b}(x) \to 0$ then loss $\to 0$

★ As $f_{w,b}(x) \to 1$ then loss $\to \infty$

cost func: $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} L\left(f_{w,b}(x^{(i)}), y^{(i)}\right)$ ★ is convex: Proof beyond scope of this course

Simplified loss func:

$$L\left(f_{w,b}(x^{(i)}), y^{(i)}\right) = -y^{(i)}\log\left(f_{w,b}(x^{(i)})\right) - (1-y^{(i)})\log\left(1-f_{w,b}(x^{(i)})\right)$$

cost func $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left[L\left(f_{w,b}(x^{(i)}), y^{(i)}\right)\right]$

★ we use this loss func cause it's related to Maximum likelihood

Gradient descent for logistic regression

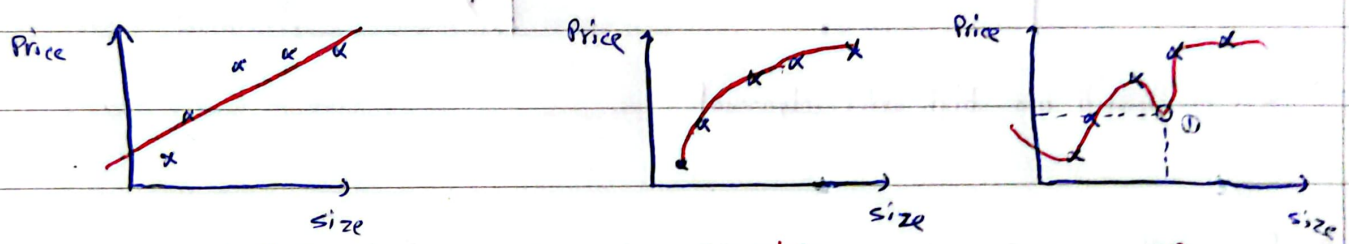
Cost: $J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right]$

★ $\frac{\partial}{\partial w_j} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}$ ★ $f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{(-\vec{w}\cdot\vec{x}+b)}}$

★ $\frac{\partial}{\partial b} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$ ★ $g'(z) = g(z)(1-g(z))$

★ Same concepts as for linear regression → learning curve - Vectorize - Feature scaling etc

Underfitting & Overfitting - for linear regression:



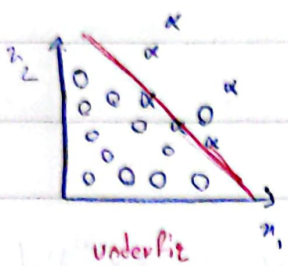
Event	Underfit	Just right	Overfit
e.g model	$w_1 x + b$	$w_1 x + w_2 x^2 + b$	$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$
another name (Technical term)	(high bias)	(generalization)	(high variance)
explanation	Does not fit the training set well	Fits training set pretty well	Fits the training set extremely well

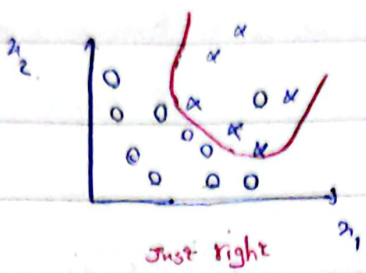
★ The middle model will do well on training examples that are not in the training set

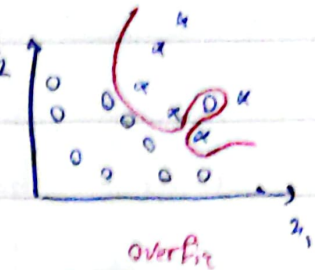
★ The overfit model will have cost func = 0 but it's not good for prediction → e.g: ①. So we can't expect this model will do well on examples that are not in the training set

for Classification:

Subject: Date:



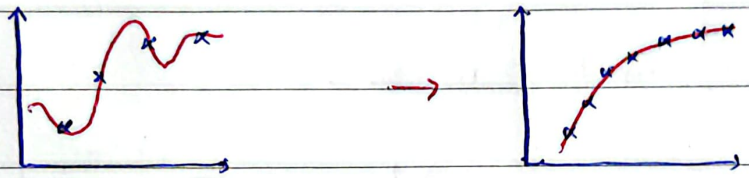




	underfit	just right	overfit
event			
Z	$z = w_1 x_1 + w_2 x_2 + b$	$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 + b$	$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 + \dots + b$

Addressing overfitting: both for regression & classification

1- Collect more training examples



*It would still has high order polynomial

2- Feature selection: try selecting and using only a subset of the features

*disadvantage: useful features could be lost

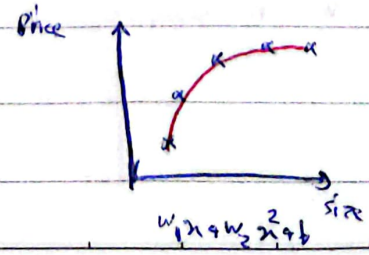
3- Regularization: reduce the impact of some of the features without eliminating them (like in feature selection) & → reduce the associated w

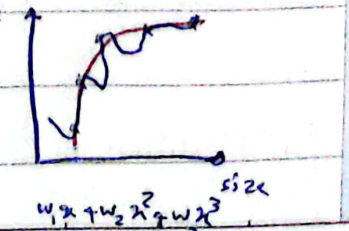
e.g.: $f(x) = 28x - 385x^2 + 39x^3 - 179x^4 + 100 \rightarrow 13x - 0.23x^2 + 0.00004x^3 - 0.001x^4 + 10$

*reduce b doesn't make that much difference → Andrew usually don't reduce it

Cost func with regularization

→ we need to make w_3 and w_4 really small





Date: Subject:

★ We can do this with a modified cost func: $\frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)})-y^{(i)})^2 + 1000\,w_3 + 1000\,w_4$

$\min J \equiv \min(1000\,w_3 + 1000\,w_4) \equiv \min(w_3, w_4)$

★ we can use any big number instead of 1000

In general: we don't know which w to penalize so we penalize all of them:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)})-y^{(i)})^2 + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}_{\text{regularization term}} + \underbrace{\frac{\lambda}{2m} b^2}_{①} \qquad \lambda > 0$$

★ λ = regularization Parameter → we need to choose it like we choose learning rate

★ n = number of features ★ Penalize b ① can be excluded → cause it has little impact

★ minimize mean squared error term → fit data well ; minimize regularization term → keep w_j small so reduce overfitting

→ λ balances both if we choose it wisely

★ if $\lambda \to 0$ then no regularization → overfit

if $\lambda \to \infty$ then $w_j \to 0$ so $f_{w,b}(x) = b$ → Underfit

Gradient descent for regularized linear regression

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b) \longrightarrow \frac{\partial}{\partial w_j} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)})-y^{(i)})x_j^{(i)} + \frac{\lambda}{m} w_j$$

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

we didn't regularize b

} simultaneous update

★ Some Math intuition on how this regularized version shrinks w_j

$$w_j = w_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left[f_{w,b}(x^{(i)})-y^{(i)}\right)x_j^{(i)}\right] + \frac{\lambda}{m} w_j\Big] \longrightarrow$$

Subject: Date:

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

$w_j - \alpha \frac{\lambda}{m} w_j = w_j \left(1 - \alpha \frac{\lambda}{m}\right)$

usual update (like before regularization)

→ $1 - \alpha \frac{\lambda}{m}$. Imagine $\alpha = 0.01$, $\lambda = 1$, $m = 50$ ⟶ $1 - \alpha \frac{\lambda}{m} = 0.9998$

→ $0.9998\, w_j$ → in every iteration we are shrinking w_j just a little bit

For logistic regression:

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(f_{w,b}(x^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - f_{w,b}(x^{(i)})\right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

☆ gradient descent for this regularized version is the same for linear regression (regularized) and the only difference is that $f_{w,b}(x)$ is the sigmoid func