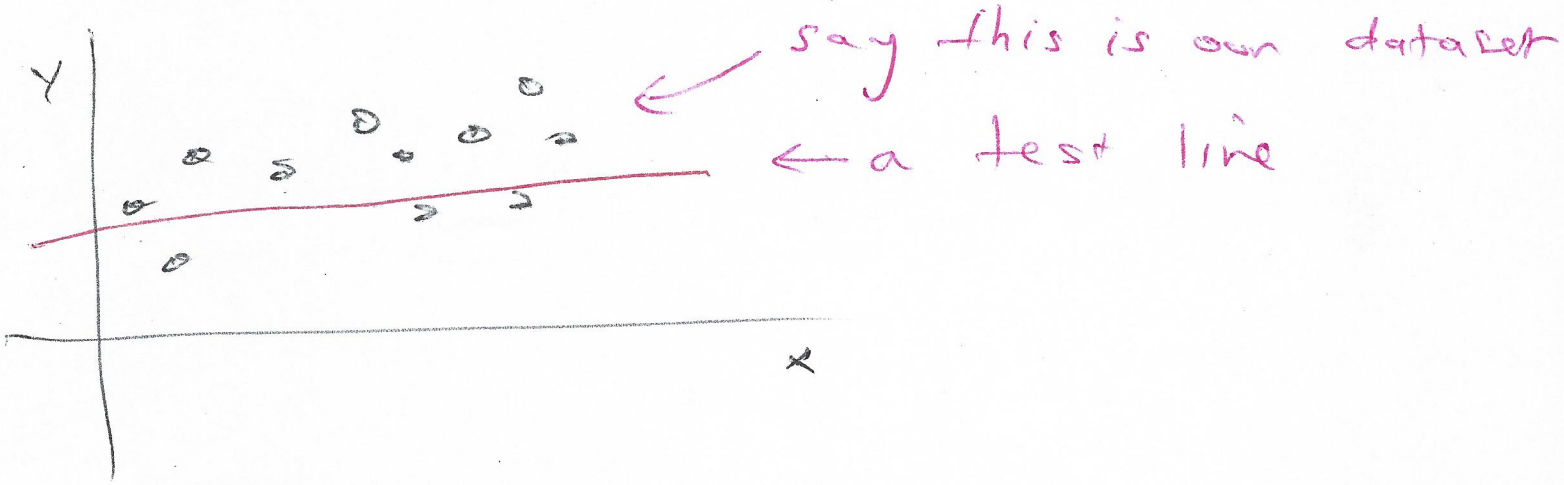
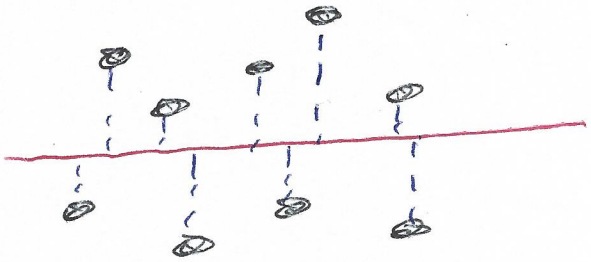
Linear regression

- Having covered cost function optimization, we're ready to tackle our first ML problem and learn how to code it up.
- Remember with linear regression we want to fit a line as best as possible to a given dataset
  - in higher dimensions a line = a "hyperplane".
- Lets get to modeling



- Remember - we want the line which minimizes the total error between it and all the data points

(our dataset, zoomed in)



the length of these dashed lines are our individual errors

- Notation for points: denote a point as an input/output pair

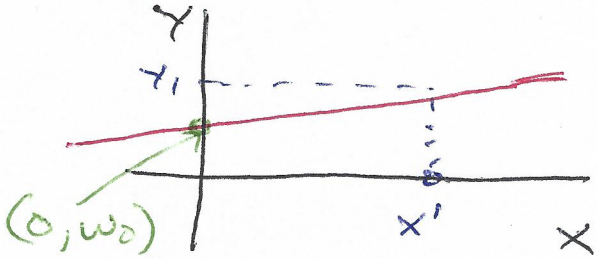
input, corresponding output

$(x_1, y_1)\ (x_2, y_2)\ \dots\ (x_P, y_P)$

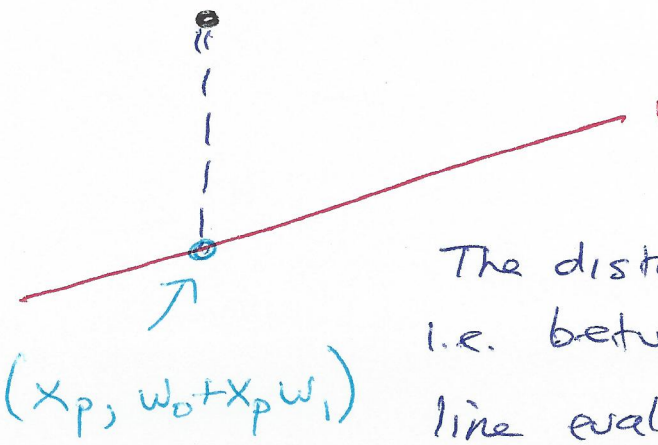
P points

- the equation of a line: $w_0 + x w_1 = y$

vertical intercept or bias, slope

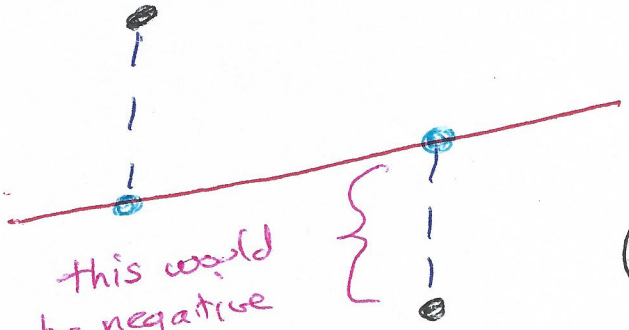


- So formally error between a single point (x_p, y_p) and a line



$$(w_0 + x_p w_1 - y_p)^2$$

The distance is vertical i.e. between y_p and the line evaluated at x_p: $w_0 + x_p w_1$



$w_0 + x w_1 = y$

$(w_0 + x_p w_1 - y_p)^2$

- by squaring the difference / all errors are positive, so we can treat all errors the same

- the total error is then just the sum of these errors over the whole data set

$$g(w_0, w_1) = \sum_{p=1}^{P} (w_0 + x_p w_1 - y_p)^2$$

- we want this to be as small as possible

- a cost function of 2 params

- can write it in vector format if we want

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \tilde{x}_p = \begin{bmatrix} 1 \\ x_p \end{bmatrix}$$

$$g(\vec{w}) = \sum_{p=1}^{P} (\tilde{x}_p^T \vec{w} - y_p)^2$$

- same thing, more compact notation

- We can apply gradient descent to minimize this cost function! Just need to calculate derivatives

- One can compute

$$\frac{\partial}{\partial w_0} g(w_0, w_1) = 2 \sum_{p=1}^{P} (w_0 + x_p w_1 - y_p)$$

$$\frac{\partial}{\partial w_1} g(w_0, w_1) = 2 \sum_{p=1}^{P} (w_0 + x_p w_1 - y_p) x_p$$

That gives us the gradient for gradient descent. $\nabla g(\vec{w}_0, w_1) = \left[\frac{\partial}{\partial w_0} g(w_0, w_1), \frac{\partial}{\partial w_1} g(w_0, w_1)\right]^T$

→ This can be written in a number of ways

here is one compact way of writing the gradient - completely equivalent

$$\nabla g(\vec{w}) = 2\left(\sum_{p=1}^{P} \tilde{x}_p \tilde{x}_p^T\right)\vec{w} - 2\sum_{p=1}^{P} \tilde{x}_p y_p$$

- A nice to know: is the cost function convex? YES. One can use math to logically deduce this in general, see chap 3 of my book for more and for N=2 you can just draw the cost function and see this!

To-do: Code up Gradient Descent for linear regression

- the Gradient calculation is given to you, you must complete the Gradient Descent loop + cost function value calculator ~~to~~
- tune the step length / by recording the cost history in your descent loop, then plot it to make sure your algo is working properly
- Use the 2-d linear regression dataset from Level 1 regression notebook

- Make sure to also return your learned weights, so you can plot your best-fit line to the dataset

- produce a plot of the dataset and your best fit line learned from linear regression by Gradient Descent.

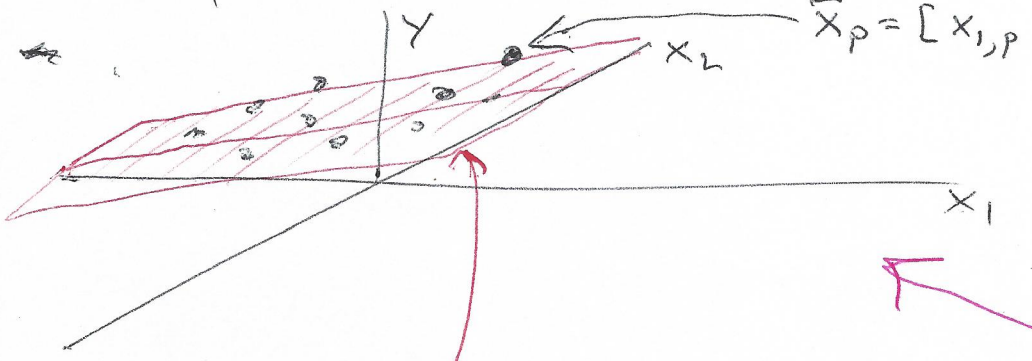
Measuring performance:

- How to measure how well our best fit line has done?
- the cost function value already gives us a measurement — the total error!
- common to take average error per point i.e. $\frac{1}{P} g(\vec{w}^K)$

Higher dimensions

- Everything previously described regarding Least Squares cost function generalizes directly to N dimensional input.

Output still a scalar.



Just need some additional notation

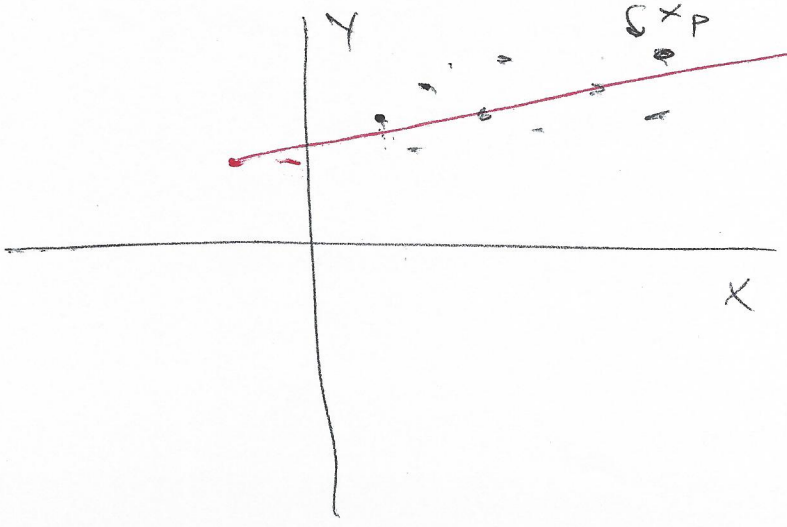
Compare

$w_0 + x_1 w_1 + x_2 w_2 = y$

↑ vertical intercept

↑ "slope" along x_1 dim

↑ "slope" along x_2 dim



$w_0 + x w_1 = y$

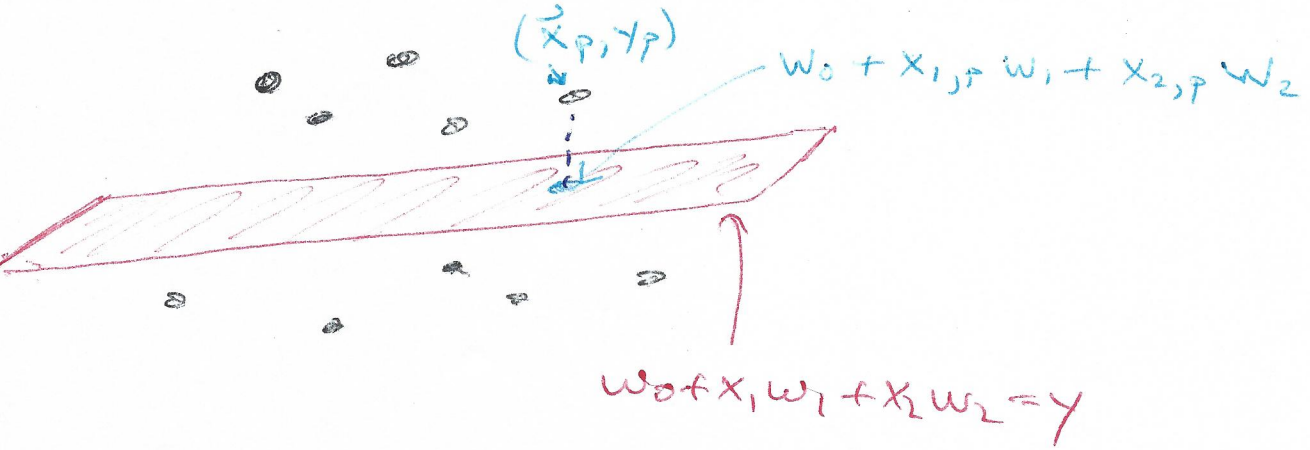
↑ vertical intercept

↑ slope along w

- Form of error between data points

$(\vec{x}_1, y_1)$ $(\vec{x}_2 y_2)$ $\cdots$ $(\vec{x}_p y_p)$ the same

(zoom in on plane and 3-d data)



- So error on $(\vec{x}_p, y_p)$

$$\left(w_0 + x_{1,p} w_1 + x_{2,p} w_2 - y_p\right)^2$$

or using vector notation

$$\left(\tilde{x}_p^T \vec{w} - y_p\right)^2 \quad \text{where} \quad \tilde{x}_p = \begin{bmatrix} 1 \\ x_{1,p} \\ x_{2,p} \end{bmatrix} \quad \vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

So Least Squares cost written exactly the same way

$$g(\vec{w}) = \sum_{p=1}^{P} \left(\tilde{x}_p^T \vec{w} - y_p\right)^2$$

- And gradient takes the same form too!!

= Cost function therefore <u>still convex</u>

Aside

- A note to the probability lovers in the audience: Using probabalistic assumptions one can derive this exact Least Squares cost function. The main assumption used here is that the error $(\tilde{x}_p^T \vec{w} - y_p)^2$ is <u>normally distributed</u>, and this leads to a Maximum Likelihood Estimate producing precisely the Least Squares cost function seen here.
- the power of the perspective we

took here was that it was simple ①

Simple = good, you don't need any more tools than vector algebra and a few pictures to derive Least Squares this way

— the probab-listic derivation takes more work, ~~into~~ requires additional tools, and is less visual. In short, it is less simple. So our way has many advantages

— However it does provide one profound insight we can't really get going the geometric way: that if the ~~noise~~ error on the data is not normally distributed (often called noise) than the squared error manner of determining performance will fail

- this, however, is more of a theoretical insight than a practical one.

in practice we don't know how the error is distributed

To-do:

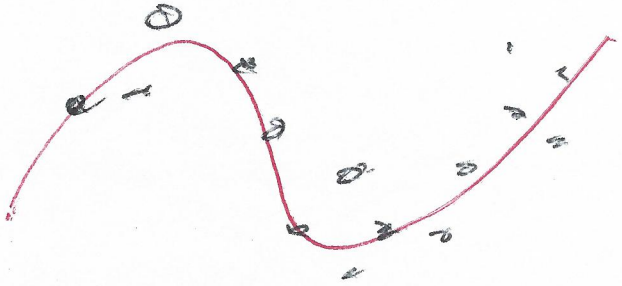
- Perform linear regression on the 3-d toy dataset in the Level 2 subdirectory.
- Use gradient descent to minimize the Least Squares cost function properly
- Use any method you desire to debug your program.
- You can copy the gradient computation function from the previous to do item.

Up next - linear classification

. . .

WOAH, WOAH, WOAH?!

How do we fit nonlinear regressors to a dataset?



We'll get there! But first lets hammer out all the linear cases (for regression and classification). Then we'll circle back and deal with the nonlinear extensions of both in one ~~fell~~ fowl swoop