Existential dread. And machine learning!

Or, How to fit a really sexy line of best fit

Here's a question: How much will water levels rise in the next 5 years? In this essay we will look at a way of tackling this problem.

A bit on lines and introducing the hypothesis

A line of best fit is a straight line that best represents the data on a scatter plot. The line may pass through some of the points, none of the points, or all of the points. Lines of best fit are extremely useful tools as they attempt to represent data as a function, enabling us to predict data values – with a reasonable degree of accuracy - that aren't yet known. So, how do you go about evaluating an equation (line or curve) that represents data on a scatter plot?

A start might be to simply just plot all your data values, then take a ruler and draw a line that roughly represents the correlation between the data points. Then you can find the gradient of this line and from there you can evaluate a linear function which will (roughly) be able to predict data values corresponding to chosen parameters. However, this is a very rough estimate, and will certainly not give you an accurate method to predict unknown values – like mean sea-levels in 2025 – so how can we increase the accuracy of our equation? This is where linear regression comes in...

Linear regression's job is to attempt to model the relationship between two (or more) variables by fitting a linear equation to already observed data values. Due to the word limitations on this essay we will only look at univariate linear (and later polynomial) regression but more complicated multivariate regression can be used to more accurately map input variables to some continuous function (this is another benefit of regression, as it allows you to have hundreds or even thousands of input variables – good luck trying to work that out by hand!). So how do we go about using linear regression I hear you ask? Well we will start off by naming our function 'h', where 'h' is a function which maps input values to output values. We can represent 'h' like this:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

This is called our hypothesis. θ_0 and θ_1 are simply parameters. So, by varying the values of our parameters, our linear equation changes – in fact, we can represent any straight line just by changing out parameters! Awesome!

Cost function and gradient descent – making the hypothesis useful

Now that we've set up a hypothesis, we need to find a way of choosing suitable parameters such that our function accurately models the relationship between our input and output variables - this is where the cost function comes in.

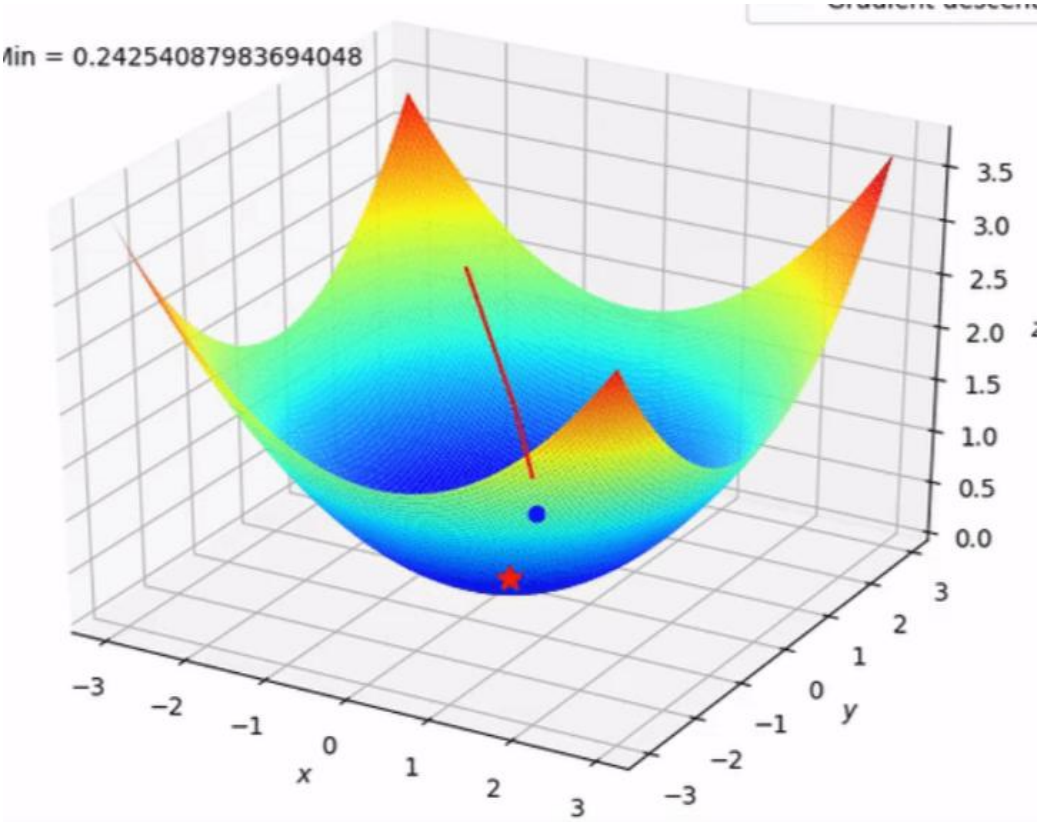
We need to choose θ_0 and θ_1 such that $h_\theta(x)$ is close to our observed values from our data set. The cost function (aka. the squared error function) is a measure of the accuracy of our hypothesis. For each data point, the cost function calculates squared difference between our hypothesis (what we *expect* the y value to be) and the *actual* y value. It then divides the sum by "m" (the number of data points) to get an average, and divides it all by 2 so that it becomes easier to differentiate later on! The cost function, J(θ), is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

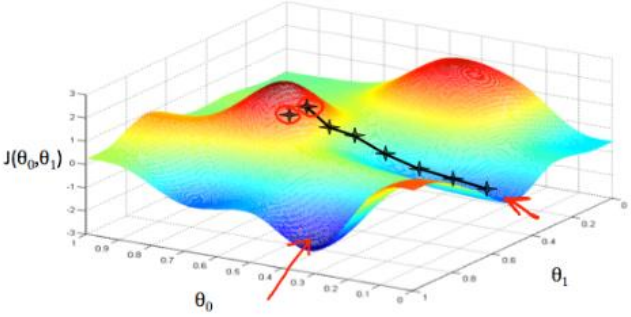
x_i *and* y_i *represent an observed "input" value and its corresponding observed "output" value ("i"th example from dataset), and "m" represents the number of data values we have in our dataset.*

The goal is to minimize the cost function such that in an ideal world the output of the cost function when iterating through our data values is equal to 0 – which would mean our equation fits all of our data with 100% accuracy – but naturally that is pretty unlikely, so we just need to minimize the value of the cost function as much as possible.

The cost function for our specific data and hypothesis can be graphed (with parameters along the x-/y-axis and J(θ) on the z-axis), giving a certain landscape where the lowest point (minimum) represents the values of our parameters that best minimize our cost function J(θ).

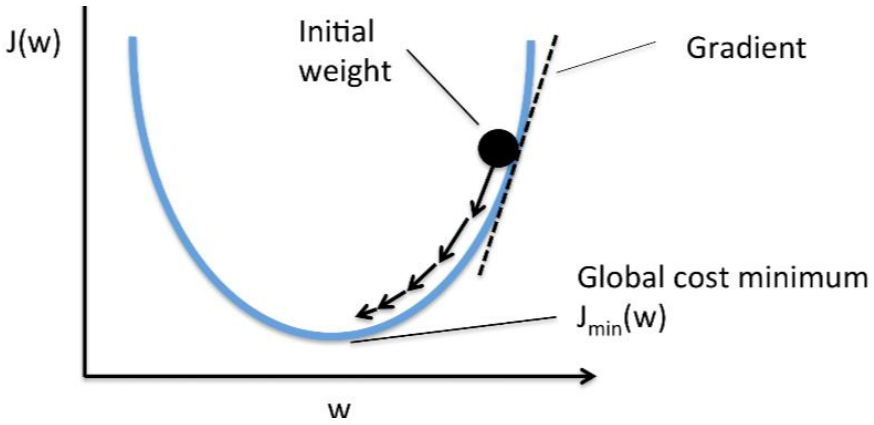


Now, it's well and good saying "cool, so we just simply choose the values of our parameters that minimize the cost function, right?", but how can we do that? For this, we need another algorithm, called gradient descent. Gradient descent is a first-order optimization algorithm used to minimize (find a local minimum) of some function – in this case, our cost function - by iteratively moving in the direction of steepest gradient. This iterative algorithm (starting at a set coordinate) with move towards a local minimum and therefore minimize the cost function; it can be thought of as a way to solve the derivative of the cost function = 0. To help you visualize what the algorithm does, below is a handy diagram showing gradient descent minimizing an arbitrary function. The line of greatest slope is taken by the algorithm to go down to a local optima, thus minimising the cost function.



Here you can see, from a given point, the gradient descent algorithm has managed to converge on a local minimum. Which in turn gives us values for our parameters that will minimize the cost function and give us an optimal equation to model the relationship between our input and output variables. Exciting stuff, but how does this algorithm actually work? Here is the formula and a pretty diagram to hopefully give intuition on why the formula works:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$



We want to get the black ball to the global minimum. Notice how the further away it is from a minimum, the larger the magnitude of its gradient. Also, if the ball needs to travel backwards, the gradient is positive, and if the ball needs to travel forwards, the gradient is negative. By taking the partial derivative of the cost function with respect to each theta value to get a gradient, and then applying the formula above many times, we eventually get to an optimum!

Looking at the formula, in our case j is equal to 0 and 1 (our parameters), J(θ) is our cost function, and alpha is a value called the 'learning rate'. We won't go into great detail on what the learning rate is, but essentially it is how big the steps taken in the direction of a local minimum are.

When applying gradient descent, we 'repeat until convergence', which is when the difference in the parameters after each iteration becomes negligible (meaning it has converged on a local minimum). Now this is where it gets rather complicated, we can now combine our cost function with the gradient descent iterative formula to give us one master equation – it is as follows:

$$\textbf{repeat until convergence: } \{$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

$$\}$$

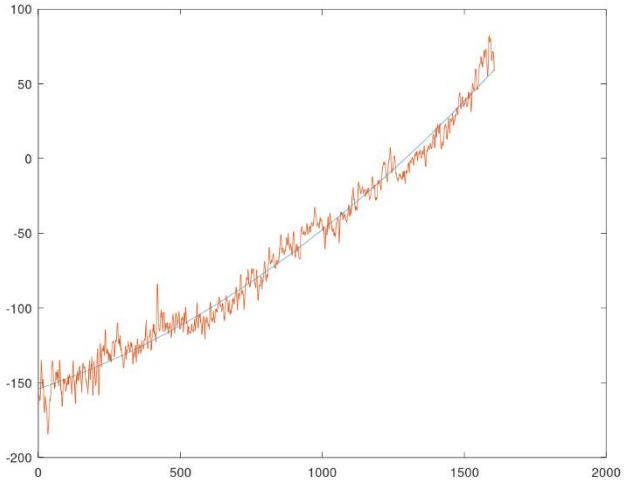
This can be derived by substituting the equation for the cost function into the gradient descent algorithm and using some fancy calculus (using partial derivatives).

So, we've arrived at this grand iterative equation capable of working out the meaning of life, and - as it's more commonly used for - the values of our parameters which give us a beautiful hypothesis. Now I think it's time to return back to our original problem (sea-levels in 2025) and apply this bad boy.....

Getting down and dirty

```
Time,GMSL,GMSL uncertainty
1880-01-15,-183.0,24.2
1880-02-15,-171.1,24.2
1880-03-15,-164.3,24.2
1880-04-15,-158.2,24.2
1880-05-15,-158.7,24.2
1880-06-15,-159.6,24.2
1880-07-15,-159.6,24.2
1880-08-15,-161.8,24.2
1880-09-15,-158.9,24.2
```

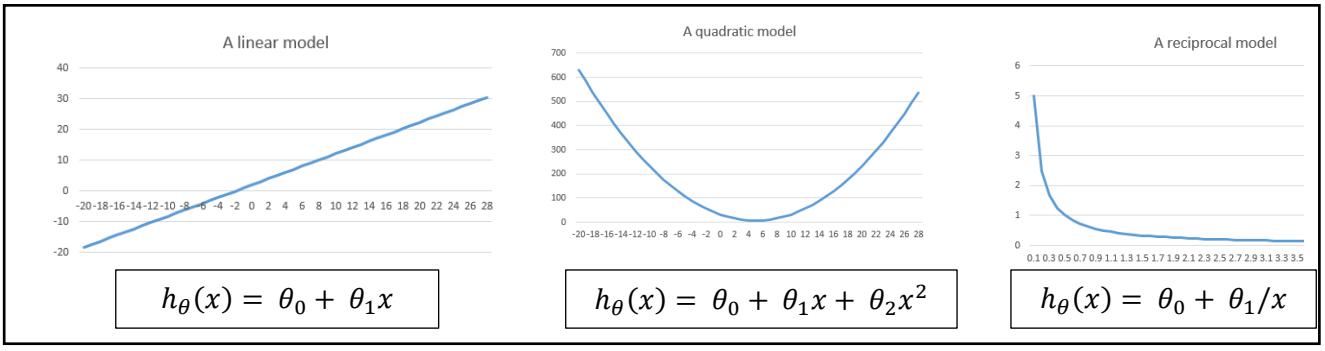
Above is a short snippet from the dataset we are using, showing sea-levels in the first few months of 1880. To simplify the problem, lets convert the date into an integer (representing number of months since January 1880), and we will only concern ourselves with GMSL (global mean sea-level in mm). Note that the sea-level data is measured compared to the sea-level in June 1990 – this means that in June 1990, the sea-level is recorded as '0'.



A plot of global sea-level (mm) against time

Now time to get line-of-best-fit-ing - let's pick the hypothesis!

We can define our hypothesis as one of a series of 'models' – where each model uses a slightly different function. For example, the model we already know and love is linear, but we can also get a bunch of other truly beautiful (and also maybe not so beautiful) models. When finding the best line – or rather, *curve* - of best fit, we want to pick the model that works best for our data.



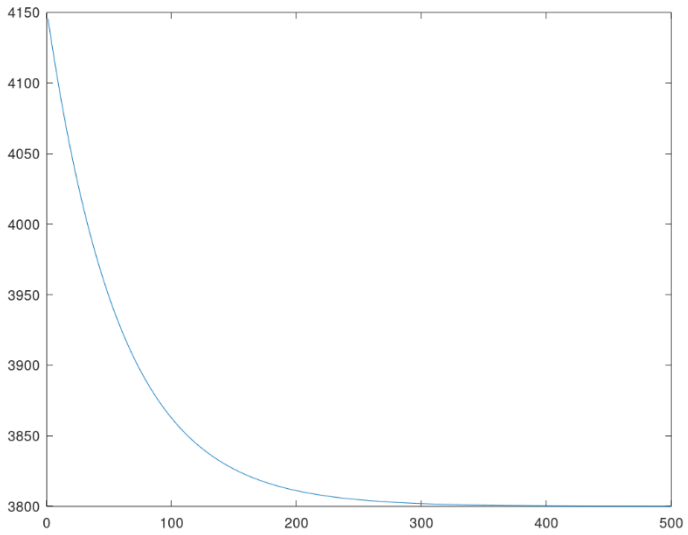
Above are three examples of different models

Notice that to spice up our hypothesis, we add different flavours of x (e.g. x squared, or 1/x), multiplying each term by a coefficient – a 'theta'. That way, when we adjust the values of theta, we adjust the nature of the function we are dealing with (e.g. if you want a miserable quadratic, grab the centre model below, and set theta-2 to '-1'). The cost function remains unaltered; this means we can use gradient descent just as normal to optimise each parameter theta to fit our data! All we need to remember to do is to adjust <u>all</u> theta parameters with every iteration!

$repeat\ until\ convergence$:

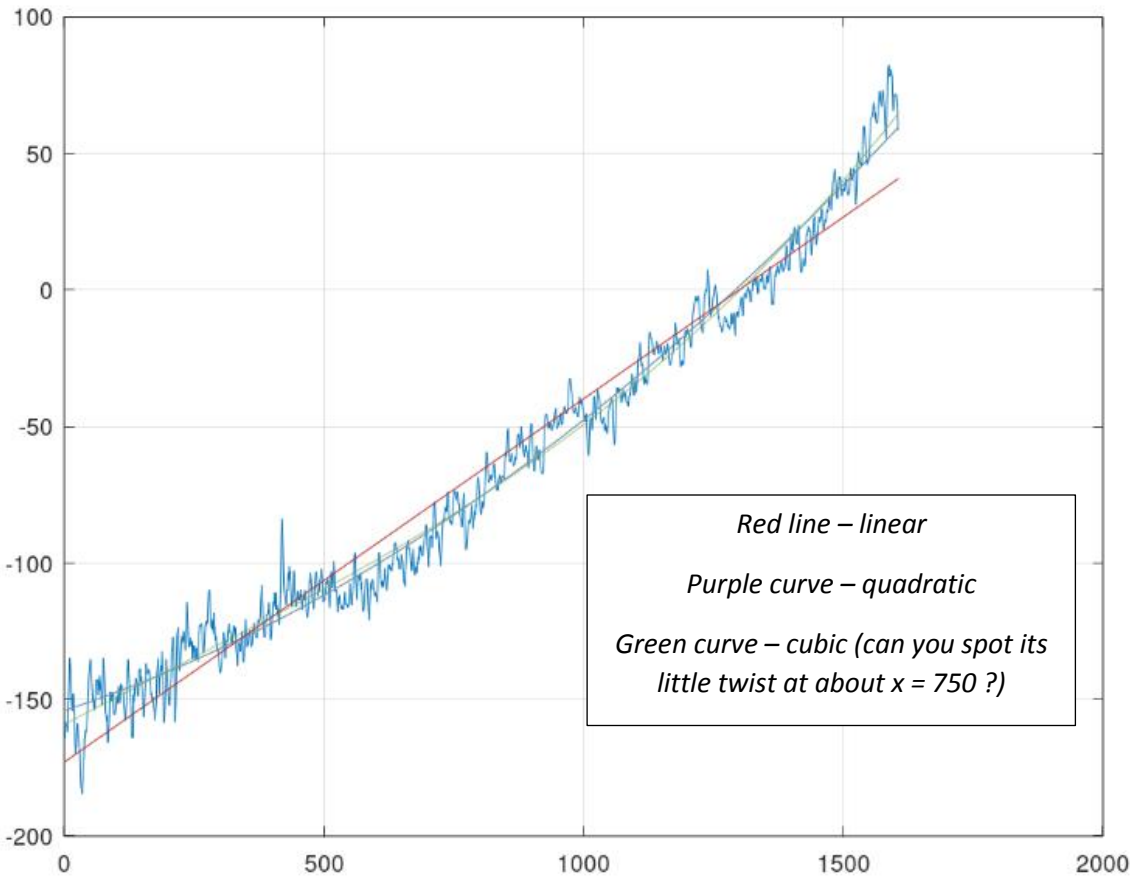
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \text{(for all j)}$$

We used three models and optimised each, tracking the cost of each model as gradient descent was running. This gave the following, really quite pleasing, graph! We can see that with each iteration, the accuracy of our model increases (i.e. cost goes down as theta parameters become optimised). After many iterations, we approach a minimum cost – we have convergence and can stop iterating!



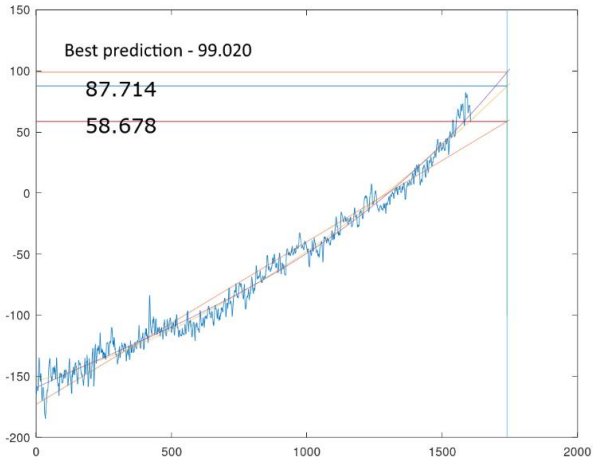
Along the x axis is the number of iterations of gradient descent, and the y axis shows the cost!

Now to see the plot – does it have curves in all the right places?? Notice that we have plotted a linear, quadratic, and cubic curve of best fit on the graph – but the cubic curve only differs very slightly from the quadratic. This is because, during optimisation, the gradient descent algorithm realised that the data points aren't hugely 'cubic' in form. This resulted in a very small coefficient for the x^3term (about 0.012), giving the curve just a little more '*finesse*' than the quadratic, but nothing crazily different!



Gradient descent – it's done it again!!

We can now extrapolate to 2025:



We have labelled the cubic model's prediction as the 'best prediction', as a cubic curve has more freedom to wrap itself closer to the data, due to its increased number of parameters.

Generally, as you increase the order of a polynomial regression curve, your cost decreases. As always, there's a drawback – you may 'overfit' your data. Luckily we have a big dataset of over 1500 entries, so we don't need to worry much about overfitting with a cubic model!

Conclusion

Phew! At this point, it's only fair to list off a few limitations of our model:

- There's only one input variable – time. This gives us a simpler hypothesis to optimise, but this model might be too simplistic (we call this 'having bias'). A better model might account for industry growth, and the growing social movement to cut emissions and reduce global warming.
- Extrapolation is always risky – but unavoidable. It causes uncertainty, so that is why we only predict to 2025, rather than something crazy like 2300. Let's leave that to the professions.
- We are using data we found in the public domain – although this means our data might not be as reliable as data we have compiled ourselves, at least it shows the fun you can have with the free data that's already out there!

BUT hopefully you'll agree that our model doesn't seem too outlandish, based off the data. Using the cubic model as our hypothesis, we can predict that sea-levels will be at 99mm by January 2025 - an increase of 40mm compared to December 2013; 4cm in only 12 years! Not very fun…

Thanks for the existential dread, but what about the machine learning?

A machine learning system can be defined as a system that becomes increasingly better at a task (based off a certain performance indicator) as it gains experience.

This is exactly what we have been doing with our regression ('curve-of-best-fit') problem – the task is to predict sea-level, the performance indictor is our cost, and experience is provided by exposing the computer to data that it can 'learn' a hypothesis function with. Regression is a widely-used form of machine learning, but the field encompasses many other learning paradigms including logistic regression (often used for simpler classification), neural networks (the buzzword of the decade, traditionally used for complex classification), and unsupervised learning (e.g. for identifying patterns in DNA sequences). The algorithms we used – especially gradient descent – are found throughout the AI field, to optimise algorithms. Pretty sexy, eh?

Harvey Yorke and Fabian Sturman

All the programming done in Octave.

Email fabiansturman@hotmail.co.uk for the code behind the graphs!

Some of the general illustrations on gradient descent by Andrew Ng.

Dataset source: https://pkgstore.datahub.io/core/sea-level-rise/csiro_recons_gmsl_mo_2015_csv/data/730bd7382bf3ce1e997becb2fadf40e6/csiro_recons_gmsl_mo_2015_csv.csv, accessed 10/3/2020