LOGISTIC REGRESSION

START FROM HERE

PREREQUISITE For Deep learning

Logistic Regression;—

① Logistic Regression can be seen as a model for a ~~Neuron~~ neuron

② Logistic Regression are combined togeather to give us a neural network As such it is the foundation for deep learning.

Statistics VS Machine Learning;—

Machine Learning;—

① Linear models are a tool to introduced common ML techniques

② Loss function

③ Minimizing the loss function using calculus

④ Train and test sets

⑤ Hyperparameters, generalizations, overfitting, regularization.

⑥ Same techniques apply on neural networks, SVM etc.

Statistical approach;—

① You will spend your entire degree learning about linear models.

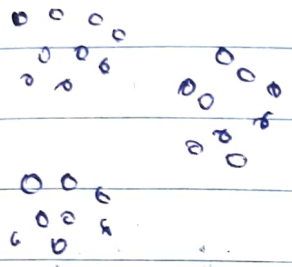
Machine Learning :-

Machine Learning

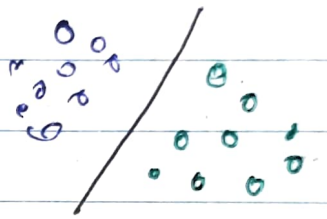
Unsupervised

Supervised

(1)



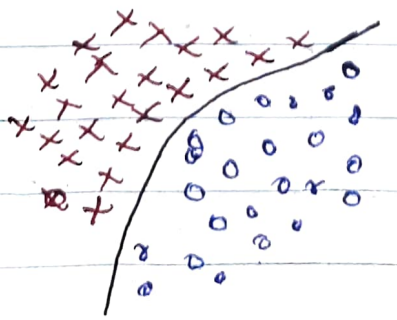
(2)



(1) In unsupervised learning you usually don't have a label so you are forced to split the data into groups by looking for patterns.

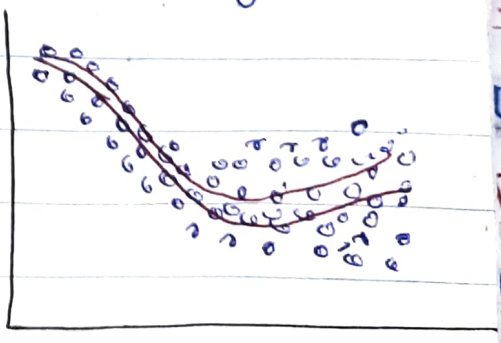
(2) In supervised learning we already have a way to split the data

Classification



Trying to split the data into categories

Regression



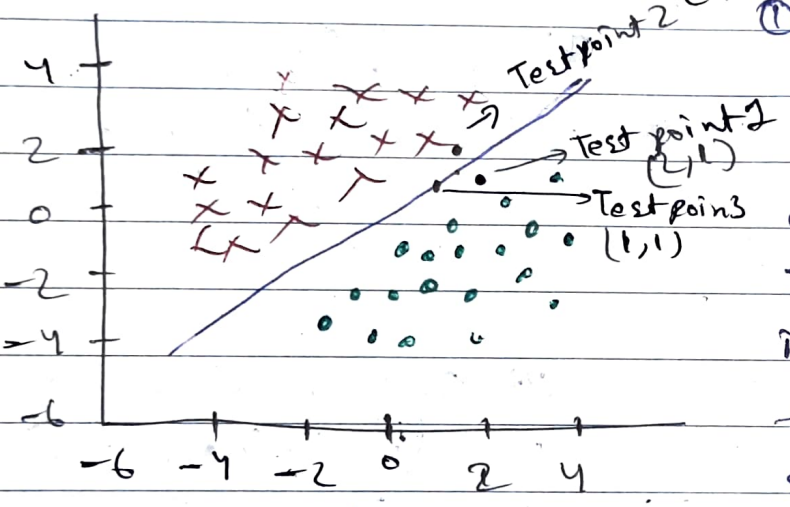
Trying to predict the value of some function.

Logistic Regression:-

① It is a model that is used for classification

② example image classification

Linear classification:-



① $y = mx + b$

OR

$0 = ax + by + c$

$a = 1, b = -1, c = 0$

If our X and O are split in 45° line ~~then and passes the line intercept or set to~~ 0

① You can see that If ~~you~~ our X's & O's are split along a 45° line that crosses the line to ~~eat~~ set to zero then $A = 1$ $B = -1$ & $C = 0$ should be the line so in other words our line is $0 = X - Y$

③ So Now let's say we have a test point at $X = 2$ & $Y = 1$

If we ~~put~~ plug that into

~~$h(x,y) = x - y$~~ $h(x,y) = x - y$

$= 2 - 1 = 1 \geq 0 \rightarrow$ '0'

There fore we should classify this test point as an '0'

④ Now lets say we have another testpoint (testpoint 2) at $x=1$ $Y=2$. If we plug that into

$h(x,y) = X-Y$

$= 1-2 = -1 < 0 \rightarrow$ 'x'

Therefore we should classify this test point 2 as an 'x'

⑤ Now lets say we have another test point (test point 3) at $x=1$, $Y=1$. If we plug that into

$h(x,y) = X-Y$

$= 1-1 = 0 \rightarrow$ don't know

There fore we cannot tell wheather testpoint 3 is an X or O

Understanding Linear classification in machine learning terms;—

We call $(x,y) \rightarrow (x_1, x_2)$

we rename A, B & C constoints to w_i

we call the bias term/intercept w_0

$h(x) = w_0 + w_1 x_1 + w_2 x_2$

we say h() hypothesis function is a linear combination of the components of x

In vector form : $h(x) = w^T x$ (Dot product)

In 3 Dimension : line is transformed into a plane

In > 3 dimension : hyperplane

So If we have a function which is not a line function separating our data we could call that a hyper surface.

Architecture of neurons:-

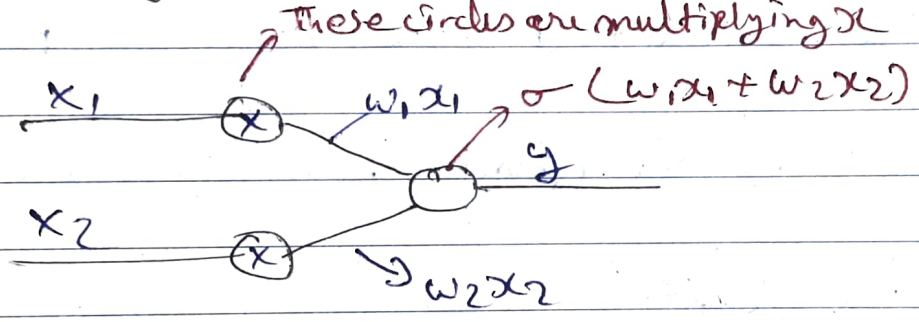
(1) neural network is a network of neurons.

(2) Many Input → One output

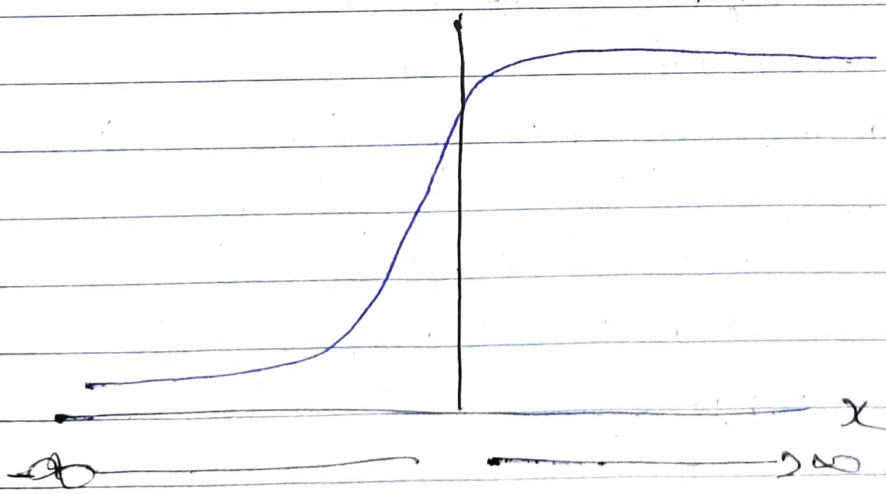
(3) neurons have an excitability threshold
If a neuron (voltage) passes this threshold then a spike occurs otherwise there is no spike. spike = 1 or no spike = 0

(4) Synapses → linear weight

Feed forward calculation: How do you get a prediction using logistic regression



Logistic function = Sigmoid function



a sigmoid function is an S shape it has a finite limit as X approaches ∞ in a finite limit as X approaches −∞

~~type~~ There are number of possible sigmoid function but the two commonly used are hyperbolic tangent

$\tanh(x) \in (-1, 1)$ $y\text{-int} = 0$ this goes from -1 to 1 and its y intercept is zero

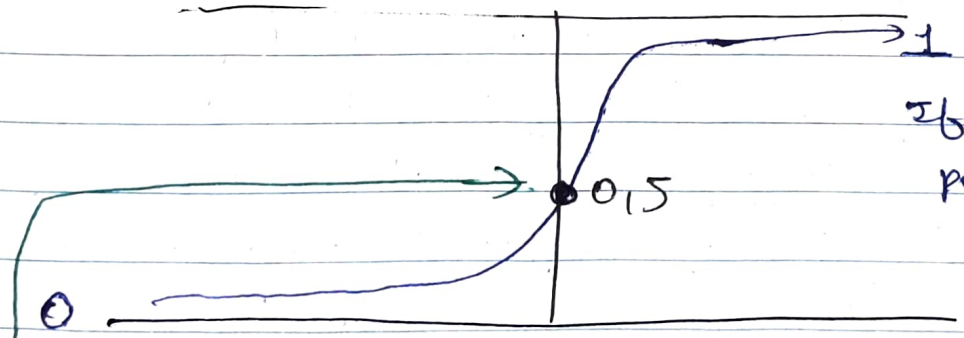
The other function which is the one we will be using is commonly denoted by the σ symbol

$$\sigma(z) = \frac{1}{1+e^{-z}} \in (0,1) \quad y\text{-int} = 0.5$$

This goes from 0 to 1 & its y intercept is 0.5

So we can combine these to say the output of a logistic regression is $\sigma(w^T \cdot x)$

so what does this function tell us so if the inner product of w & x is very positive we will get a number very close to 1



we get a number very close to 0 the value of y equals 0.5 means the inner product is zero which means we are right in the boundary of between the two classes. So in other words in probability of belonging to the class is 50%.

Difference between logistic Regression and general linear classifier we talked ~~out~~ about earlier =

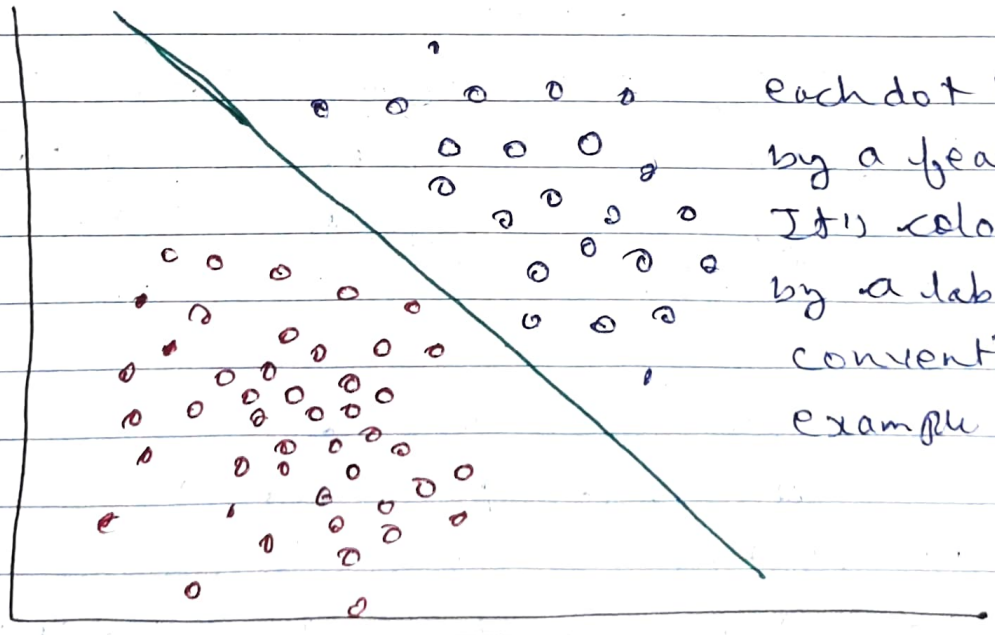
(1) we have this logistic function at the end which gives us a number between 0 & 1 when we do our classification we can say anything above 0.5 will gives class 1 and anything else will give us class 0

What does the output mean?

$$\text{Output} = \sigma(W^T \cdot x) \in (0,1)$$

The output of a logistic regression ~~is a~~ is a sigmoid from the sigmoid function, we are going to go get a no between 0 & 1 in deep learning This has a nice & intuitive interpretation.

Recollection of what we are trying to do in classification



each dot is represented by a feature vector (X) It's color is represented by a label (y)

convention : y=0 or 1

example Blue = 1, Red=0

① We have some red dots and we have blue dots & we have a line that separates them.

② Each dot is represented by a feature vector x and its color or in other words its label is represented by the letter y.

③ As per convention y is assigned a value 0 or 1

④ The output of logistic Regression is a number between 0 and 1 and we interpret this as

① $p(y=1|x) = \sigma(w^T, x) \in (0,1)$

② The probability that y equals 1 given x

Now because y can only take on two values 0 or 1 that means ~~pur~~ y equals 0 given x is just 1 - P of y equals 1 given x in other words the two probabilities must sum to 1

↳ y can only have 2 values

$p(y=0|x) = 1 - P(y=1|x)$

↳ i.e.

$p(y=0|x) + p(y=1|x) = 1$

↳ $p(y=1|x) = \sigma(w^T, x) \in (0,1)$

⑤ So this gives us a handy way of making predictions

if $p(y=1|x) > p(y=0|x)$:

predict class 1

else:

predict class 0

(e) It's easy to show that an equivalent way of making this prediction is just to say if P of y equals one given x is greater than 0.5 then predict class 1 otherwise predict class 0

```
if P(y=1|x) > 0.5:
        predict class 1
else:
        predict class 0
```

→ Here we are using 0.5 because =

$$1 = \underbrace{p(y=1|x)}_{\text{class 1}} + \underbrace{p(y=0|x)}_{\text{class 2}}$$

$$0.5 + 0.5 = 1$$

~~Sowe~~ So if we want to predict classes then initially we were doing

```
if P(y=1|x) > P(y=0|x):        → we can replace this part with 0.5
        predict class 1
else:
        predict class 0
```

and our statement becomes as below

```
if P(y=1|x) > 0.5:        → 0.5 here is a threshold
        predict class 1
else
        predict class 0
```

⑦ So using 0.5 as a threshold it is easy to make a prediction without using an if statement. Instead we can round the output of logistic regression

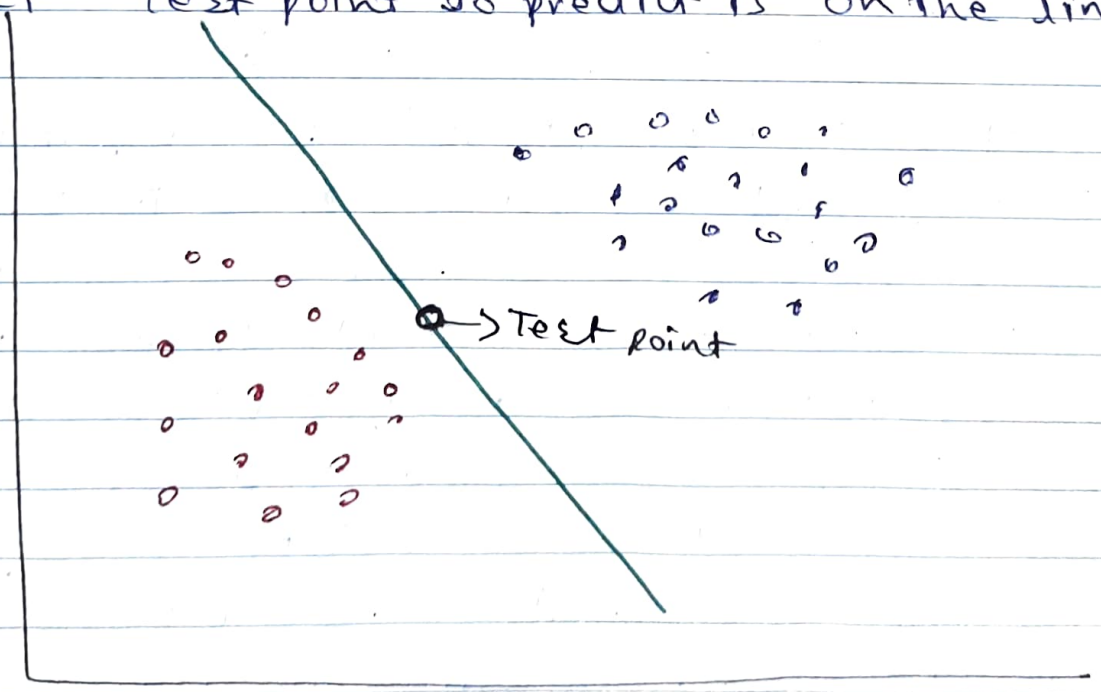
⑧ Rounding this number will give us either 0 or 1 which is exactly what we want. So now you can see the advantage of using 0 OR 1 as our labels.

⑨ Note:- Although it is ~~possible~~ possible to use number other than 0.5 as a threshold that usually doesn't happen too often in deep learning

⑩ Predict round ($P(y=1|x)$) → gives us either 0 OR 1

Another reason why ~~the ou~~ interpreting the output of the logistic regression as a probability makes sense →

Case I:- Test point to predict is on the line

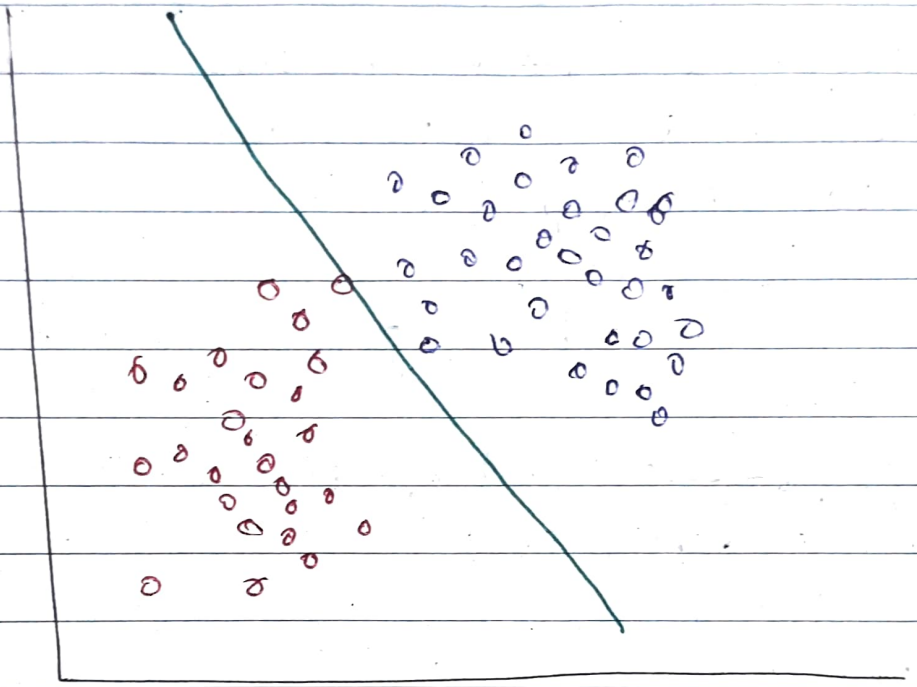


① If a point lies directly on the line the probability is exactly 0.5

② well if the point lies directly on the line we can't

really say either way whether this point should be assigned RED OR BLUE class

Case(II) What happens when a point (Test point) gets further and further away from the line.



① $|W^T.X|$ gets larger as we move away from the line i.e the magnitude of w transpose X gets larger and larger. In other words the arguments into the sigmoid approaches ∞

② The probability that y equals one given X approaches 1 or 0 depending on which way you go. $P(y=1|x)$ gets closer to 1 or 0

③ ~~The further~~ (we know that right in the middle it ~~can~~ means it can equally be red or blue. Thus it also makes sense that the further you go away from the line the stronger your belief gets that the target point is either Red or Blue

Quiz:- Absence of sigmoid

① To get an output predictions from a logistic regression we do the dot product between our weights & input features x and pass that to the sigmoid function

* $y = \sigma(w^T . x)$ → dot product between $w^T . x$

σ = sigmoid function

→ This is used in forward function to make predictions in logistic regression

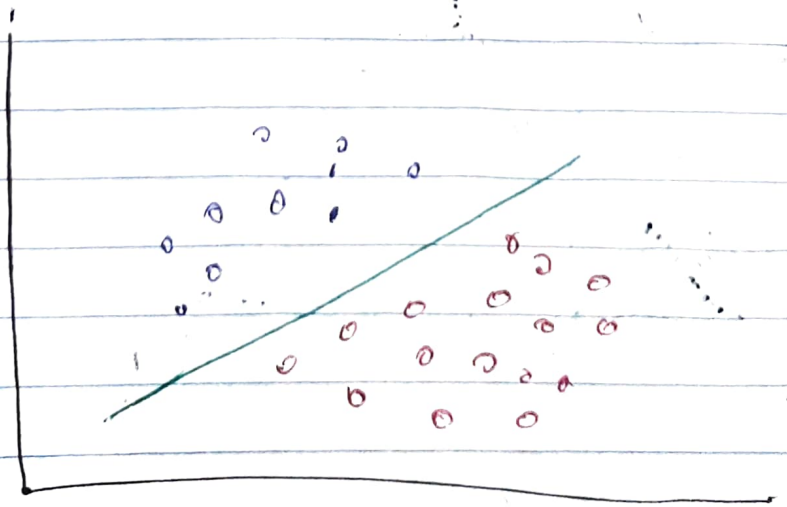
② The Question is what kind of model do we get if we don't use the sigmoid?

$$y = w^T . x$$

The answer is:- we get a model that looks like linear regression. ~~It is~~ The only thing that makes this like linear regression is it's functional form. It's a linear combination of input features weighted by the weight

③ Logistic regression makes the assumption that your data can be separated by a line / plane

$$y = \sigma(w^T . x)$$



④ That's what we mean by modeling. we make some assumptions about the mathematical form and all the work we do thereafter follows from these assumptions.

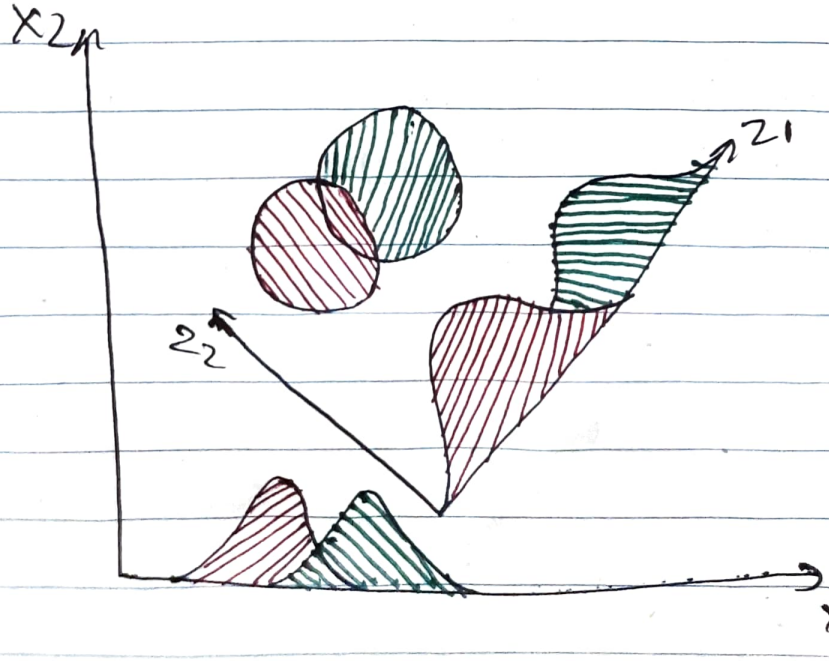
⑤ we have these weights we have n't found them OR choose them we just randomely select the weights using random function of python. we have to find weights using probability

Solving for the optimal weights

① In machine learning finding weights w for our model is called "training"

② Training OR learning is finding the parameters of your model that make it accurate

Special case of logistic Regression:-



① Here we are going to look at a special case of logistic regression one where we can find a closed form solution to the problem

② In general we cannot and we will have to use gradient decent but the solution here could be useful if your data is distributed in a specific way.

③ The problem is this:-

→ your data is from 2 classes & they are both Gaussian distributed

→ They have the same same covariance but different means

→ For this example you should be familiar with multivariate Gaussian distribution

~~Bayes~~ Bayes' Rule:-

ⓐ $p(Y|X) = p(X|Y)\,p(Y)\,/\,p(X)$ (① ② ③ ④) This just says that the posterior or p of y given x (①) is equal to the likelihood which is p of x given y (②) → gaussian times the prior or p of y (③) divided by p of x (④)

ⓑ The likelihood part is the gaussian that we just talked about

ⓒ We can calculate gaussian by taking the data from each class and getting their means and covariances the prior can just be the maximum likelihood estimate. p(X|Y) is the gaussian - we calculate it over all the data that belongs to class Y.

ⓓ P(Y) is just the frequency estimate of Y →

example = p(Y=1) = # times class 1 appeared / total

④ The next thing we can do is take Bayes rule for the +ve class and expand the bottom portion so it looks like the top portion

$$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x)} = \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1) + p(x|y=0)p(y=0)}$$

Next we divide the top & bottom by what's on top

$$p(y=1|x) = \frac{1}{1 + \frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)}}$$

⑤ $$p(y=1|x) = \frac{1}{1 + \frac{p(x|y=0)(p(y=0))}{p(x|y=1)p(y=1)}}$$

$$= \frac{1}{1 + \exp(-\{w^T x + b\})}$$ → sigmoid used in programs

$$-(w^T x + b) = \ln\left(\frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)}\right)$$ → log of this ratio from before

⑥ Let $p(y=1) = 1-\alpha$ and $p(y=0) = \alpha$

$$\ln\left(\frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)}\right) = \ln p(x|y=0) + \ln\alpha - \ln p(x|y=1) - \ln(1-\alpha)$$

$$= \ln \frac{1}{\sqrt{(2\pi)^p |\Sigma|}} e^{-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)} - \ln \frac{1}{\sqrt{(2\pi)^q |\Sigma|}} e^{-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)} + \ln\left(\frac{\alpha}{1-\alpha}\right)$$

Square root cancels out;-

$$= -\frac{1}{2}\left(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_0 - \mu_0^T\Sigma^{-1}x + \mu_0^T\Sigma^{-1}\mu_0\right) + \frac{1}{2}\left(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_1 - \mu_1^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}\mu_1\right) + \ln\frac{\alpha}{1-\alpha}$$

The quadratic term

$$= \mu_0^T\Sigma^{-1}x - \mu_1^T\Sigma^{-1}x$$

$$-(\mu_0^T + \mu_1^T)\Sigma^{-1}x = -w^T$$

$$= (\mu_0^T - \mu_1^T)\Sigma^{-1}x$$

$$= -(w^T x + b)$$ → It depends on x

If you split up the

↳ $w^T = (\mu_1^T - \mu$

$b = \frac{1}{2}\mu_0^T\Sigma^{-1}\mu$

⑦ During coding centered at (−2

The variance each dimension

$\mu_0 = (-2,-2)^T$,

$w = (4,4), b = 0$ (

each class)

so when you

⊙

The quadratic term cancells out

$$= \mu_0^T \Sigma^{-1} x - \mu_1^T \Sigma^{-1} x - \frac{1}{2}\mu_0^T \Sigma^{-1}\mu_0 + \frac{1}{2}\mu_1^T \Sigma^{-1}\mu_1 + \ln\frac{\alpha}{1-\alpha}$$

$-(\mu_1^T + \mu_1^T)\Sigma^{-1}x = -w - w^T x$

$$= \boxed{(\mu_0^T - \mu_1^T)\Sigma^{-1}x} \quad \boxed{-\frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \ln\frac{\alpha}{1-\alpha}}$$

$$= -(w_1^T x + b)$$

w_1: It depends on x ; b: It does not depends on x → sigmoid used in prog ram

$b = -\left(\frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 - \ln\frac{\alpha}{1-\alpha}\right)$, $-b$

If you split up the terms w & b then

$$\rightarrow w^T = (\mu_1^T - \mu_0^T)\Sigma^{-1}$$

$$b = \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 - \ln\frac{\alpha}{1-\alpha}$$

(7) During coding we will use two gaussians one centered at $(-2,-2)$ & one centered at $(+2,+2)$

The variance of each dimension will be 1 & each dimension will be independent

$\mu_0 = (-2,-2)^T$, $\mu_1 = (+2,+2)^T$, $\sigma^2 = 1$

$w = (4,4)$, $b = 0$ (assume ~~equal~~ no of samples of each class)

So when you ~~off~~ diagonals covariance will be 0

$$\Sigma^{-1}(x-\mu_1) - \ln\left(\frac{\alpha}{1-\alpha}\right)$$

Variables generally used in Machine learning & Deep learning:-

1. N = Number of samples we've collected
2. D = Number of dimensions OR Features per sample
   ↳ measure hight, weight & girth to try to predict body fat %. D=3 (height, weight & girth)
3. X = matrix of data is an NXD matrix
   Sample ← row, column → value of D feature in each sample
4. Sometimes we say that we train model on inputs X and targets Y
5. If we consider all Ys at the same time, then it is an NX1 matrix (for binary classification, an NX1 matrix 0s & 1s)
6. Sometimes we use T for target instead
   ↳ i.e The cross-entropy error functions!-

$$J = -\sum [t \log(Y) + (1-t)\log(1-Y)]$$

Ytest OR Ytrain = Targets; → Pytrain = P of Y given X

7. Now Y is used for something else;- output of logistic regression
8. Before the output was more precise $P(Y=1|X)$
9. Cost function = error function = objective function
10. We want to minimize the cost ~~function~~ or error - easy to see semantically
11. Objective function can mean something to maximize OR minimize
12. We will first maximize likelihood - P(data|model)

i.e If A > B, then log(A) > log(B)

(13) Sometimes we use letter J
↳ may be in form of J = L → maximize it
↳ OR J = L → minimize it.

Cross Entropy error function;—
linear Regression - squared error
$J = \Sigma_n (t_n - y_n)^2$
Assumes Gaussian ~~error~~ distributed error, because log(Gaussian) = squared function
Logistic Regression error can't be Gaussian distributed because;—
- Target is only 0/1
- Output is only a number between 0-1

we want; 0 if correct, >0 if not correct, more wrong == bigger cost
Cross entropy error function is used in logistic Regression instead of squared function
Cross entropy error function →

$$J = -\{t \log(y) + (1-t)\log(1-y)\}$$ (t = target, y = output of logistic)

If t=1 only first term matters, if t=0 only second term matters
log(y) → number between 0 and -infinity
Note;- we want to calculate the total error to optimize over all the training data simultaneously

$$J = -\sum_{n=1}^{N} t_n \log(y_n) + (1-t_n)\log(1-y_n)$$

Scalar random variable: $X \sim N(\mu_x, \sigma_x)$

Vector of Random variable: $\underline{X} \sim N(M, \Sigma)$ — Normal distribution

M = mean value

Σ = covariance matrix

$$\underline{X} = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} \sim N\left(\begin{bmatrix} \mu_{x_1} \\ \mu_{x_2} \\ \vdots \\ \mu_{x_n} \end{bmatrix}, \begin{bmatrix} \sigma^2_{x_1} & \sigma_{x_1 x_2} & \cdots & \sigma_{x_1 x_n} \\ \sigma_{x_2 x_1} & \sigma^2_{x_2} & \cdots & \\ \vdots & & & \\ \sigma_{x_n x_1} & \cdots & \cdots & \sigma^2_{x_n} \end{bmatrix} \right)$$

bold or underline is gonna indicate a vector or matrix

Closed form solution

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

Inverse of a matrix is one of the most computationally expensive method in computing. So if you have a very big data inverse of a matrix might not even fit in your memory. Hence we don't use closed form solution when our data is very big.

How to calculate likelihood for the biased point:-

① So we have a coin which have a probability of heads equal to peace so it's unknown and the probability of tails is of course $1-p$ i.e $P(T) = 1-p$

$P(H) = p$

② We're going to do an experiment to help us determine p we flip a coin 10 time & we get

$N = 10$ $\quad 7H \cdot 3T \rightarrow$ 3 Tails

$\downarrow$ 7 Heads

④ How would we write the total likelihood which is the probability of seeing the result that we saw.

$L = p^7 (1-p)^3$

$(1-p)$: Probability of tails

we can do this because each coin tosses independent so we can multiply each individual probability.

⑤ Now what we want to do here is we want to maximize L with respect to p so we choose a p that maximizes out and to do this we will use calculus.

⑥ Now most of these problems we take the log and maximize the log likelihood which is acceptable because the log function is monotonically increasing meaning that the point at which the likelihood is maximum. the log likelihood is also maximum.

$l = \log [p^7 (1-p)^3] \rightarrow$ log likelihood

$= \log p^7 + \log (1-p)^3$

(6) $= 7 \log p + 3 \log(1-p)$

$$\frac{\partial l}{\partial p} = \frac{7}{p} + \frac{3}{1-p}(-1) = 0$$

$$\frac{7}{p} = \frac{3}{1-p}$$

$$\frac{1-p}{p} = \frac{3}{7}$$

$$\frac{1}{p} - 1 = \frac{3}{7}$$

$$\frac{1}{p} = \frac{10}{7}$$

$$p = \frac{7}{10} = p(H)$$

(7) So now apply a similar idea to logistic regression. we have a probability that y is equal to 1 so we can think of it as heads.

$p(y=1|x) = \sigma(w^T x) = y$

$$L = \prod_{n=1}^{N} y_n^{t_n} (1-y_n)^{1-t_n}$$

t_n: Target=1, $1-t_n$: Target=0

taking log likelihood:

$$l = \sum_n t_n \log y_n + (1-t_n)\log(1-y_n)$$

we get the same form as a cross entropy error function.

⑧ So In other words maximizing the log likelihood is same as minimizing the cross entropy error

Updating the weights using gradient descent :-

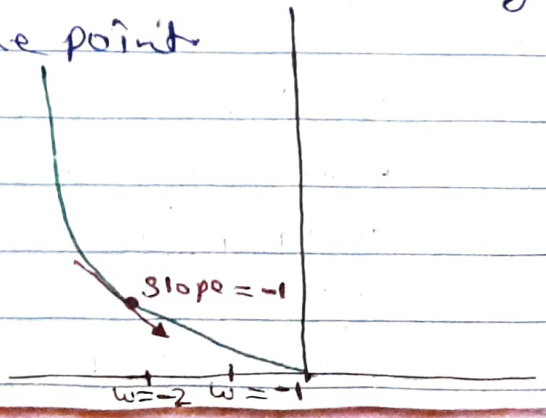
① Here we're gonna talk about how to update the weights of your logistic regression model to improve its error. So for all we know how to initialize the weights of logistic Regression Randomely.

② we also know that we can solve for the weights if we assume that the data is Gaussian distributed and has same covariance for each class, generally speaking this isn't the case always. So we want something that will work always for all kinds of data.

③ we took the squared error calculated its derivative with respect to the ~~way it is~~ weights set it to zero and solve for the weights. With the cross entropy error we can still take the derivatives but we can't solve for the weights when we set it to zero

④ So what can we do Instead :-

Let's say we have an error curve & we randomly initialized the weights to some point



Suppose the derivative at this point is −1 and suppose the point is at −2 while the idea is we want to take small steps in the direction

Step size == learning rate (1 for this example)
update becomes:-

$w = -2$

$w = -2 - 1*(-1) = -1$

so our update becomes −1 which ~~brig~~ brings us closer to the optimal point ($w=0$)
we just keep doing that until we get to the bottom
Derivative at the bottom = 0, so once we get there the weights won't be updated
This method works for any objective function not just for logistic regression

Note:- Calculus is at the very heart of deeplearning & a lot of machine learning & statistics

(3) How can we do gradient decent for logistic Regression specifically.

Objective function:- t_n = targets or Ytrain or Ytest
y_n = P(y=1 given X or pytrain pytest

$$J = -\sum_{n=1}^{N} t_n \log(y_n) + (1 - t_n)\log(1 - y_n)$$ → logistic regression train & test error function

we have to find the derivative of this with respect to each weight.
split into 3 derivatives

$$\frac{\partial J}{\partial w_i} = \sum_{n=1}^{N} \frac{\partial J}{\partial y_n} \frac{\partial y_n}{\partial a_n} \frac{\partial a_n}{\partial w_i}$$

→ Input of sigmoid OR sometimes we call it activation

$a_n = w^T \cdot x_n$

Solution ⇒

(a) $J = -\sum_{n=1}^{N} t_n \log(y_n) + (1-t_n)\log(1-y_n)$

The derivative of J with respect to y is just sum of N over y of n minus one minus the event over

$$\frac{\partial J}{\partial y_n} = -t_n \frac{1}{y_n} \cdot 1 + (1-t_n)\frac{1}{1-y_n}(-1)$$

(b) Next we need to find the derivative of sigmoid $y_n = \sigma(a_n) = \frac{1}{1+e^{-a_n}}$ → sigmoid function

use the division rule to differentiate y_n wrt a_n

$$\frac{\partial y_n}{\partial a_n} = \frac{\partial}{\partial a_n}\left(\frac{1}{1+e^{-a_n}}\right)$$

By division Rule :-

$$\frac{dy}{dx} = \frac{v\frac{du}{dx} - u\frac{dv}{dx}}{v^2}$$

OR

we can write $\frac{1}{1+e^{-a_n}}$ as $(1+e^{-a_n})^{-1}$

then simply differentiate it wrt a_n

$$-1(1+e^{-a_n})^{-2}\frac{d}{da_n}(1+e^{-a_n})$$

$$\frac{-1}{(1+e^{-a_n})^2}(0+e^{-a_n}\cdot -1)$$

$$\frac{\partial y_n}{\partial a_n} = \frac{-1}{(1+e^{-a_n})^2}(e^{-a_n})(-1)$$

$\frac{d}{da_n} 1 = 0$

$\frac{d}{da_n} e^{-a_n} = e^{-a_n} \cdot -1$

$$\frac{\partial y_n}{\partial a_n} = \frac{-1}{(1+e^{-a_n})^2}(e^{-a_n})(-1)$$

$$\frac{\partial y_n}{\partial a_n} = \frac{e^{-a_n}}{(1+e^{-a_n})^2} = \frac{1}{(1+e^{-a_n})} \cdot \frac{e^{-a_n}}{(1+e^{-a_n})}$$

$\frac{1}{(1+e^{-a_n})} \rightarrow y_n$

$\frac{e^{-a_n}}{(1+e^{-a_n})} \rightarrow (1-y_n)$

$$1 - \frac{1}{1+e^{-a_n}}$$

$$\frac{1+e^{-a_n}-1}{1+e^{-a_n}}$$

$$\boxed{(1-y_n) = \frac{e^{-a_n}}{1+e^{-a_n}}}$$

So we can say that

$$\boxed{\frac{\partial y_n}{\partial a_n} = y_n(1-y_n)}$$

(c) Take the derivative of a wrt w.

$$a_n = w^T \cdot x_n$$

$$a_n = w_0 x_{n0} + w_1 x_{n1} + w_2 x_{n2} + \cdots$$

$$\boxed{\frac{\partial a_n}{\partial w_i} = x_{ni}}$$

Finally we can put all this together

$$\frac{\partial J}{\partial w_i} = \sum_{n=1}^{N} \frac{\partial J}{\partial y_n} \frac{\partial y_n}{\partial a_n} \frac{\partial a_n}{\partial w_i}$$

$$= -\sum_{n=1}^{N} \frac{t_n}{y_n} y_n(1-y_n) x_{ni} - \frac{1-t_n}{1-y_n} y_n(1-y_n) x_{ni}$$

$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} \frac{t_n}{y_n} y_n(1-y_n)^2 x_{ni} - \frac{1-t_n}{1-y_n} y_n(1-y_n) x_{ni}$$

taking $1-y_n$ common

$$\frac{\partial J}{\partial w_i} = \frac{1-y_n}{1-y_n} - \sum_{n=1}^{N} \frac{t_n}{y_n} y_n(1-y_n)x_{ni} - (1-t_n) y_n x_{ni}$$

$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} t_n(1-y_n)x_{ni} - (1-t_n) y_n x_{ni}$$

taking x_{ni} common

$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} \left[t_n(1-y_n) - (1-t_n) y_n \right] x_{ni}$$

$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} \left[t_n - t_n y_n - y_n + t_n y_n \right] x_{ni}$$

$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} \left[t_n - y_n \right] x_{ni}$$

$$\boxed{\frac{\partial J}{\partial w_i} = \sum_{n=1}^{N} \left[y_n - t_n \right] x_{ni}}$$

y_n = Pobs given x
P(train or test)

t_n = targets (Y_{train} or Y_{test})

x_{ni} = $(X_{train})^T$ (Transpose)

Since each of i^{th} component on the left side only depends on the i^{th} component on the Right side we can write this in vector form.

$$\frac{\partial J}{\partial w} = \sum_{n=1}^{N} (y_n - t_n) x_n \quad \text{where } x \text{ is a Vector}$$

you can vectorize $\frac{\partial J}{\partial w} = \sum_{n=1}^{N} (y_n - t_n) x_n$ further

gradient descent formula for finding weights

$$\frac{\partial J}{\partial w} = X^T.(Y-T)$$ → weights finding formula *

$$a^T b = \sum_{n=1}^{N} a_n b_n$$

X^T = (Xtrain) Transpose
Y = pytrain Pofygivenx
T = targets ytrain

X = N X D (Xmatrix)

Y, T are N X 1 (matrices Y, T)

w = D X 1 (w matrix)

$X^T.(Y-T)$ shape is (D X N)(N X 1) → (D X 1)

which is the correct shape of w

N gets summed over

Explaination of above →

(a) A dot product is summission over an index

(b) we can achieve w as D X 1 matrix by writing $X^T.(Y-T)$ This is a D X [N] matrix times [N] X 1 matrix

Since N is the inner dimension so it goes away and you are left with the outer dimension

(c) what happens with the biased term. in logistic Regression it's easy to incorporate biased term into x itself just by adding a column of ones

(d) Let's say you want to find the derivative of the biased term explicitely for that let's return to the original expression

$$\frac{\partial J}{\partial w_0} = \sum_{n=1}^{N} (y_n - t_n) x_{n0} = \sum_{n=1}^{N} (y_n - t_n)$$ → bias term *

Here all the x_0's are 1

Summary of Section 3 :-

① How to find weights of a logistic Regression model given some training data

② we started by making an assumption that our data is gaussian distributed with equal covariances. We saw that this leads to linear classifier exactly like logistic regression (Use Bayes Rule

③ General solution :- Use gradient descent on cross-entropy (negative log likelihood) function and thought about how to maximize it or rather how to maximize the log likelyhood. There is no closed form solution to this problem. So we need a more general optimization method gradient descent

④ we saw that the hard part of gradient descent is finding the gradient. To find the gradient we have to use the Rules of calculus

⑤ We also used our of vectors & matrices to vectorize the solution. Since we know from our work in numpy that vectorize expression can be calculated faster than for loops

Practical issues surrounding logistic regression:-

① For any supervised ML model, we MUST know how to do these two things:
   (a) Prediction (Inputs → Outputs)
   (b) Training (Finding w)

② Regularization & overfitting (what happens when your model performs too well on the training data?)

③ Overfitting happens when your model performs too well on the training data but training data doesn't matter really on the real world

④ where machine learning model becomes powerful when they predict things for us in the future

⑤ It doesn't matter how good our model is at predicting stock returns over the past year we want to be able to predict stock returns tomorrow

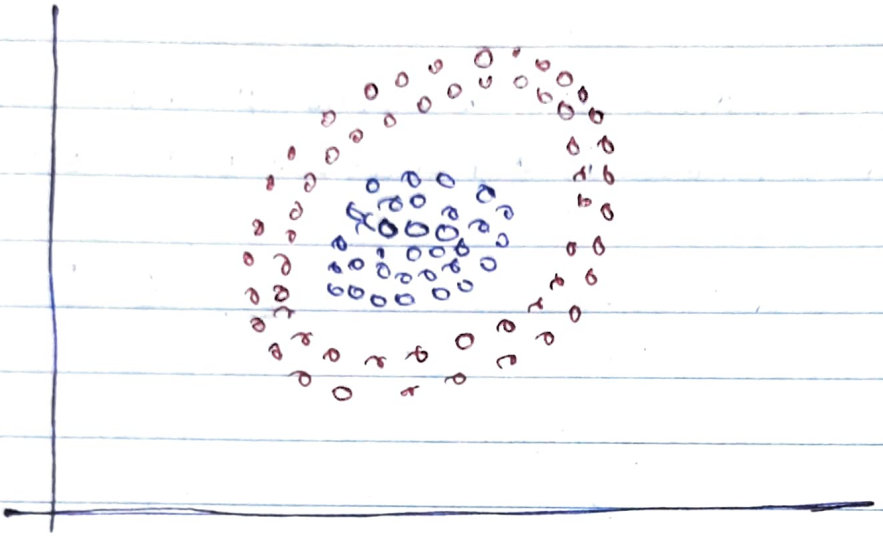
⑥ overfitting can happen for any number of reasons one of which is having irrelevant inputs in your model we will see how to bring the weights for irrelevant inputs down to zero

⑦ a related concept in logistic Regression is that there is a subtle problem with our cost function The way that our cost function is constructed what can happen is that the ideal weights are actually infinite. In this section we are gonna see how it happens & how we can fix it

Practical limitation of logistic Regression:-

① One practical limitation of logistic regression is that the model is a straight line or a hyper plane in other words it cannot curve. This, could be problammatic in many situations.

② For example:-



In the above data we can see that the two classes are clearly separable

If a new test point is given then a human can easily classify wheather the test point is Red or Blue

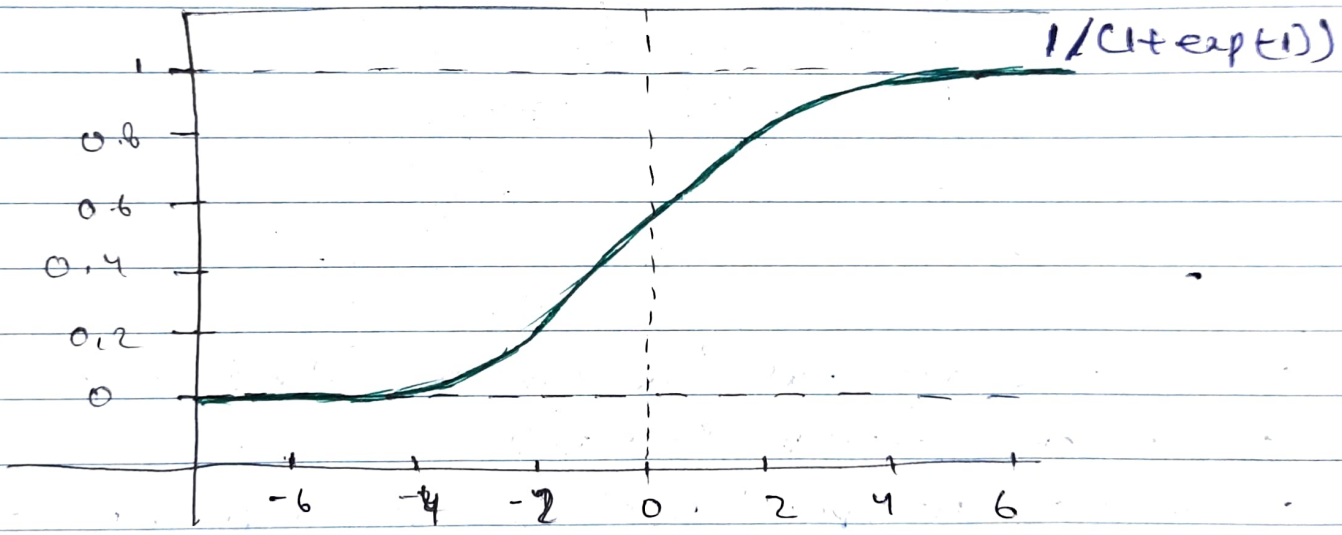
→ So why can't logistic Regression do this?

↳ It's because logistic Regression model is constrained to be straight

How to interpret the weights learnt by logistic Regression:-

① w_i is the amount of y will increase if x_i is increased by 1 and all other x's remain constant. i.e The weight w_i can be interpreted as the amount that y will increase if x_i increases by 1 and all other x's remain constant.

② In the binary scenario the output prediction from a logistic regression model is going to be either 0 or 1



③ So the weight is either going to either bring the output closer to 1 or 0.

④ A bigger weight means bigger effect. If weight w_i is big and positive than a small then a small increase in x_i assuming x_i is also positive will push the output closer to 1.

⑤ If w_i is big and negative than a small in increase in x_i assumming x_i is positive will push the output closer to zero

⑥ In other words larger magnitude means larger impacts on the output.

⑦ If we see the interpretation of the output of the logistic model

$$p(y=1|x) = \sigma(w^T.x) = \frac{1}{1+\exp(-w^T.x)}$$

Therefore if we subtract 1 we get

$$p(y=0|x) = 1-\sigma(w^T.x) = 1 - \frac{1}{1+\exp(-w^T.x)}$$

$$= \frac{1+\exp(-w^T.x) - 1}{1+\exp(-w^T.x)} = \frac{\exp(-w^T.x)}{1+\exp(-w^T.x)}$$

⑧ The next step is to calculate the odds.

$$\text{Odds} = \frac{P(\text{Event})}{P(\sim\text{Event})}$$ Probability that event happens / Probability that event doesn't happen

$$\text{In our case} = \frac{p(y=1|x)}{p(y=0|x)} = \text{odds}$$

⑨ If we write this down in terms of w & x we get exponential of $w^T.x$

$$\frac{p(y=1|x)}{p(y=0|x)} = \frac{\frac{1}{1+\exp(-w^T.x)}}{\frac{\exp(-w^T.x)}{1+\exp(-w^T.x)}}$$

$$= \frac{1}{1+\exp(-w^T.x)} \times \frac{1+\exp(-w^T.x)}{\exp(-w^T.x)} = \frac{1}{\exp(-w^T.x)}$$

$$= \exp(w^T.x)$$

$$\boxed{\frac{p(y=1|x)}{p(y=0|x)} = \exp(w^T.x)}$$

(10) If we take log of both sides :-

$$\log\left(\frac{P(y=1|x)}{P(y=0|x)}\right) = \underbrace{w^T.x}_{\text{It's exactly same as linear Regression}} = w_1x_1 + \cdots + w_Dx_D$$

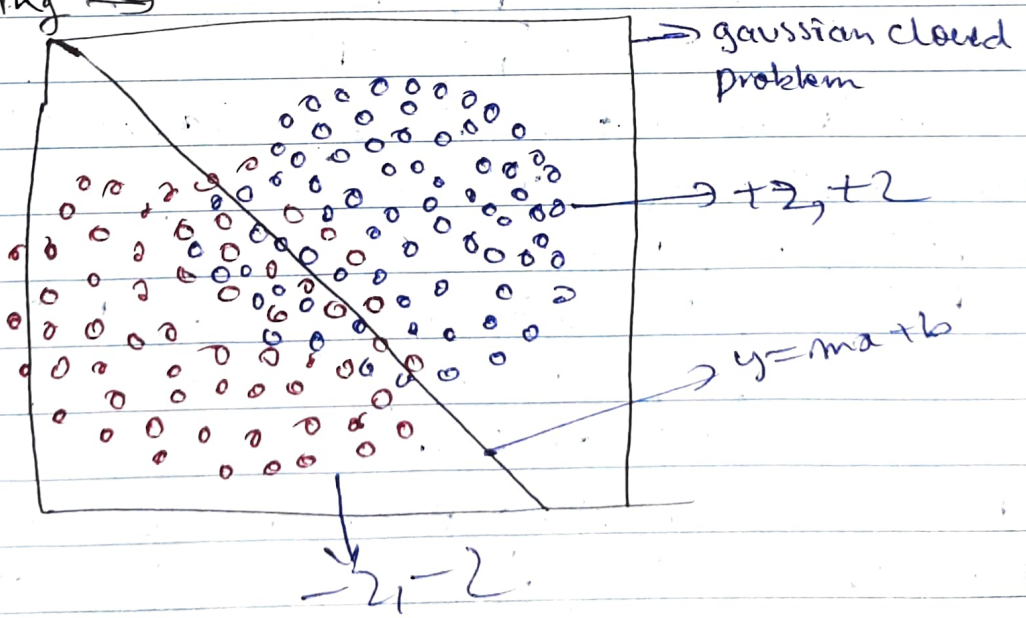
Since on the left we took the log of the odds. This quantity is appropriately known as log odds. Therefore we are doing linear regression on the log odds

(11) This means we can interpret the weights like in linear regression.

The weight w_i can be interpreted as the amount that a log odds will increase. If x_i increases by 1 and all other x's remain constant this interpretation is something that shows.

Regularization :- It is a generalization method

(A) Overfitting →



① Again we are gonna look at Gaussian cloud problem with 1 gaussian centered at +2,+2 & one gaussian centered at -2,-2. This the same problem that we found out that exact Baysine solution & it's easy to visualize because it's just two circles & you draw a line to separate them

② The Baysian solution that we found was $w=(0,4,4)$ where 0 is the bias

③ So what this line corresponds to = $y=mx+b$ is the representation of line

④ $\boxed{y} = mx+b$

y coordinate of the xy plane & not y as the output of logistic Regression

⑤ So we can say $0+4x+4y=0$
This gives us $y=-m$
so that means we get a slope of -1 & y intercept=0

⑥ Why is the Baysian solution (4,4)? why not 1,1 or 10,10? Infact these all represent the same line. This is why we need Regularization here

⑦ Consider the objective/error function so now this y is the output of logistic Regression.

→ Let's take a test point $(x_1,x_2)=(1,1)$

→ we know this should be in the same class as all the points surrounding is 2,2 so the class should be 1

→ So if we plug this into logistic Regression model

with the current setting of w

$$\sigma(0+4+4) = \sigma(8) = 0.99966$$

→ So with sigmoid of eight we get this objective which is approximately .0003

$y = \sigma(8)$, $J = -0.0003$ (good)

y: output of logistic Regression; σ: Sigmoid; J: objective

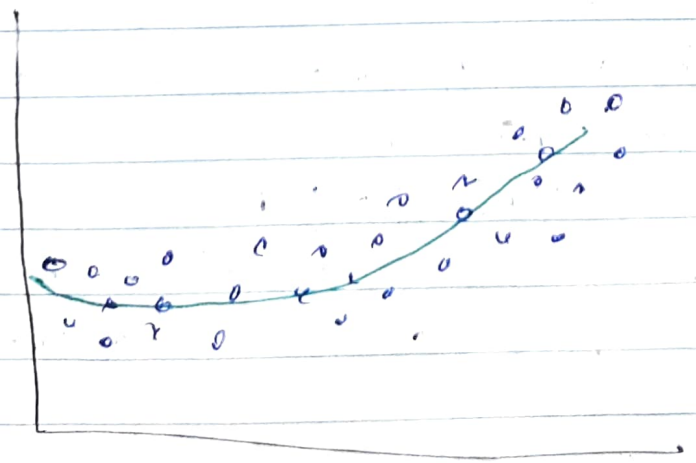
→ Now what if my weights were (0,1,1) 0 is the bias
J would become = −0.1269 (Not good)

→ what if my weights were (0,10,10) 0 is the bias
J would become = −2.06115e−09 OR
$= 2.0 \times 10^{-9}$ (Not good)

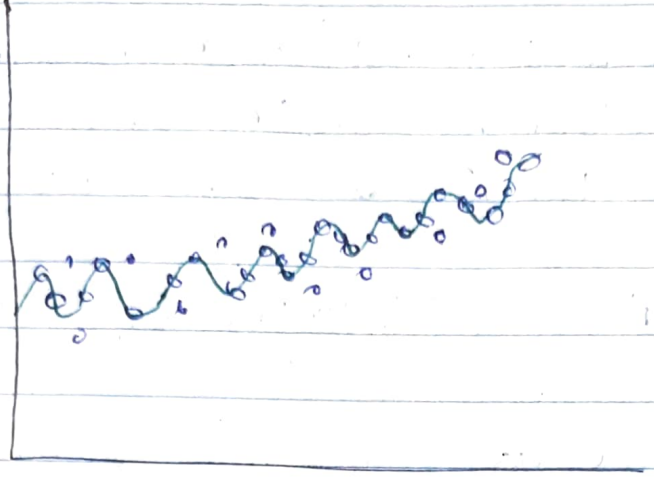
→ Moral of this is that the best weight of this model is (0,∞,∞) But of course computers can't represent ∞. So eventually we will just get an error

③ Regularization:-

① a lot of the times people explain regularisation in terms of regression which I've already done



Good fit



overfitting

(2) In fact your data completely fills up all possible inputs yourshouldn't overfit eve if your model is very complex
(3) That is why we like having lots of data your model overfits when it has to guess what the output should be in a space ~~that~~ that it is never seen before
(4) But if your training data is nicely spreadout and covers all the different possibilities. you can train to do well on that data
(5) If your test data looks exactly like a training data and you do well on your training data then you'll also do well on your test data
(6) Even though our data is nicely shaped and represents all the possible inputs the logistic Regression model still tries to give us a solution that the computer can't calculate the solution to $w=(\infty,\infty,\infty)$. The solution to this is Regularization
(7) Regularization penalizes very large weight.
(8) So we have a cost function which is our cross entropy

$$J = -[t\log(y) + (1-t)\log(1-y)]$$

we just add another term to this cost so that it makes our cost grow larger If any weights grow bigger.
So the ~~Th~~ thing we add is the $\frac{\lambda}{2}$ times the L2 norm of the weights

$$J_{reg} = J + \left(\frac{\lambda}{2}\right) \|W\|^2 = J + \left(\frac{\lambda}{2}\right) w^T . w$$

what it does it encourages the weights to be close to zero. So now you will not get a weight like (zero, 10, 10) because that will make your cost very large.

λ is what we call a smoothing parameter
λ balances the cross entropy cost & regularization penalty.
If λ is larger the weights will go closer to zero
If λ is smaller the weights will try to minimize the cross entropy
Typical values of $\lambda = 0.1$ or 1 or something in between

How to choose a ~~cross~~ λ value.
observe the behaviour of the cost function and look at the results
There is no universal way to choose the best LAMBDA

How does λ effect our solution for W:-
(1) To perform gradient descent all we need to do is take the gradient of the objective function & move in that direction since addition doesn't effect the gradient calculation

dJ_{reg}/dw instead of dJ/dw

Add doesn't effect other terms in derivative

reg. cost = $\left(\frac{\lambda}{2}\right)(w_0^2 + w_1^2 + w_2^2 + \cdots)$

You just take the gradient separately. So we just need to focus on the gradient of a regularization penalty.

$\frac{d(\text{reg-cost})}{dw_i} = \lambda w_i$ → The derivative wrt any particular w_i is just Lambda times w_i

So now we can add this full vectorized form to our original gradient

In vector form = $\frac{d(\text{reg-cost})}{dw} = \lambda w$

$dJ_{reg}/dw = X^T(\hat{Y} - T) + \boxed{\lambda w}$ → In program we set $\lambda = 0.1$

learning rate * hpidot

↳ Regularization component.

Regularization in Probabilistic prospective:-

① We know that with the cross entropy what we are really trying to do is maximizing the likelihood.

~~$J = t[t \log y + (1-t)\log(1-y)] - (\lambda/2)\|w\|^2$~~

$$J = t\left[t \log y + (1-t)\log(1-y)\right] - \left(\frac{\lambda}{2}\right)\|w\|^2$$

$\exp(J) = y^t(1-y)^{(1-t)} \exp(-\lambda\|w\|^2/2)$

↳ when I exponentiate a -ve squared term

↳ L2 norm of w

Binomial distribution of the likelihood

we call this gaussian distribution over the weights prior i.e It represents our prior beliefs about the weights

← That's just a gaussian distribution

In particular that they should be small & centered around zero

The variance of this gaussian distribution is $\frac{1}{\lambda}$

$w \sim N(0, \frac{1}{\lambda}) \rightarrow$ (prior)

$t \sim$ Bernoulli$(y) \rightarrow$ (likelihood)

posterior $\propto$ likelihood $\times$ prior.

λ is also called the precision which is the inverse variance $\frac{1}{\lambda}$

This will keep occuring in Baysian machine learning

The basic rule is posterior $\propto$ likelihood $\times$ prior

Bayse Rule is-

$$P(W|X, y) \propto P(X, y|W) P(W)$$

Consequences of: $P(B|A) = \frac{P(A|B) P(B)}{P(A)} = \frac{P(A,B)}{P(A)}$

Bottom part is just: $\sum_B P(A,B) = \sum_B P(A|B) P(B)$

sum over B

with regularization we maximize the posterior

without regularization we maximize the likelihood

This is called maximum a posteriori OR MAP estimation.

L1 Regularization:

① In L2 Regularization even when we add a column of completely random noise to our data we can still improve our training score

② In general we want the dimensionality of x to be much smaller than the number of samples N

$D \ll N$	$D > N$
(ideal)	Not ideal

③ In certain data sets the dimensionality is much greater than the number of samples. One way to visualize this via the X matrix. we want this X matrix to be skinny i.e. N (no of samples) to be large & D (features per sample) to be small. If we have opposite situation where X is fat where N is small and these large that's not a good situation & we need to take steps to avoid potential problems.

④ what we would like to be able to do is select the small number of important features that produce that ~~thread~~ trend & remove all the features that are just noise. we call it sparsity since the end result will be that most of the weights will be zero and only a few of the weights will be non-zero

⑤ L1 Regularization which helps us achieve sparcity

⑥ L1 Regularization has a pretty simple setup similar to L2 we just add a penalty term to the original cost

(7) As you might assume since we use a multiple of the L2 norm of the weights in L2 regularization

$$L_{RIDGE} = \sum_{n=1}^{N} (t_n \log y_n + (1-t_n) \log(1-y_n)) + \lambda \|w\|_2^2$$

$$L_{LASSO} = \sum_{n=1}^{N} (t_n \log y_n + (1-t_n) \log(1-y_n)) + \lambda \|w\|_1,$$

we will similarly use L1 norm of the weights in L1 Regularization.

→ L2 Regularization is called Ridge Regression

→ L1 Regularization is called Lasso Regression

(8) Let's look at the probability distribution involved

$$p(w) = \frac{\lambda}{2} \exp(-\lambda |w|),$$

↑ gaussian distribution

(9) So for L2 regularization we had Gaussian likelihood and a Gaussian prior on w in this case we no longer have a gaussian prior on w

Gaussian likelihood ≠ Bernaulli likelihood

(10) So what distribution has a negative absolute value in the exponent. This would be the laplace distribution.

(11) so with L1 regularization we are putting a laplace distributed prior on the weights and we are solving for the posterior of w with bel applies

(12) ★ $J = -\sum_{n=1}^{N} t_n \log(y_n) + (1-t_n) \log(1-y_n) + \lambda |w|$ (I^W)

↑ cost function

gradient descent equation

finding weights with L1 Regularisation

$$\frac{\partial J}{\partial w} = X^T(Y-T) + \lambda \, \text{sign}(w), \quad (\text{II}^{\text{nd}})$$

L1 Regularization constant OR L1 Penalty

For logistic regression we just calculate the gradient & move in the direction of O.

For L1 regularisation you already know to take the gradient of the first part. The 2nd part is just Lamda times the sign of w.

The sign function returns 1 if its argument is greater than zero negative 1 if its argument is less than zero & zero if its argument is = 0

L1 vs L2 Regularization :-

① L1 Regularization encourages features weights to be sparse ie only a few will be non-zero to regularisation

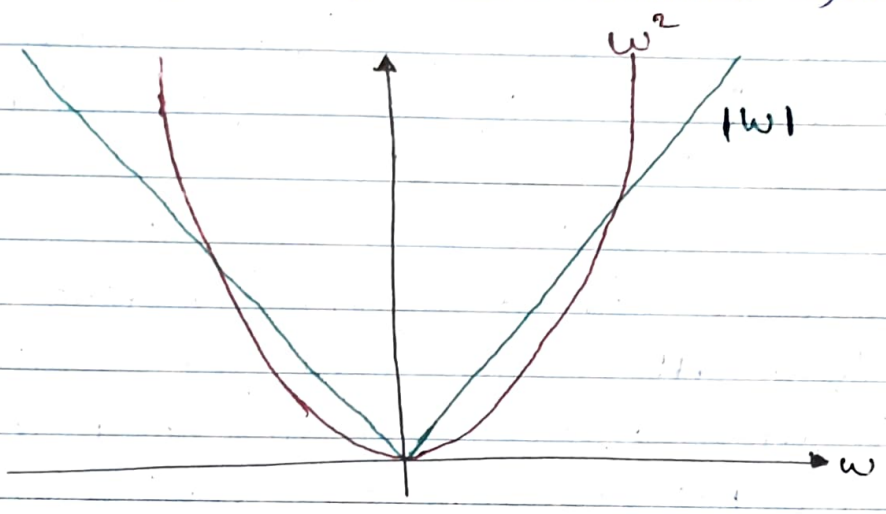
② L2 Regularization encourages feature weights to be small but will not encourage weights to go exactly zero

③ Both L1 & L2 Help improve generalization & testor because they prevent you from overfitting to the noise in the data L-1 does this by choosing the most most important features the ones that have the most impact with respect to the output you can imagine that the features with a smaller impact might in fact just help you predict that the noise in the training set.

L-2 ~~does~~ prevents you from overfitting by making the assertion that none of the weights are disproportionately large

(4) Think of a one dimentional weight for L2 Regularization I get the -logprior -ve log prior of w to be quadratic. For L1 regularization we get the negative log prior of w to be the absolute function

(5) The important thing to consider here is the derivative of this functions because gradient descent goes in the direction of the derivative, for a quadratic the closer you get to zero the smaller your derivative gets



Sine it also approaches to zero.

(6) So for L2 Regularization where your weight is already small. The further gradient descent won't change it that much ($w \to 0$, derivative $\to 0$)

(7) For the absolute function the gradient is constant it always has a magnitude of 1 (+/-1). (at $w = 0$) The derivative at 0 is undefined but we treat it as 0. So for L1 regularization gradient descent will go towards zero at a constant rate. Then when it arrives at 0 it stays there forever. So that is why L2 Regularization encourages weights to be small while L1 Regularization encourages

weights to be zero therefore encouraging Sparsety.

Note:- It is possible to include both L_1 & L_2 Regularization in your model. This also has a special name called elastic net. Here we will add L_1 & L_2 penalty to the cost function.

→ $J_{RIDGE} = J + \lambda_2 |w|^2$

$J_{LASSO} = J + \lambda_1 |w|$

$J_{ELASTICNET} = J + \lambda_1 |w| + \lambda_2 |w|^2$

Why we devided $w = \frac{np.random.randn(D)}{np.sqrt(D)}$

$w = \frac{\text{random } D}{\sqrt{D}}$ where D is number of features per sample OR Dimensions

① We did $w = \frac{np.random.randn(D)}{np.sqrt(D)}$ in L_1 regularization program.

② poor weight initialization can lead to poor convergence of your loss per iteration

③ Sometimes your loss might even explode to ∞. If you've never encountered this then ~~dirtedly~~ it is likely you just haven't practiced enough

④ Once you've seen enough datasets and done enough machine learning work you will encounter a scenario where weight initialization matters. That is not to say it ~~always~~ always matters and that's not to say there won't be some instances where these recommendation will appear to perform poorly

(5) How it works!-

weight initialization

W = np.random.randn(D) / np.sqrt(D),

→ we know that the randn function draws a samples from the standard normal and that's usually denoted as $N_{0,1}$

→ what happens when we multiply these samples by some number of C. Well you can show that if you multiply random samples from the standard normal by a number C they will ~~are~~ have mean zero and standard deviation C

randn() * C == samples from $N(0, C^2)$

→ In other words it's equivalent to sampling from the distribution ~~and a ger~~ nzero C^2

→ Infact this is just the opposite of standardization or normalization

→ when we want to standardize a set of samples we subtract their mean and divide by their standard deviation sigma

To standardize ⇒ $z = (x - \mu)/\sigma$

z has a mean 0, standard deviation 1

~~To~~ inverse transform ⇒ $x = z\sigma + \mu$.

x has mean μ, standard deviation σ

this makes it so that the standard samples have a mean zero and standard deviation 1.

→ Thus we can perform the inverse operation as well If we multiply it by a ~~num~~ number sigma and add ~~a number you on a μ on a standard dev~~

a number μ on a standardized sample the result will be a sample that has mean μ and standard deviation sigma
Inverse transform: $X = z\sigma + \mu$ // X has mean μ, sd σ

⑥ Let's go back to our initialization of weights

$w =$ np.random.randn(D) / np.sqrt(D)

→ If we divide by the squareroot of D, This is equivalent to multiplying by 1 over squareroot of D

$$\frac{1}{\sqrt{D}}$$

→ Therefore we are saying that we want the initial weight vector W to have mean zero and variance one over D.

→ So why do we want to be that

If you don't do this →

$w =$ np.random.randn(D) / np.sqrt(D)

then our loss per iteration might explode and go to infinity.

→ The reason it's exploding is because those weights are too large and if they are too large that means their variance is too large. Which means we may be able to avoid this problem if we make the variance smaller

→ In fact in some code there are people who will just multiply by a constant like 0.01. This is also perfectly ok. It comes down to doing experiment and finding the optimal value for your own dataset This is always the ultimate foolproof method.

P Q R S T U V W X Y Z

(7) Okay so we know that we want weights with small variance but why exactly $\frac{1}{D}$ variance

→ Let's go back to multiple linear regression where we have D input features and 1 output, since we have D input features our weight vector will be a vector of size D. ~~As you know~~

→ As you know we like to normalize our data before passing it through any kind of model whether it's a linear regression or a neural network. Again it's all about experimentation

~~$y = w_1x_1 + w_2x_2 + \cdots + w_Dx_D$~~

$y = w_1x_1 + w_2x_2 \text{ ---- } + w_Dx_D$

→ If we normalize ~~every~~ our data then that means each input feature will have means zero & variance 1

→ Suppose we want to know the variance of the output Y. Why do we care about the variance of Y

→ well in neural networks, After the first layer you simply have more layers of the same type. Therefore the output of the first layer will be the input of the second layer and so on. But since those will be the inputs to the next layer you want those to be normalized as well.

→ If it's just linear regression your target will probably be standardized so there's still a ~~reason~~ reason to care about the variance of Y

→ we know that the expression of Y =>

$Y = w_1x_1 + w_2x_2 + \cdots + w_Dx_D$

So how do we calculate the variance of Y

→ Since all the Ws & Xs are independent this expression simplifies to

$Var(y) = Var(W_1)Var(X_1) + Var(W_2)Var(X_2) + \dots + Var(W_D)Var(X_D)$

($Var(W_1)$: variance of W_1; $Var(X_1)$: variance of X_1)

$= Var(W_1) + Var(W_2) + \dots + Var(W_D)$ // since $Var(X_i) = 1$

$= D\,Var(W)$

→ Variance of all the Xs = 1 ["VarXi = 1"] because we have standardized our data & we also know that variance of all the Ws will be the same

at last we get that variance of $Y = D * Variance(W)$ ($Var(W)$)

→ It means if we want our variance of Y to be 1 which we do then the variance of W must be $\frac{1}{D}$ or standard deviation of $W = \frac{1}{\sqrt{D}}$

NOTE:-

- ✱ → This isn't the only way to initialize the weights
- ✱ → You want the variance of the weights to be initialized to small values
- ✱ → Other method include He-normal, Glorot-normal etc. to initialize the weights

  Typically these are used in Deep neural networks & not linear models like linear Regression
- ✱ → It's normal to just multiply by a constant value like 0.01

Q R S T U V W X Y Z

Sentiment analyzer (theory):-

① What is sentiment analyzer?

↳ Sentiment = how positive or negative some text is

↳ amazon reviews, Yelp reviews, hotel reviews etc...

② Our data is in form of xml file so you will need an xml parser

③ Outline of the sentiment analyzer:-

(a) We are gonna use the reviews on the electronics section but this code should work on others as well

(b) xml parser used here is Beautiful Soup

(c) only look at the key "review_text"

(d) we will need 2 passes, one to determine vocabulary size and which index corresponds to which word, and one to create data vectors

(e) after that we could just use SKlearn classifier as we did previously

(f) But we will use logistic regression so we can interpret the weights.