← this is a cost function!

it tells us whether or not a choice of parameters classifies (x_p, y_p) correctly or not.

- One can alternatively take — much akin to linear regression — a Least Squares cost function approach here. → but these lead to very non-convex costs (although they are used sometimes).

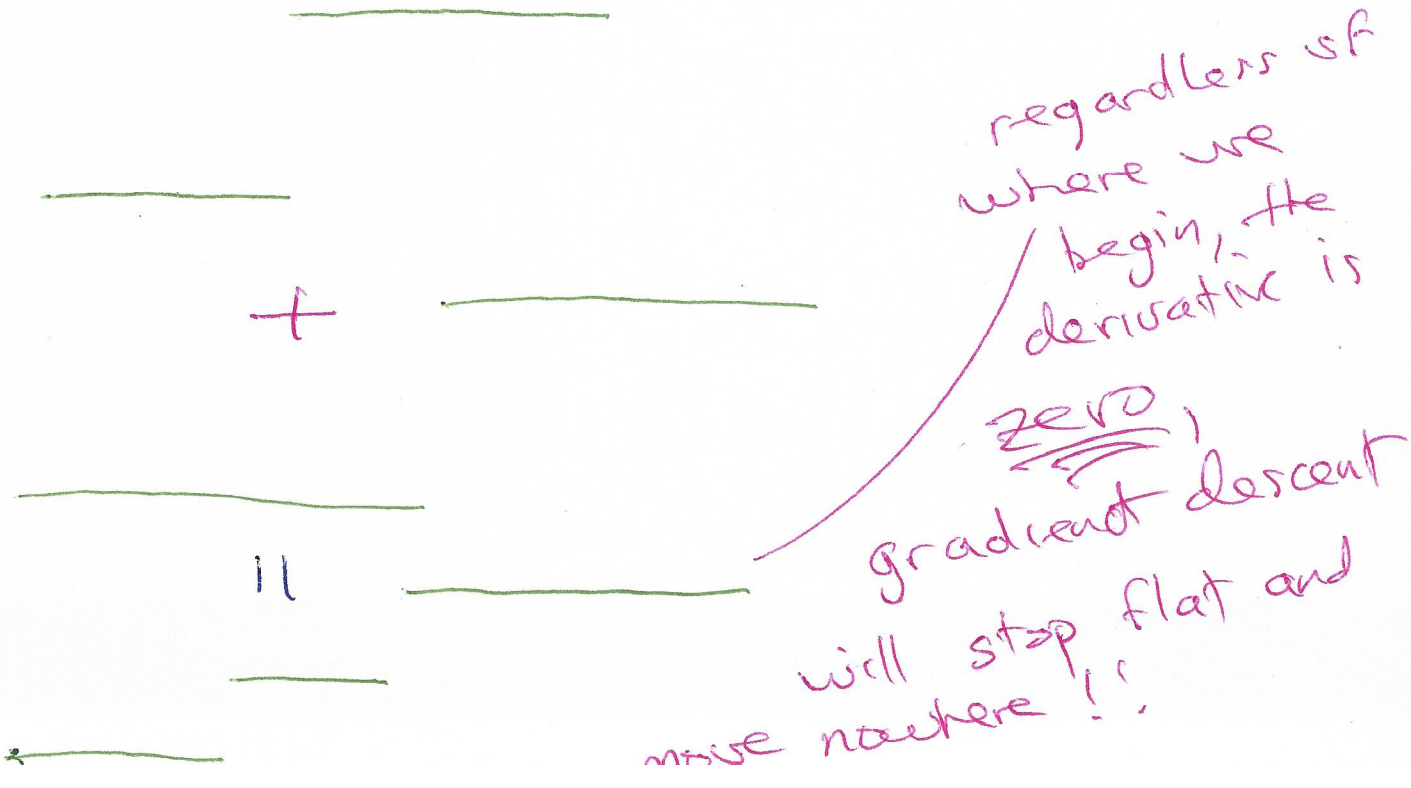
- the "counting cost" we have here is more natural — we want a model that classifies the most points correctly

- before summing these up and call-ing it a day, notice right away we can tell that there will be a few major problems if we try to minimize

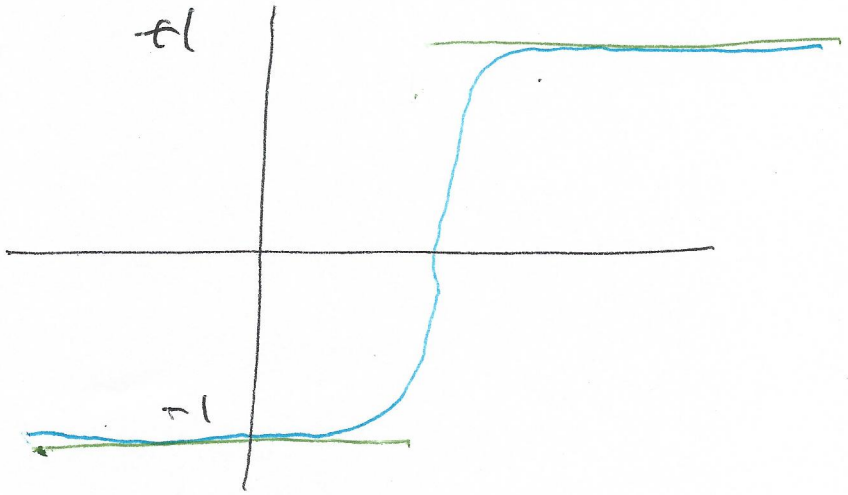
$$-\sum_{p=1}^{P} \text{sign}\left(y_p(w_0 + w_1 x_p)\right)$$

each of these is nonconvex, discontinuous, and has zero derivative almost everywhere

e.g.



- Some of these issues can be dealt with by using a smooth approximation to the sign function: the logistic sigmoid tanh (hyperbolic tangent)



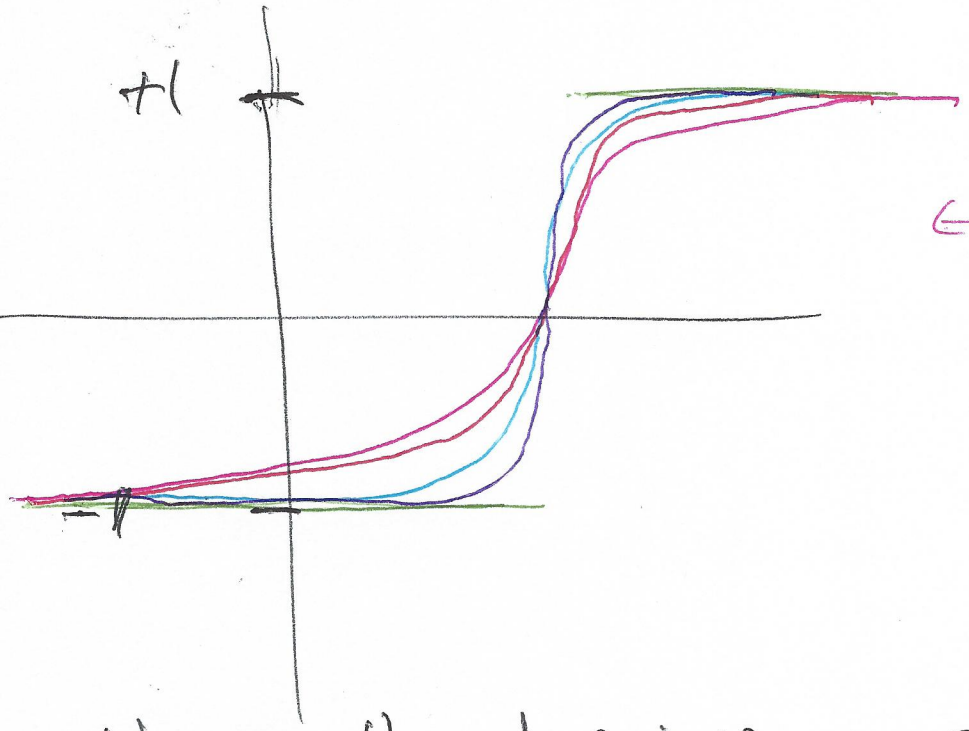
$$\tanh(\alpha) = 2\frac{1}{1+e^{-\alpha}} - 1$$

- Swap out sign with tanh, so shove our line through tanh

$$\tanh\left(y_p(w_0 + w_1 x_p)\right) \approx \begin{cases} +1 & \text{if correct} \\ -1 & \text{if incorrect} \end{cases}$$

this is now an $\approx$ since tanh is an approximation to sign

- Note, by tuning w_0, w_1 we can also make tanh infinitely close to step



← we can use tanh to approximate sign as closely as desired

Using the definition of tanh our counting cost is equivalently

$$\frac{1}{1+e^{-y_p(w_0+w_1x_p)}} \approx \begin{cases} +1 & \text{if correct} \\ 0 & \text{if incorrect} \end{cases}$$

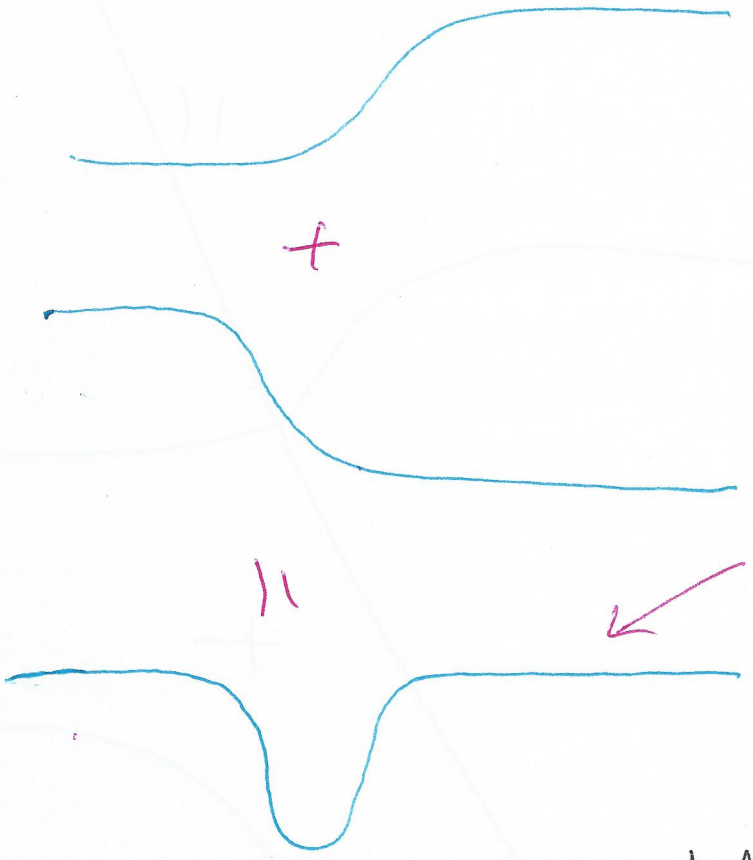
the negative of

- We can try to sum these up and minimize

minus sign

$$\sum_{p=1}^{P} \frac{1}{1+e^{-y_p(w_0+w_1x_p)}}$$

- Still pretty nonconvex since each term is nonconvex



totally valid because we're regressing with logistic function

- Nonetheless, this is a logistic regression cost function

- But can we do better? A convex cost maybe?

Lets look at one of the terms

$$\frac{1}{1+e^{-y_p(w_0+w_1 x_p)}} \approx \begin{cases} 1 & \text{if correct} \\ 0 & \text{else} \end{cases}$$

Ok, wouldn't you agree that the only thing determining the value of the left hand side above is $e^{-y_p(w_0+w_1 x_p)}$

- if it is small the point is correctly classified. If its large, its not

- So how about minimizing the cost function

$$\sum_{p=1}^{P} e^{-y_p(w_0+w_1 x_p)}$$

- well, each term is convex, and looks like ∪ or ∩

So too is its sum!

So a perfectly legit convex logistic regression cost is

$$g(w_0, w_1) = \sum_{p=1}^{P} e^{-y_p(w_0 + w_1 x_p)}$$

often the cost function for boosting

- one last common adjustment is made to each summand to make the cost less sensitive to outliers

$$e^{-y_p(w_0 + w_1 x_p)} \longrightarrow \log\left(1 + e^{-y_p(w_0 + w_1 x_p)}\right)$$

- See exercise 4.11 of book to see why.

- So our final convex cost for logistic regression

$$g(w_0, w_1) = \sum_{p=1}^{P} \log\left(1 + e^{-y_p(w_0 + w_1 x_p)}\right)$$

We'll use this one - its very common - but the moral here is that you have several available choices

- Minimize by gradient descent
- the derivatives

$$\frac{\partial}{\partial w_0} g(\vec{w}) = \frac{-1}{1+e^{-y_p(w_0+w_1 x_p)}} e^{-y_p(w_0+w_1 x_p)} y_p$$

$$\frac{\partial}{\partial w_1} g(\vec{w}) = \frac{-1}{1+e^{-y_p(w_0+w_1 x_p)}} e^{-y_p(w_0+w_1 x_p)} x_p y_p$$

- Moral: For least squares linear regression the cost fell quickly out, but for logistic regression although the idea (minimize # misclassifications cost) is clear in the beginning, it takes some tinkering to get a good convex cost. Sometimes you gotta work for it.

- It will not be the case that every machine learning problem can be modelled in such a way that its convex

  e.g. neural networks

- And - furthermore - "non-convex" isn't necessarily bad. It's just that - if we can avoid it - we should.
- Higher dimension than 2? No problem everything generalizes nicely, all the formulae are completely analagous
- for input $(\vec{x}_1, y_1)\ (\vec{x}_2, y_2) \cdots (\vec{x}_P, y_P)$

  input
  each point has N dimension

$$\vec{x}_p = \begin{bmatrix} x_{p,1} \\ x_{p,2} \\ \vdots \\ x_{p,N} \end{bmatrix}$$

Our cost function is

$$g(\vec{w}) = \sum_{p=1}^{P} \log\left(1 + e^{-y_p(w_0 + w_1 x_{p,1} + w_2 x_{p,2} + \dots + w_N x_{p,N})}\right)$$

And derivatives

$$\frac{\partial}{\partial w_0} g(\vec{w}) = \frac{-1}{1+e^{-y_p(w_0+w_1 x_{p,1}+\dots+w_N x_{pN})}} e^{-y_p(w_0+w_1 x_{p,1}+\dots+w_N x_{pN})} y_p$$

and for $n = 1 \dots N$

$$\frac{\partial}{\partial w_n} g(\vec{w}) = \frac{-1}{1+e^{-y_p(w_0+w_1 x_{p,1}+\dots+w_N x_{pN})}} e^{-y_p(w_0+w_1 x_{p,1}+\dots+w_N x_{pN})} x_{p,n} y_p$$

- Can be written much more compactly, and in other ways

Support Vector Machines:

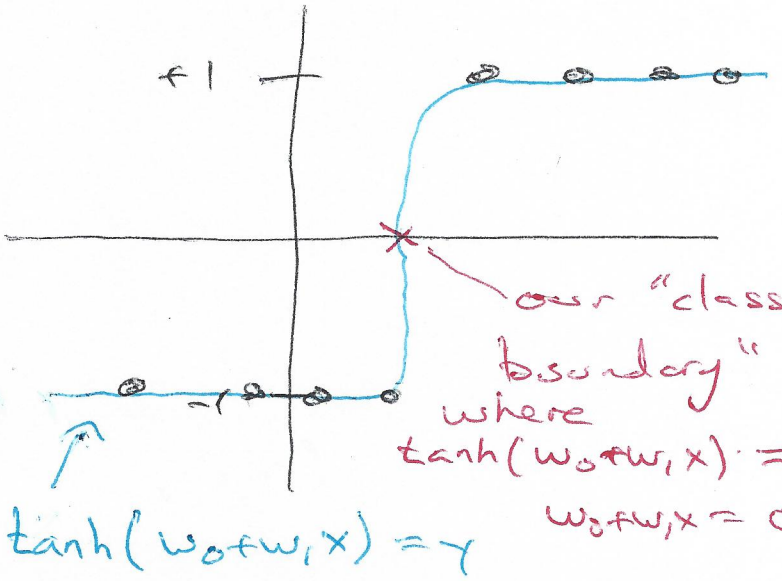
- We won't have time to cover this in detail, but you're in luck: its theoretically only slightly different than logistic regression, and practically

its not different at all.

- You can see Chapter 4 of the book to learn why they are so similar in detail, and see chapter 7 if you're interested in seeing how to "kernelize" both

- in practice never really clear which will work better, so you can use both and take best result, or stick with either one (never the case that one is much better than the other).

How to view linear classification from above & where does the boundary come from?

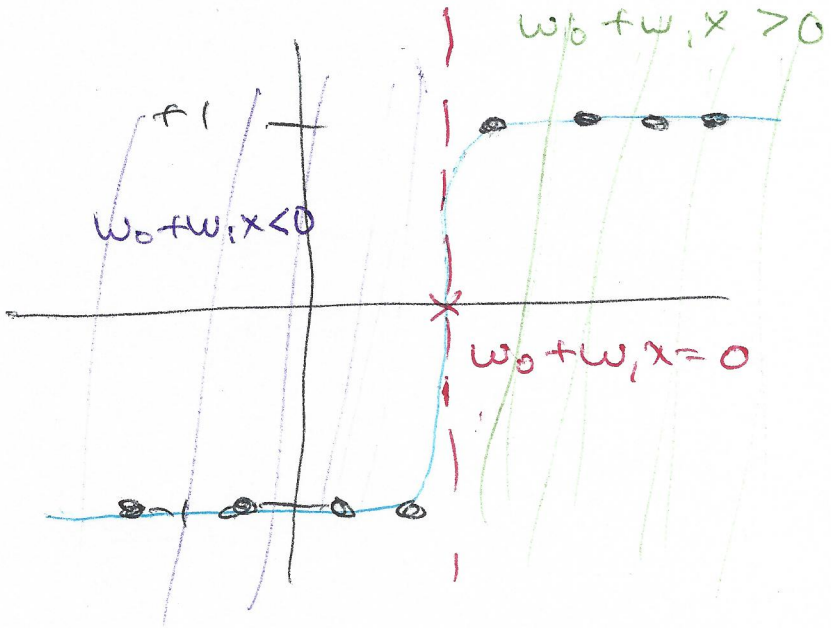


- Say we've successfully fitted a logistic regression. Where do we get the classification boundary from? Like we saw when we ~~solved~~ used logistic regression in Level 1 ~~for~~ 3-d data?

- In other words, how do we classify data after fitting? As shown in the picture ~~the~~ point $(x,0)$ where

$$\tanh(w_0+w_1x) = 0$$

gives us the cutoff. The classification boundary



- notice, the same point is

$w_0 + w_1 x = 0$

due to tanh only being = 0 when it's input = 0

- every input x where

$w_0 + w_1 x > 0$ is predicted of class $+1$

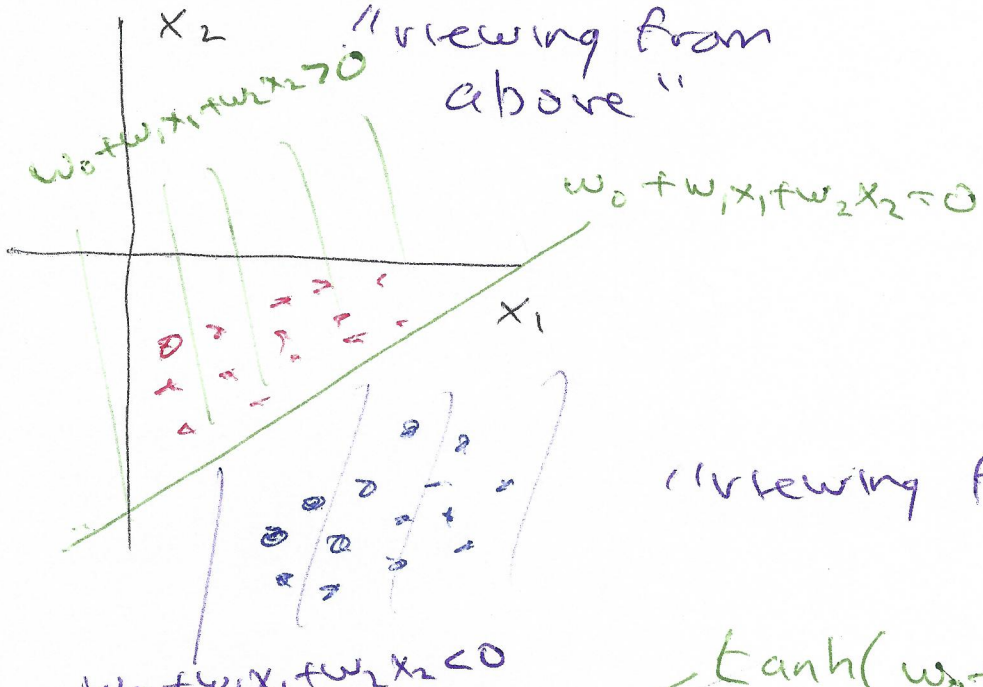
$w_0 + w_1 x < 0$ is predicted of class -1

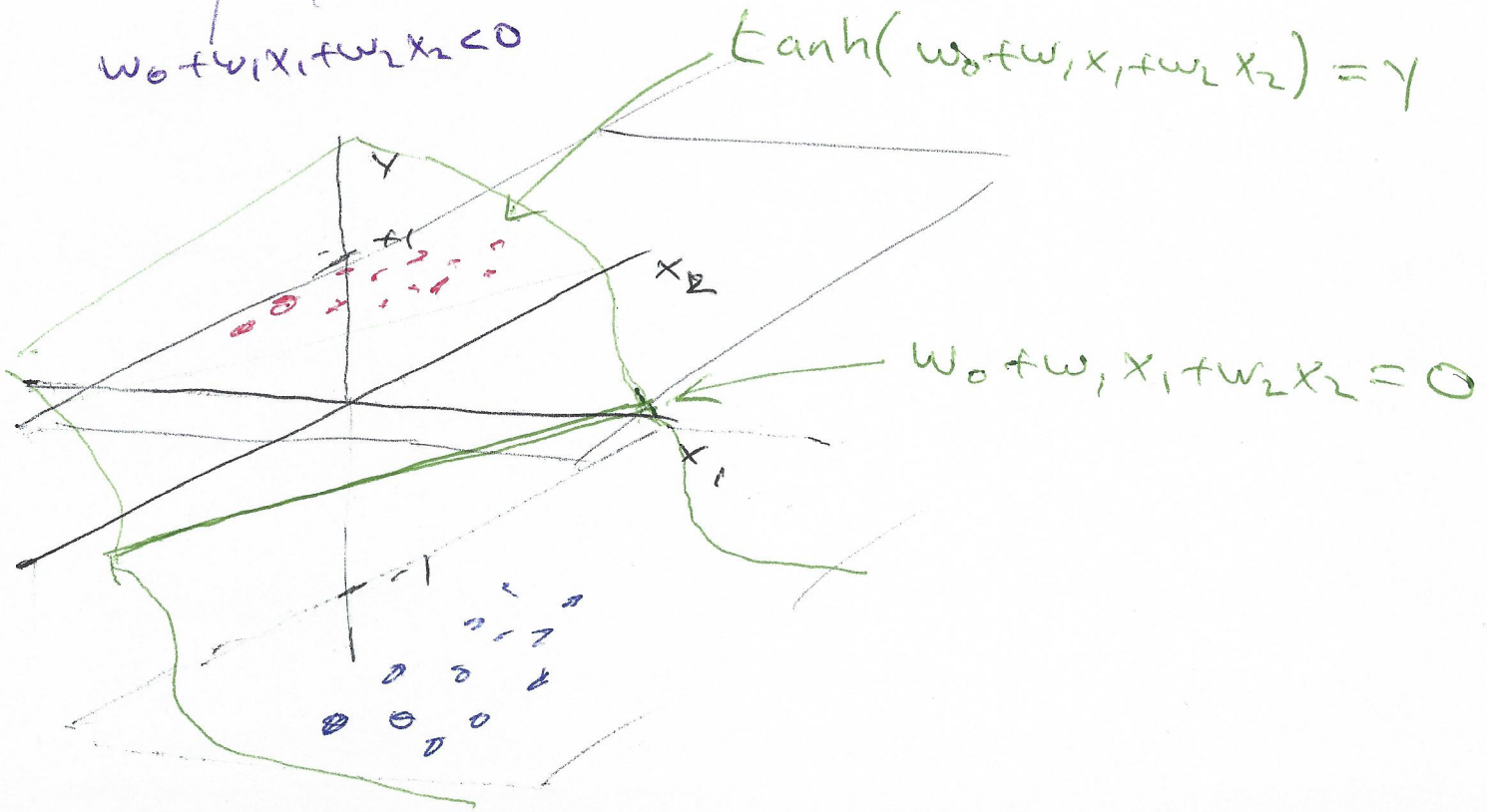
- in short, to make prediction we use sign! i.e.

$$\text{sign}(w_0 + w_1 x) = \text{label (output) of point } x$$

- If we go up a level / dimension we have a separating line or plane

- e.g. in 3-d





How do we then calculate how well we have classified?

Use the learned boundary in higher dim too e.g. in 3-d, $(x_1 x_2)$

$w_0 + w_1 x_1 + w_2 x_2 > 0 \rightarrow$ predicted class +1

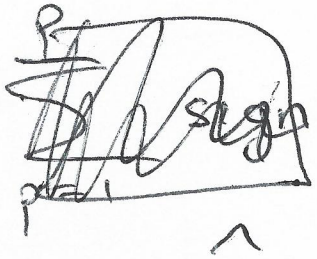
$w_0 + w_1 x_1 + w_2 x_2 < 0 \rightarrow$ predicted class −1

i.e. $\text{sign}(w_0 + w_1 x_1 + w_2 x_2)$ gives predicted label

---

So count misclassifications of dataset

$(\vec{x}_1, y_1)\ (\vec{x}_2, y_2) \ldots (\vec{x}_p, y_p)$



– calculate predicted labels

$\hat{y}_p = \text{sign}(w_0 + w_1 x_{p,1} + w_{p,2})$

– count how many $y_p = \hat{y}_p$