SUPERVISED MACHINE LEARNING: REGRESSION & CLASSIFICATION

~~UDACITY : INTRO TO MACHINE LEARNING~~

~~KK1~~ ~~NAIVE BAYES~~ ~~WHAT IS MACHINE LEARNING~~

KK1 SUPERVISED vs. UNSUPERVISED MACHINE LEARNING

- Machine learning is a field of study that gives computers the ability to learn w/out being explicitly programmed.
- Common ML algorithms — (i) Supervised learning
  (ii) Unsupervised learning (iii) Recommender systems (iv) Reinforcement learning

(A) SUPERVISED LEARNING:

- Refers to algorithm that learn $x \rightarrow y$ or input to output mappings. Key characteristic = in Supervised learning, we give the learning algorithm examples to learn from that include the right answers i.e. correct label, y for a given input x.
- It's by using those examples and learning to relationships, that the algorithm is able to give a reasonably accurate prediction of the output given only the input
- Regression, a type of supervised learning, refers to predicting a continuous number from infinitely many possible outputs
- Classification algorithm, is another type of supervised learning. This is different from regression because we're trying to predict only a small number of outputs while regression tries to predict a continuous number.
  - → output class or output category ≡ output.
  - Classification algorithms predict categories and they don't have to be numbers
  - → Needs how to fit a boundary line to the data in cases of 2 or more inputs.

(B) UNSUPERVISED LEARNING

~~You're given~~ In the supervised learning, here we're given data that isn't associated with any output labels y. It mostly finding something interesting in unlabeled data

Algorithm types: (i) Clustering: used in google news [uses words in heading to note top news]

"Data only comes w/inputs x not output labels y. The algorithm has to find structure in the data.

(i) Clustering algorithms grouping similar data points together

(ii) Anomaly detection = algorithms finds unusual data points useful for fraud detection

(iii) Dimensionality reduction = algorithms compressing data from big to small w/out losing too much information

WK 1 REGRESSION MODEL

Cost Function: Take the prediction $\hat{y}$ & subtracts the target y from it the squares the difference. $\frac{1}{2m}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 = J_{(w,b)}$

- n = number of training examples

$y^{(i)}$ — error

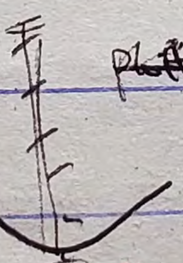
Squared error cost function.

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

$$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$$

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

- In linear regression, we use the cost function to find the value of w & b that minimize $J_{(w,b)}$



plotting all possible value of

Goal of Linear Regression.

WK 1 TRAIN THE MODEL W/ GRADIENT DESCENT

- An algorithmic way of finding the exact w & b that result in the min. $J_{(w,b)}$ refers to
- Gradient descent can work w/several parameter to minimize the cost function represents

↳ Interesting property of Gradient Descent: It contains LOCAL MINIMA i.e. different local points w/only one being that can be reached based on your starting point.

Gradient descent is done in such a way that wherever you are, you take the steepest baby step and keep on taking the steepest baby steps around you until you reach a local minimum.

Gradient Descent Algorithm

$$w = w - \alpha \left[\frac{\partial}{\partial w} J(w,b) \right] \quad (1)$$

→ derivative of $J(w,b)$ | → tells us the direction of the step

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \quad (2)$$

learning rate → α

- Learning rate, α, controls how big of a step you take downhill.

large α = large steep step, small α = baby steps.

- In the gradient descent algorithm, you repeat the update steps (1) & (2) until the algorithm converges. Convergence = the point @ a local minimum where the parameters w & b no longer change much with each step taken.

- Note: Both w & b in (1) & (2) are updated simultaneously

Example: 1. $tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$

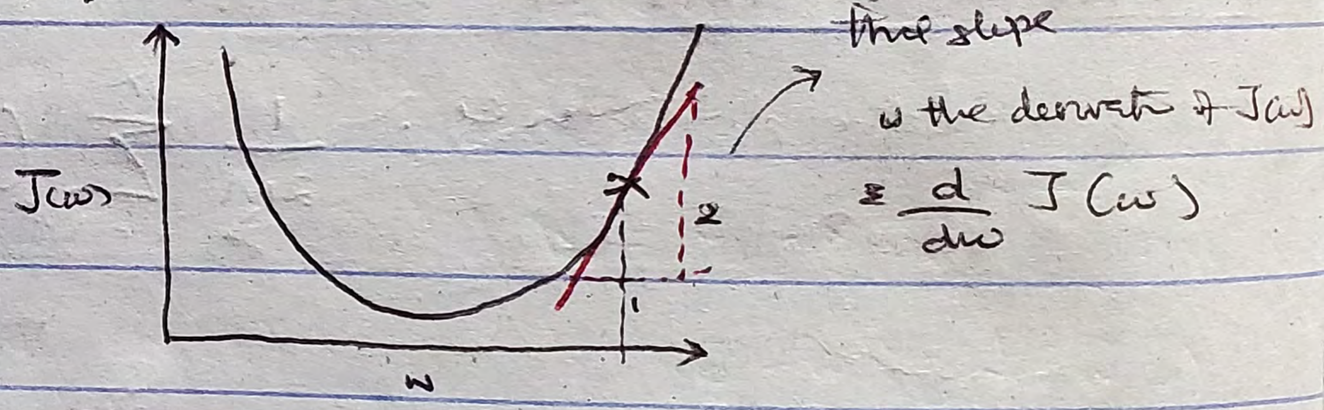
2. $tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$

3. $w = tmp\_w$, $b = tmp\_b$

w = weight
b = bias

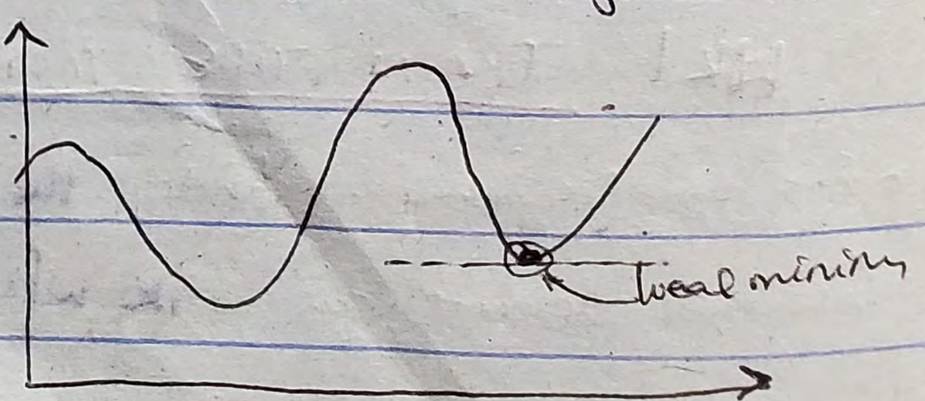
Gradient Descent Intuition → Derivative

- Looking @ $J(w)$; → setting intercept $b = 0$



LEARNING RATE: → The has a huge impact on the efficiency of gradient descent

- If α is too small; gradient descent may be slow. If α is too large; gradient descent may overshoot & never reach the minimum i.e. it may fail to converge instead it'll diverge.



What if you've already reached a local minimum?

Note: This is not a squared error cost function and it has 2 local minima.

$$w = w - \alpha \left[\frac{d}{dw} J(w) \right]$$

→ The slope @ the local minimum is equal to 0

$\therefore w = w - \alpha(0)$

$w = w$ ... no update

- As we approach a local minimum;
  - The derivative becomes smaller
  ∴ - The update steps become smaller
- This means that we can reach minimum w/out decreasing learning rate α

GRADIENT DESCENT for LINEAR REGRESSION

- for our linear regression model we have;

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

- Our cost function, $J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left[f_{w,b}(x^{(i)}) - y^{(i)} \right]^2$

∴ the gradient descent algorithm

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \quad ; \text{ where } \frac{\partial}{\partial w} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left[f_{w,b}(x^{(i)}) - y^{(i)} \right] x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \quad ; \text{ where } \frac{\partial}{\partial b} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left[f_{w,b}(x^{(i)}) - y^{(i)} \right]$$

- The squared error cost function always has a global minimum & will never have multiple local minima. In order words, it is a convex function.

CONVEX FUNCTION: A bowl shaped function that can't have any local minima other than the single global minimum.

- The gradient descent used = batch gradient descent. "Batch" because each step of the gradient descent uses all the training examples (→ m = 1000s)

WK2 MULTIPLE LINEAR REGRESSION

A: MULTIPLE FEATURES (variables)

	Size in feet^2 (x_1)	No. of Bedrooms (x_2)	No. of floors (x_3)	Age of home in yrs (x_4)	Price($) in $1000s
	2104	5	1	45	460
$\vec{x}_2$	1416	3	2	40	232
	1534	3	2	30	315
	852	2	1	36	178
	...	...	...	...	...

x_j = j^{th} feature ; n = no. of features ; $\vec{x}^{(i)}$ = features of i^{th} training example

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

e.g. $\vec{x}^{(2)} = [1416, 3, 2, 40]$

$x_3^{(2)} = 2$

$j = 1 \cdots 4$

$n = 4$

New model; $f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$

↳ $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$; $\vec{w} = [w_1, w_2, \cdots, w_n]$

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

- Note this is not multivariate regression

B: VECTORIZATION → Helps makes code shorter & more efficient when implementing a learning algorithm

↳ Make use of Numpy functions.

C: GRADIENT DESCENT for MULTIPLE LINEAR REGRESSION

	Previous notation	Vector notation (vector of length n)
Parameters	$w_1, \cdots, w_n$ b	$\vec{w} = [w_1 \cdots w_n]$ b
Model	$f_{\vec{w},b}(\vec{x}) = w_1x_1 + \cdots + w_nx_n + b$	$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$
Cost function	$J(w_1, \cdots, w_n, b)$	$J(\vec{w}, b)$

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

Normal equation → An alternative to gradient descent.

↳ . Only for linear regression & solves for w, b w/out iterations.

Disadvantages:

(i) Doesn't generalize to other learning algorithms

(ii) Slow when no. of features is large (> 10000)

What you need to know:

- Normal eqn method may be used in ML libraries to implement linear regression & not by yourself.
- Gradient descent is a better way

Wk2 GRADIENT DESCENT IN PRACTICE

A. FEATURE SCALING : When the possible range of values of a feature is large, it's more likely that a good model would learn to choose a relatively small parameter value. Likewise when the possible values of a feature are small, the reasonable value for the parameters will be large

How they does this relate to Gradient Descent?

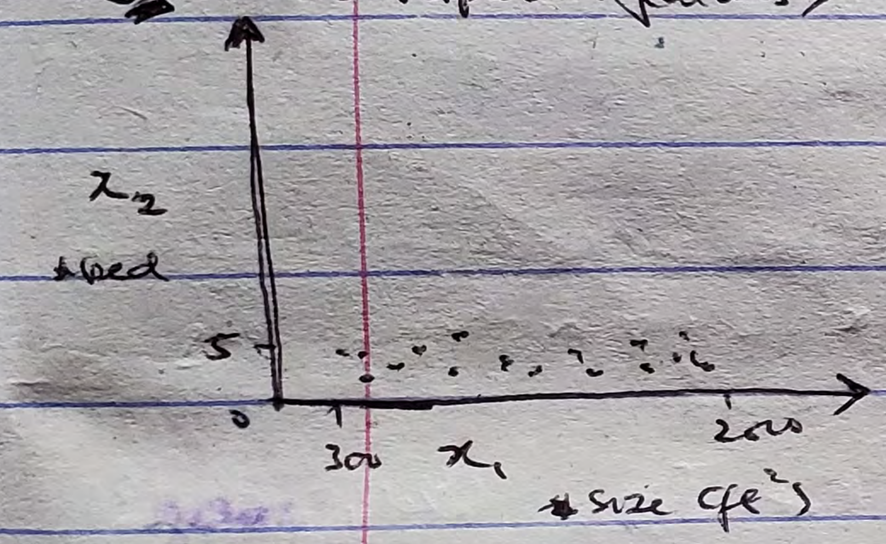
E.g: price $= w_1 x_1 + w_2 x_2 + b$; x_1: size (feet^2) ; x_2 = No. of bedrooms

Range = 300 - 2000 ; Range = 0 - 5

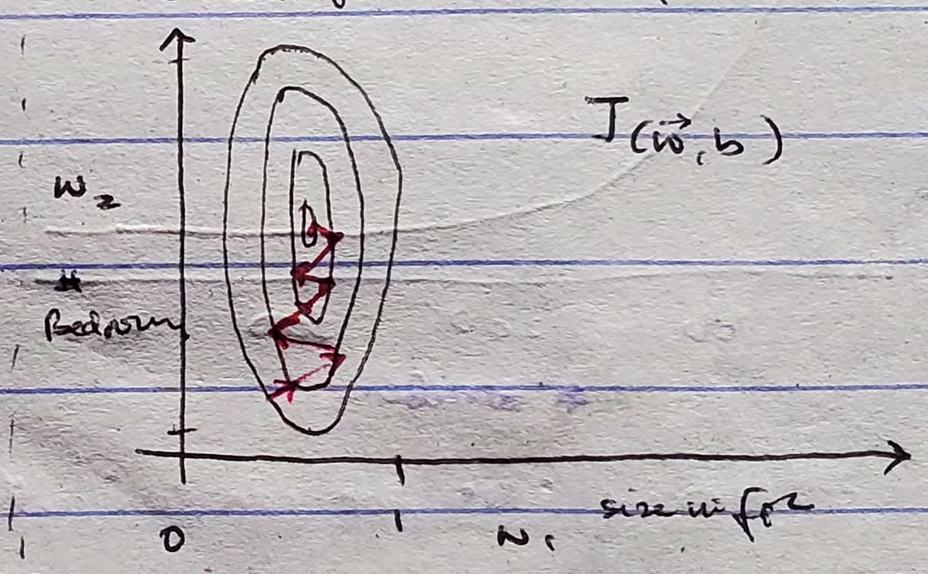
Possible
- Size of parameters for a house w/ $x_1 = 2000$, $x_2 = 5$ & price = \$500k

$w_1 = 0.1$; $w_2 = 50$, $b = 50$ => price $= 0.1(2000K) + 50(5) + 50 = \$500k$

<scatterplot of features>



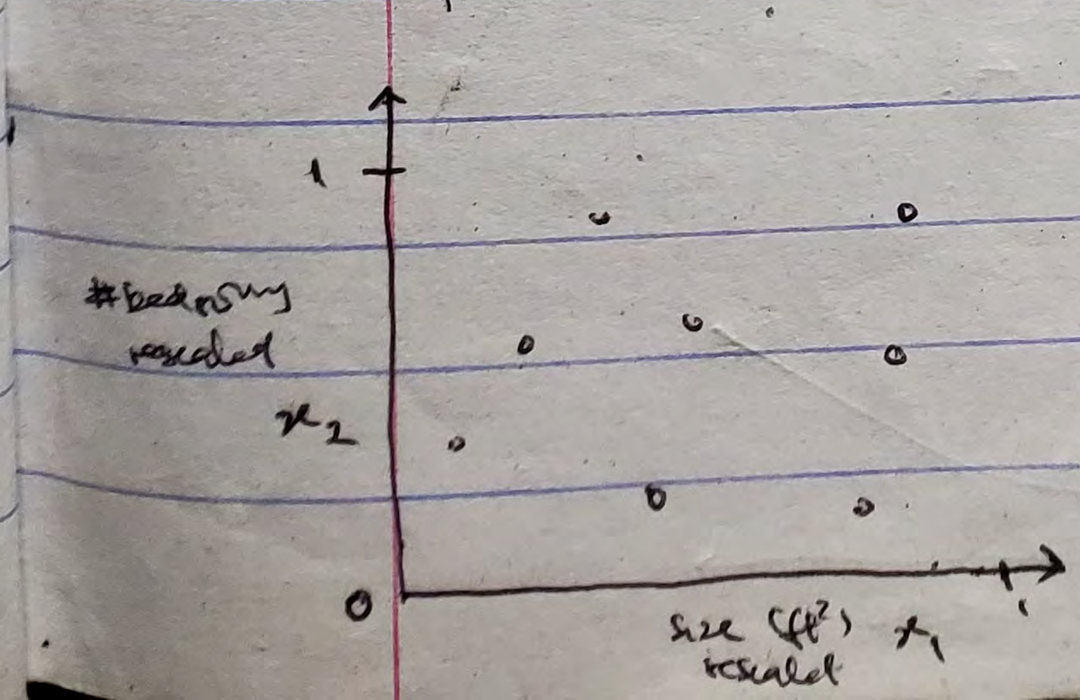
Contour plot
<contour plot of cost function>

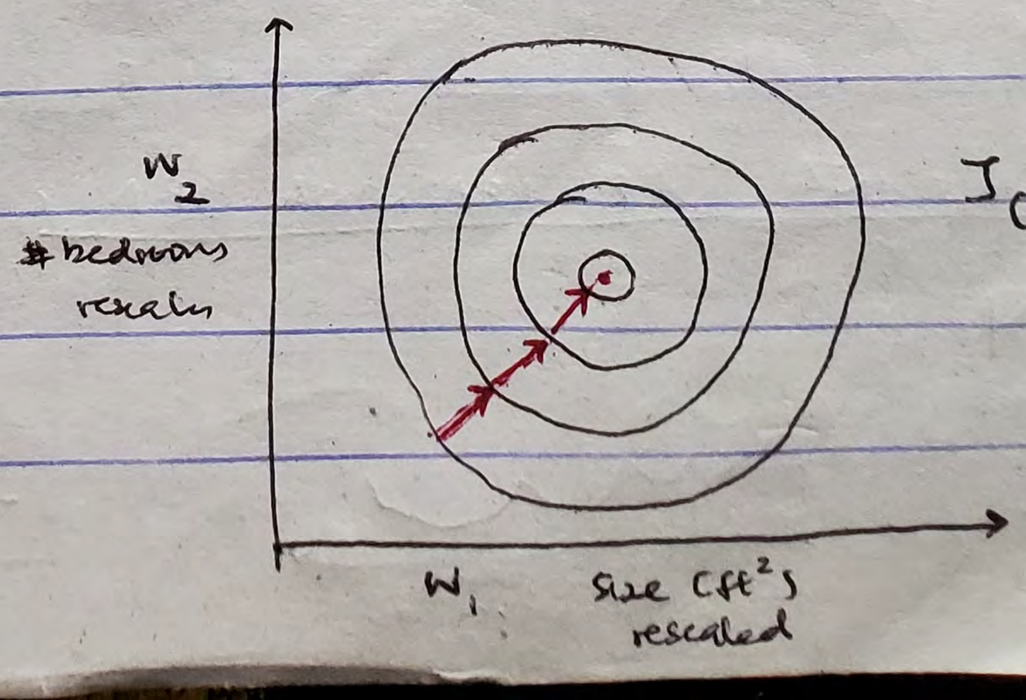


FEATURES

Horizontal axis has a much narrower range, w/ the vertical having a larger range. This is a very small change to w_1 can have a very large impact on the estimated price & thus a large impact on the cost J. The opp. is for w_2

=> Because the contours are so tall & skinny, gradient descent might end up bouncing back & forth for a long time before settling on the global minimum. In order to avoid this, the features need to be scaled i.e. transform the data so the x_1, x_2 range from around from ~0 to ~1





summary: Rescaling of features speed up gradient descent significantly

Scaling methods

- Mean normalization → $300 \le x_1 \le 2000$

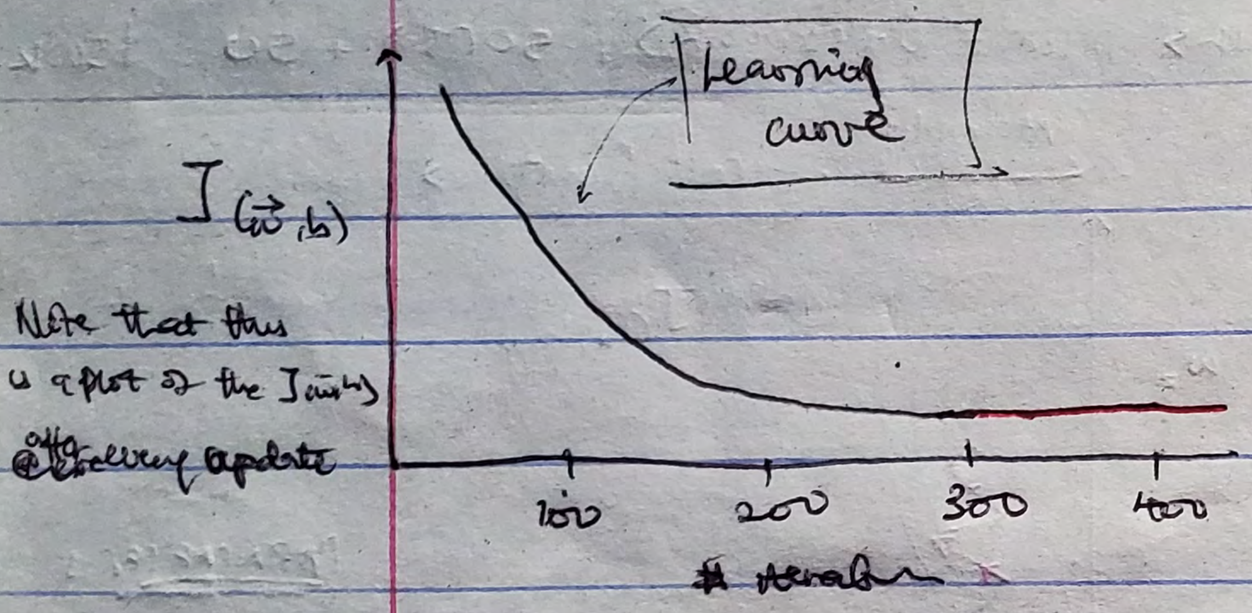
$$\hookrightarrow x_i = \frac{x_i - \mu_i}{2000 - 300}$$ (μ_i: mean of x_i; $2000-300$: $x_{max} - x_{min}$)

More common: Z-score normalization $= \frac{x_i - \mu}{\sigma_i}$

Aim for Feature scaling : $-1 \le x_j \le 1$, $-3 \le x_j \le 3$, $-0.3 \le x_j \le 0.3$ } acceptable range for each feature

B: CHECKING GRADIENT DESCENT for CONVERGENCE

The objective of gradient descent is to minimize the cost: $\min_{\vec{w},b} J(\vec{w},b)$



Note that this is a plot of the $J_{(\vec{w},b)}$ after every update

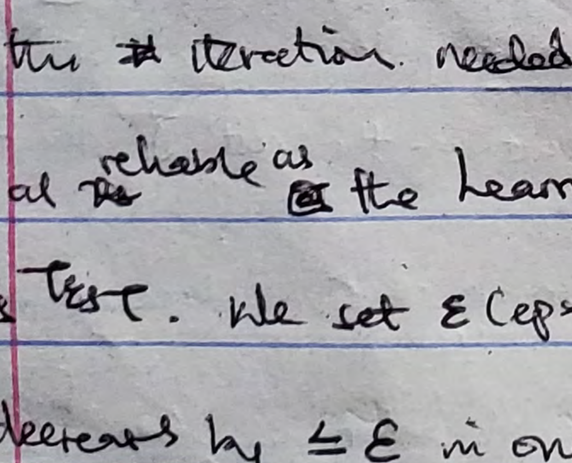
- The graph helps us visualize how $J_{(\vec{w},b)}$ changes after every single iteration
- If it ever increases, it either means that α is too large or there's a bug in the code
- Looking @ the learning curve, we can confidently say that $J_{(\vec{w},b)}$ likely converged by 400 iterations because the curve is no longer decreasing.
- Note that the # iteration needed to converge varies

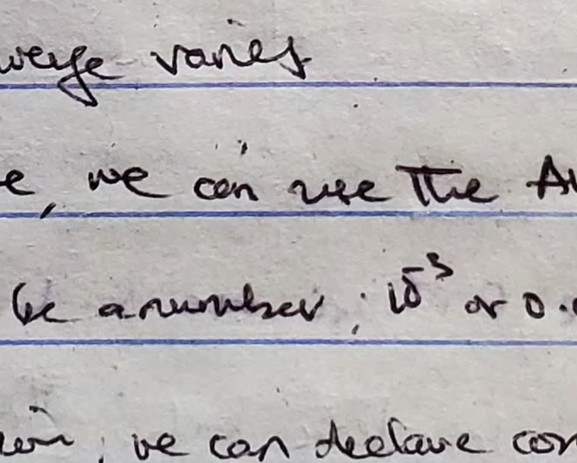
Although not as reliable as looking @ the learning curve, we can use the Automatic convergence Test. We set ε (epsilon) to be a number: 10^{-3} or 0.001. We say that if $J_{(\vec{w},b)}$ decreases by $\le \varepsilon$ in one iteration, we can declare convergence

CONVERGENCE ≡ We've found parameters $\vec{w}, b$ that give us the global minimum where our cost, $J_{\vec{w},b}$ is least.

C: CHOOSING THE LEARNING RATE

- Faulty gradient learning curves





→ most likely a bug

- Both can be solved by choosing a smaller learning rate.

Ro Debugging Tip |→ When in error, set α to a very low value & see if the learning curve decreases. If it doesn't, then it's a bug.

└ use values for α = [0.001, 0.01, 0.1, 1] ; [0.003, 0.03, 0.3, 3] (×3)

D: FEATURE ENGINEERING

↳ This involves using your knowledge or intuition about the problem to design new features. Usually by transforming or combining original features.

Example: Suppose you have a target eto: Y of price of a house with features, x_1 Frontage of the house (ft) & x_2: depth of the house i.e. basically length & width. You might decide that area would be a useful feature and have $x_3 = x_1 x_2$ (ft²) ∴ your function would be; $y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$

E: POLYNOMIAL REGRESSION

WK3 CLASSIFICATION WITH LOGISTIC REGRESSION

• Binary classification = 2 classes or categories.

└ Example:

Question	Answer "y"	
	No	Yes
Is this email spam?	No	Yes
Is the transaction fraudulent?	No	Yes
Is the tumor malignant?	No	Yes

False → 0 or 1 ← True (for model)

-ve class ↑ +ve class

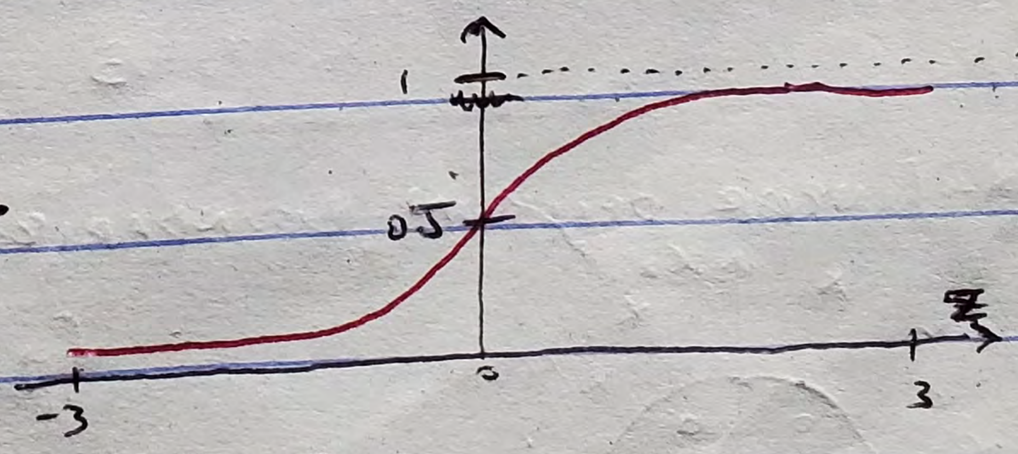
How do you build a classification model?

A: LOGISTIC REGRESSION

↳ sigmoid or logistic function →

outputs values between 0 & 1

$g(z) = \frac{1}{1+e^{-z}}$; $0 < g(z) < 1$ (e ≈ 2.7)



ⓐ $f_{\vec{w},b}(\vec{x})$ → $z = \vec{w} \cdot \vec{x} + b$ → ⓑ $g(z) = \frac{1}{1+e^{-z}}$ ← ⓒ $f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$

INTERPRETATION of LOGISTIC REGRESSION OUTPUT

$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$ } think of the output as the probability that the class or "y" is 1 given a certain input, x

Examp: x = "tumor size" ; y = "0 (not malignant) / 1 (malignant)"

↳ if $f_{\vec{w},b}(\vec{x}) = 0.7$, then the model thinks that the patient has a 70% chance of their tumor turning out to be malignant.

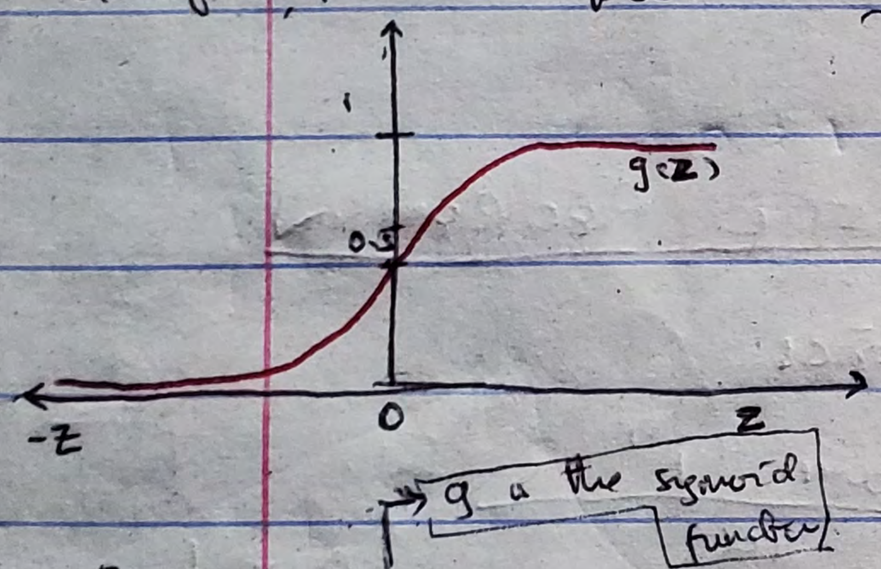
Note when ready research papers you might see → $f_{\vec{w},b}(\vec{x}) = P(y=1|\vec{x};\vec{w},b)$

this means: the probability that $y=1$, given the input $\vec{x}$ with parameters $\vec{w},b$

semicolon denotes what the regression params are.

B: DECISION BOUNDARY : Choosing what threshold determines whether $\hat{y}=1$ or 0 based on $f_{\vec{w},b}(\vec{x})$

For example, take the sigmoid plot



↳ g a the sigmoid function

↳ Suppose @ $f_{\vec{w},b}(\vec{x}) \geq 0.5$, let $\hat{y}=1$

then when $\hat{y}=1$, ↳ $g(z) \geq 0.5$

$z \geq 0$

$\vec{w}\cdot\vec{x}+b \geq 0$ ≃ $\vec{w}\cdot\vec{x}+b < 0$

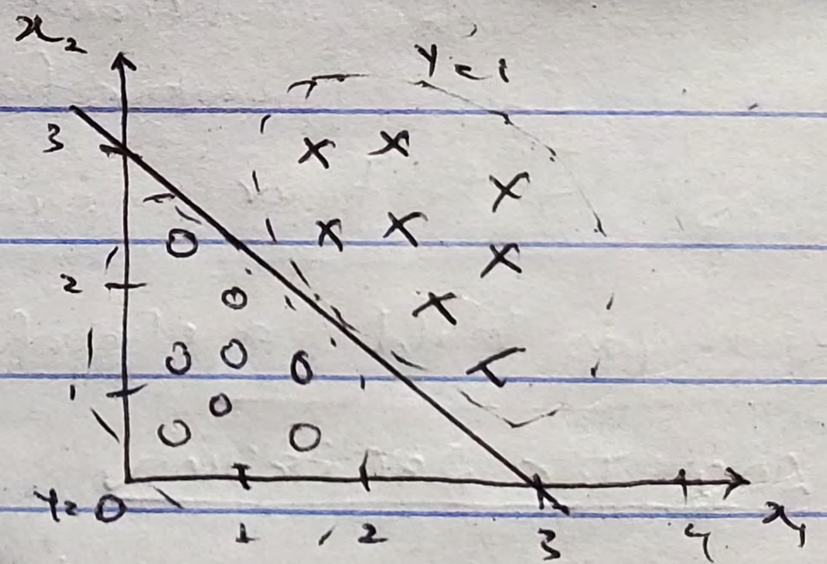
$\hat{y}=1$ $\hat{y}=0$

$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underset{1}{w_1}x_1 + \underset{1}{w_2}x_2 + \underset{-3}{b})$

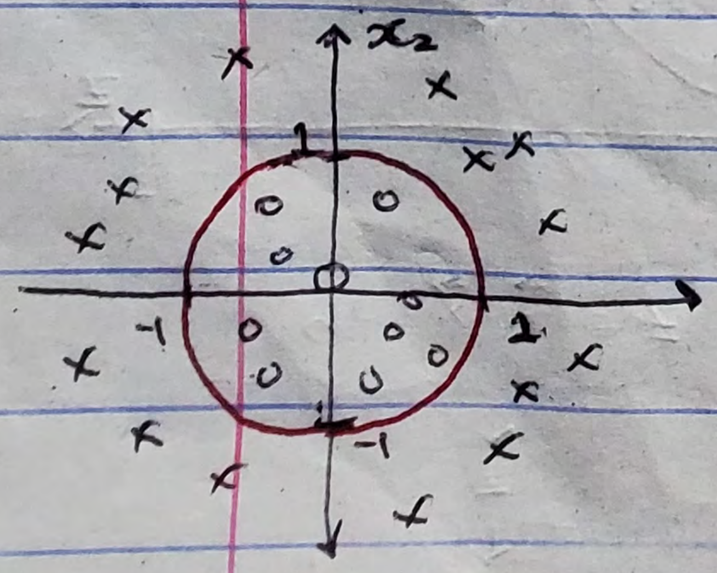
$z = \vec{w}\cdot\vec{x}+b = 0$

Decision boundary: $z = x_1 + x_2 - 3 = 0$

$x_1 + x_2 = 3$



NON-LINEAR DECISION BOUNDARIES



Decision boundary

$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underbrace{\underset{1}{w_1}x_1^2 + \underset{1}{w_2}x_2^2 + \underset{-1}{b}}_{z})$

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

$x_1^2 + x_2^2 = 1$

WK3 | COST FUNCTION for LOGISTIC REGRESSION

Training set:

tumor size cems x_1	...	Patient's age x_n	malignant? y
10		52	1
2		73	0
5		55	0
12		49	1
...		...	...

training example

$i = 1, \dots, m$

$j = 1, \dots, n$ — features

target $y = 0$ or 1

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

Research Question: Given the training set, how can you choose parameters $\vec{w} = [w_1 w_2 \cdots w_n]$ and b?

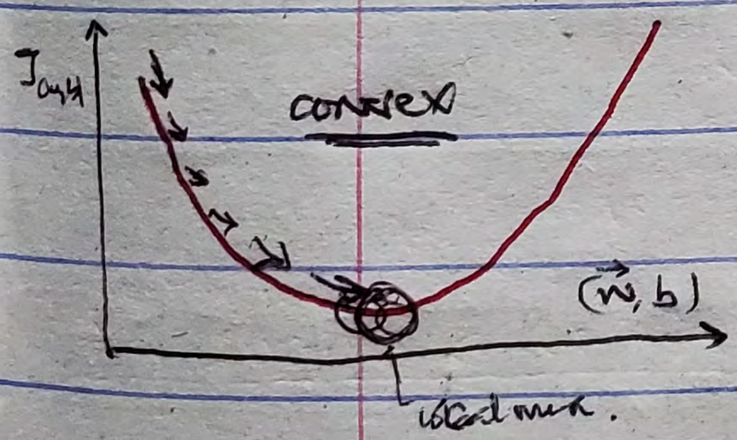
recall that for linear regression, we used the SQUARED ERROR COST function

$$J_{(\vec{w},b)} = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left[f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right]^2$$

If we were to use gradient descent to find the ~~local~~ local minima & plot the cost, $J_{w,b}$ for each value of $\vec{w}, b$ (considering b a constant); we'd get

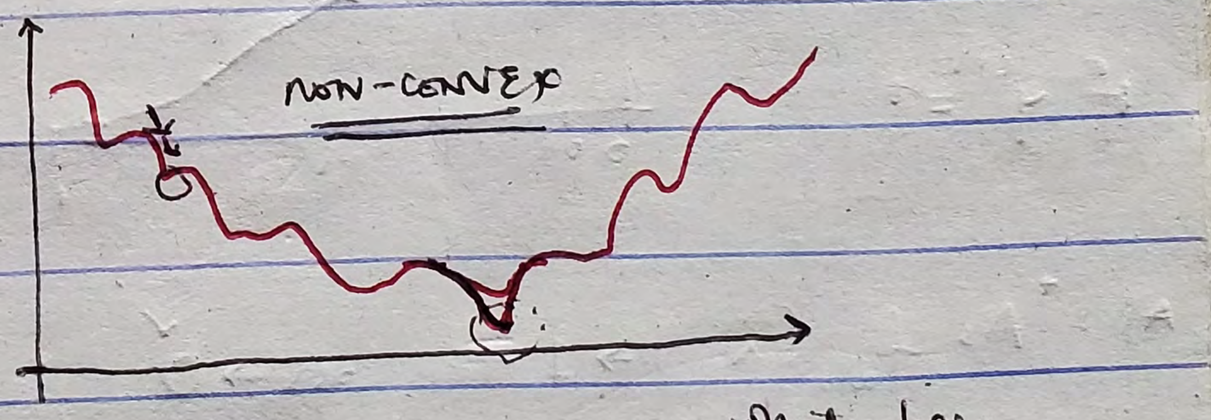
linear regression

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



Logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$



using Squared error cost funct results in a ~~for~~ non-convex shape & which has lots of false local minima & prevents gradient descent from reaching the true minimum.

∴ Squared error cost function not good for LOGISTIC regression

To get our new cost function: Let's first look @ the Squared cost function & change its definition a bit.

Looking @ the expression, $J_{(\vec{w},b)} = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left[f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right]^2$

inside the summation

→ this is the Loss on a single training example

Loss; $L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)$ ← denoted by

Definition of loss, L; $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$ is a function of the prediction of the learning algorithm $f_{\vec{w},b}(\vec{x})$ as well as of the true label $y^{(i)}$

∴ In squared error cost function; the loss, given the predictor $f_{\vec{w},b}(\vec{x})$ and the true predictor y is equal to $\frac{1}{2}$ of the squared difference.

— Choosing a different form for the loss function will help us keep the overall cost function $= \frac{1}{m} \times$ sum of loss functions, to be a convex function.

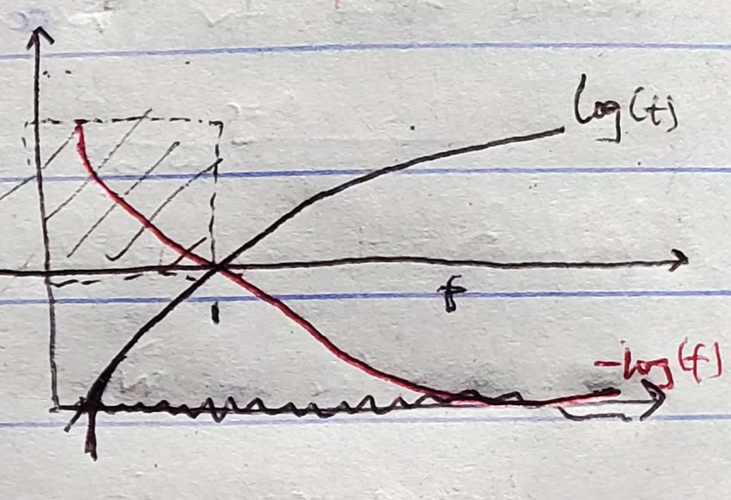
∴ Recall the loss function inputs $f(x)$ & y and tells us how well we're doing on that example -

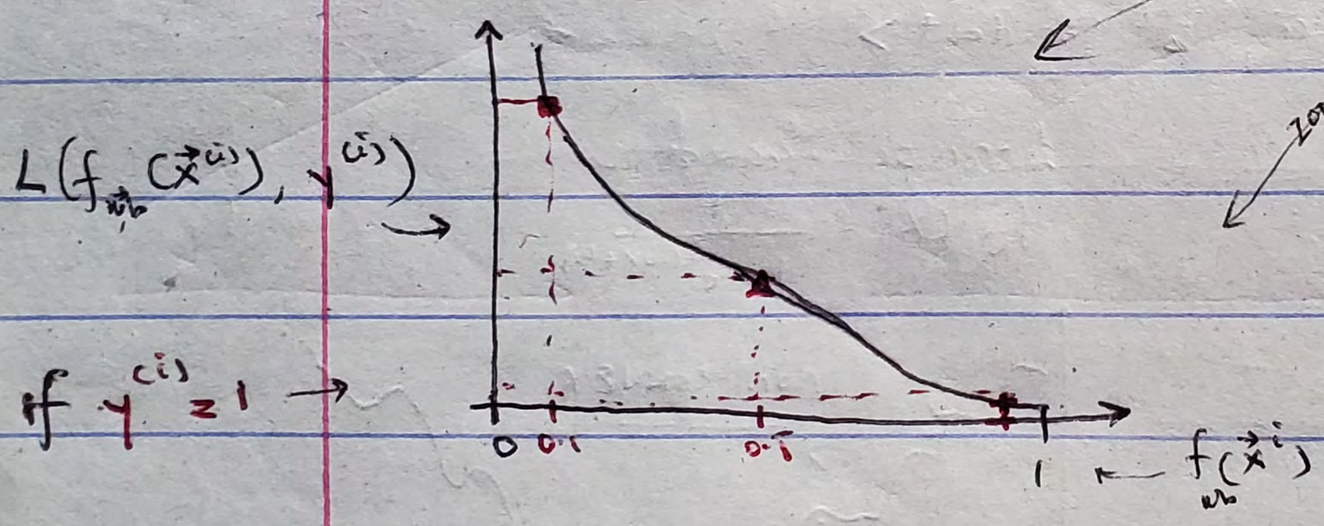
Logistic Loss function: $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{for } y^{(i)}=1 \\ -\log(1-f_{\vec{w},b}(\vec{x}^{(i)})) & \text{for } y^{(i)}=0 \end{cases}$

∴ plotting the output of the log reg. f gives:

∴ f is always between 0 & 1

∴ only the shaded portion is useful to us





A) As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to 0$ ✓ ⟹ If the algorithm predicts a probability close to 1 & the true label is $y^{(i)}=1$, then the loss ≈ 0.

- If the output, $f_{\vec{w},b}(\vec{x}^{(i)}) = 0.5$ when $y^{(i)}=1$ then the loss is a bit higher
- If the output, $f_{\vec{w},b}(\vec{x}^{(i)}) = 0.1$ when $y^{(i)}=1$, then loss is really high

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

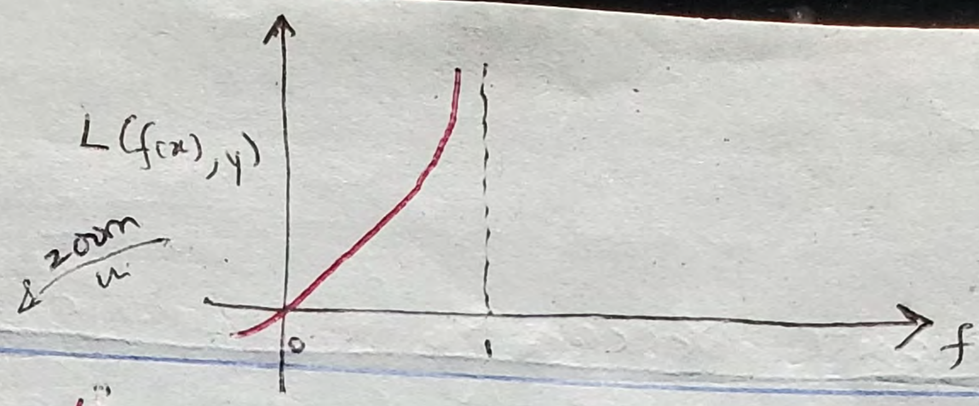
For $y^{(i)}=1$, the Loss is lowest when $f_{\vec{w},b}(\vec{x}^{(i)})$ predicts close to true label $y^{(i)}=1$

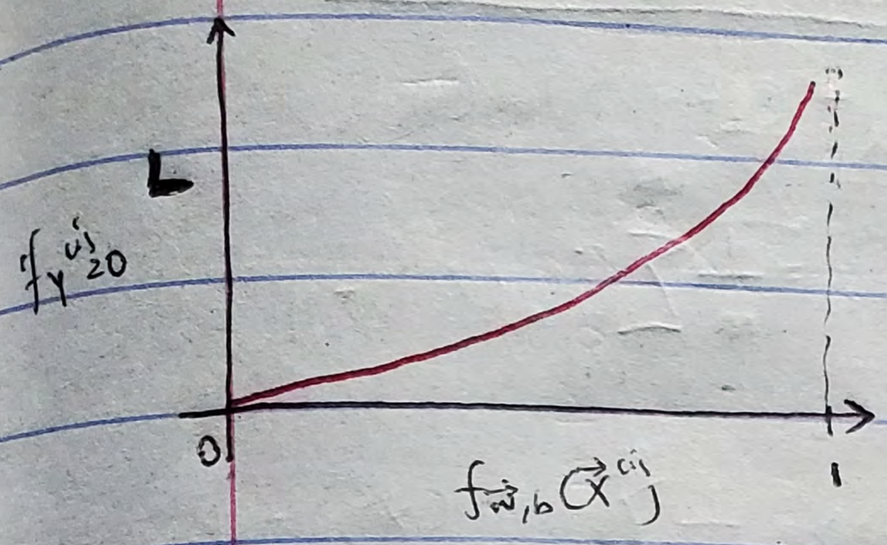
Same for $y^{(i)}=0$

* Now let's look @ the second part of the loss function

$$-\log\left[1 - f_{\vec{w},b}(\vec{x}^{(i)})\right] \quad \text{if } y^{(i)}=0$$

When the function is plotted →





When $f(x) = 0$, then the loss, $L = 0$. The larger the $f(x)$ gets, the bigger the loss. As the prediction, $f(x)$ approaches 1, the Loss approach ∞

⇒ As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to \infty$ <bad boy :>

∴ for $y^{(i)} = 0$, the further the prediction $f_{\vec{w},b}(\vec{x}^{(i)})$ is from the ~~prediction~~ target $y^{(i)}$, the higher the loss

⇒ From our loss function, we can see that we're incentivizing our algorithm to predict the right value by giving it a lower loss & punishing it for predicting a false value by giving it a higher loss

⇒ The Logistic loss function gives us a convex shape when we plot the cost function $J(\vec{w}, b)$. Note that the cost, $J(\vec{w}, b)$ is the average of the loss for all the training examples.

SIMPLIFIED COST FUNCTION for LOGISTIC REGRESSION

Recall our initial loss fnt, $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)}=1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)}=0 \end{cases}$

↳ it can be re-represented as;

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

This re-representation is the same as above because $y^{(i)}$ can only take either 1 or 0

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

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

* Note we chose this cost function using Maximum Likelihood Estimation which is an idea from statistics on how to efficiently find parameters for models.

WK3 GRADIENT DESCENT for LOGISTIC REGRESSION.

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

repeat $\left[\begin{array}{l} w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\vec{w},b) \\ b = b - \alpha\frac{\partial}{\partial b}J(\vec{w},b) \end{array}\right.$

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

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

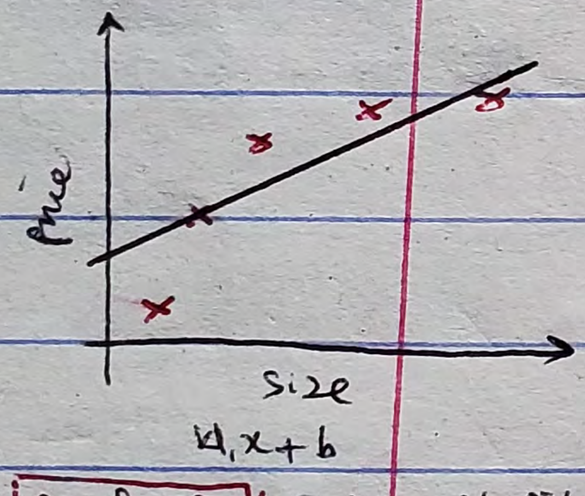
Although the algorithm written looks the same for Linear Regression & Logistic Regression

recall that Linear regression: $f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$

logistic regression: $f_{\vec{w},b}(\vec{x}) = g(f(x)) \text{ or } g(z) = \text{sigmoid}(f\vec{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$

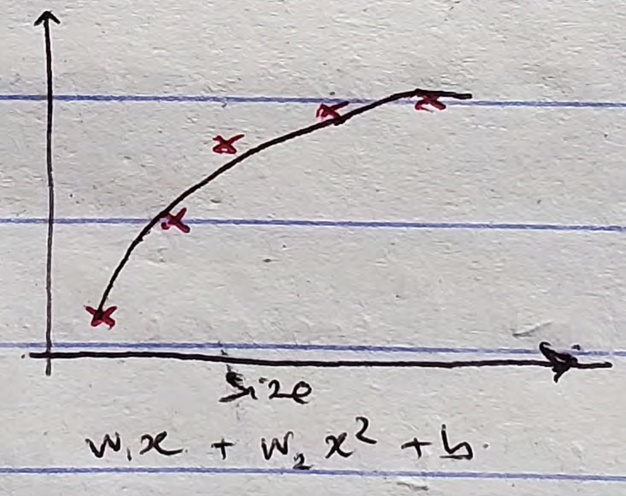
- Monitor gradient descent using a learning curve.

WK3 THE PROBLEM OF OVERFITTING



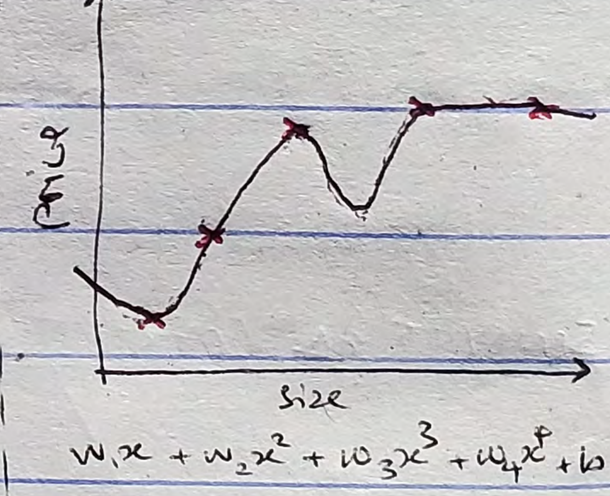
$w_1x + b$

- Underfit
or high bias
- Doesn't fit the transcet well
- the model has a preconceived notion of what the relationship should be despite the data showing otherwise



$w_1x + w_2x^2 + b$

- Fits training set pretty well
- Model is Generalized, i.e. it makes ~~good~~ predictions even on brand new examples it hasn't seen before
- This is GENERALIZATION

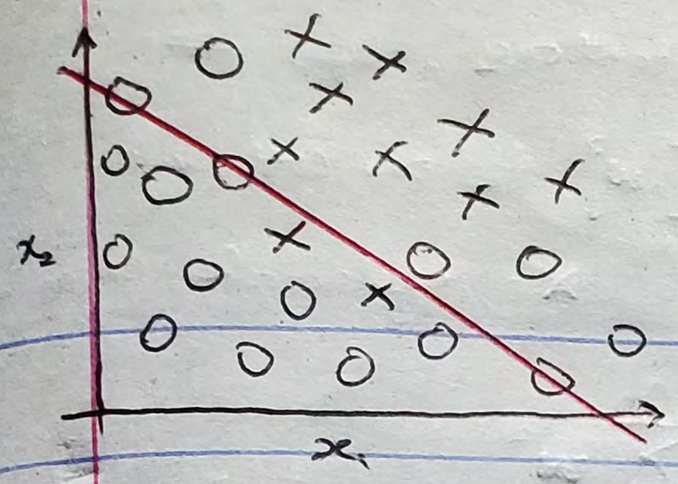


$w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

- Fits the training set extremely well
- It fits the model too well that it isn't generalized
- OVERFITTING or HIGH VARIANCE

- In ML, people use underfit & highbias interchangeably & overfit & their variance interchangeably.
- The In overfitting, the algorithm is trying very hard to fit every example and it turns out that if the training set was a little different, the function that the algorithm fits would be totally different & that results in highly variable predictions

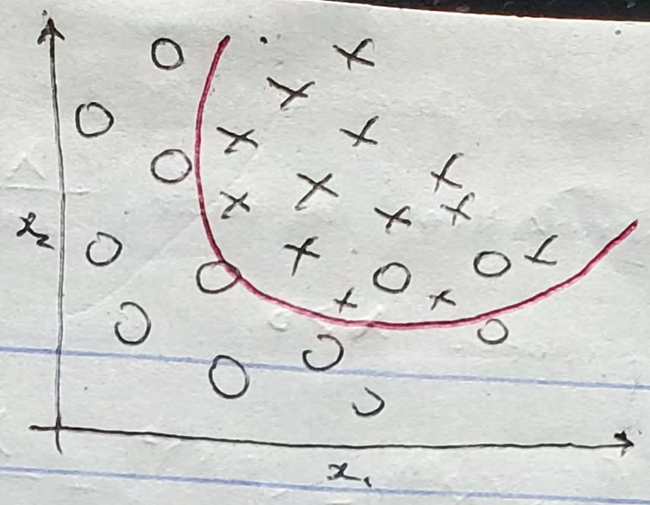
- the same applies in classification



$z = w_1x_1 + w_2x_2 + b$

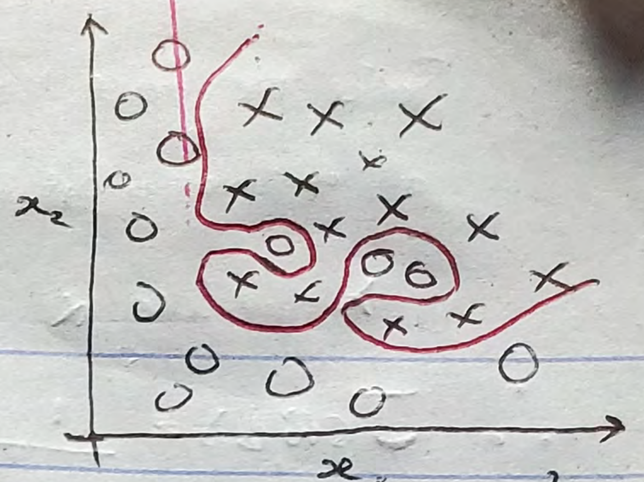
$f_{\vec{w},b}(\vec{x}) = g(z)$

UNDERFIT ✗



$z = w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2 + b$

Just right ✓



$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + w_6x_1^3x_2 + \dots + b$

OVERFIT ✗

A: ADDRESSING OVERFITTING

#1. Collecting more training data helps address overfitting [more training data for OVERFITTING]

#2. Reducing the No of features used also addresses overfitting

Lots of features + Insufficient data → OVERFIT

(Select features) + Insufficient data → Just right

(given those features relative)

#3. Regularization: A way to more gently reduce the impact of some of the features w/o doing something as harsh as eliminating it outright. It encourages the learning algorithm to shrink the values of the parameters w/out necessarily demanding that they're set to exact 0

→ It lets you keep all your features but prevents them from having an overly large effect which what mostly cause OVERFITTING

WK3 REGULARIZATION

- If you have a lot of features, n features. You wouldn't know which features are important and which should be penalized so regularization penalizes all the w_j parameters. This results in a smoother & simpler function less prone to overfitting

Example: Let's say we have these features;

size x_1	bedrooms x_2	floors x_3	age x_4	avg income x_5	...	dist to coffee shop x_{100}	price y

we have n features, $n \geq 100$ ∴ reg parameters = $w_1, w_2, w_3, \dots, w_{100}, b$

Since we don't know which parameters are the important ones, let's penalize all of them a bit and shrink them by adding $\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$ to our cost function

λ = regularization parameter

λ : just like α, can be chosen ($\lambda > 0$)

$$\therefore J(\vec{w}, b) = \underbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}_{\text{(i) mean squared error}} + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}_{\text{regularization term (ii)}}$$

λ is divided by $2m$ to scale (i) & (ii) so as to help us choose a good value for λ. Therefore even if m increases, the same value of λ is also more likely to work because of the extra scaling $\left(\frac{1}{2m}\right)$.

- Regularizing the b term (regression parameter) doesn't really change anything.

∴ In the modified cost function, we want to minimize the original cost ($J_{(w,b)} = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$ ≡ Mean squared error cost)

+ additionally the regularization term

└ This has 2 goals → (i) minimize the mean squared error encourages the algorithm to fit the data well by minimizing the loss

└ (ii) Minimizing the regularization term encourages the algorithm to keep the parameters w_j small which tends to reduce overfitting.

The value of λ (chosen date) determines how you balance the two goals.

(underfit) Increasing λ tends to cause your learning algorithm to decrease the size of w_j

- Decrease λ (overfit) tends to cause your learning algorithm to increase the size of w_j

WK3 REGULARIZED LINEAR REGRESSION

Our aim find parameters $\vec{w}$ & b that minimize the cost function $J(\vec{w}, b)$

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

Previously, we used gradient descent for the mean squared error portion only now we have the regularization term -

repeat {

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

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b) \rightarrow = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$ this stays the same because we don't regularize b

Let's look @ w_j <note that I've substituted the terms for $\frac{\partial}{\partial w_j} J(\vec{w},b)$ in & I'm grouping like terms>

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

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

→ recall that the way our initial unregularized value for $\alpha\frac{\partial}{\partial w} J(w,b)$

∴ The only change in the ~~regularized gradient descent~~ gradient descent for regularized linear regression is that instead of $w_j - \alpha\frac{\partial}{\partial w} J_{(w,b)}$ we now have $w_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{\partial}{\partial w} J_{(w,b)}$ [I hope you understand as I do]

Note: In differentials between unregularized & regularized linear regression

∴ @ every update, instead of having w_j, you have e.g. $w_j\left(1 - 0.01\frac{1}{50}\right)$ → $0.9998\, w_j$

this effectively shrinks w_j ~~after each~~ on every iteration

WK3 REGULARIZED LOGISTIC REGRESSION

Our modified cost function that uses regularization is;

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

- Minimizing the cost function $\min_{\vec{w},b} J(\vec{w},b)$ penalizes parameters $\vec{w}$ & prevents them from being too large
- For gradient descent, our gradient $\frac{\partial}{\partial w} J(\vec{w},b)$ is;

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

$wx+b$

- It's the exact same equation as the linear regression except for the fact that $f(x) = g(z)$