# Machine Learning for the eager theorist

james.pallister

December 2023

## 1 Introduction

If, like me, you are looking to learn about Machine Learning (ML) and are coming from a theoretical physics background then look no further. Here, I will present the essentials as simply as possible by appealing to one's intuition gained from a life time of doing physics. It transpires that, perhaps unsurprisingly, most of the techniques used in ML have been stolen from physics![1]. This is great news because it means you will be able to understand a lot about this vast subject in a very short amount of time. Read on!

## 2 Overview

Roughly speaking in ML we want to train a computer on a series of inputs and outputs and then use this training to make predictions based on real-world inputs. The applications of this abound, so it is no surprise that more and more companies are interested in hiring people who know something about it.

How, then, does the training work? The answer is simple and well known to any physicist, we make an *ansatz* about the relation between the inputs and outputs and then go from there; adjusting key parameters until the model makes good predictions (definition of 'good' in this context to follow). Question: how should we adjust these parameters? Answer: we penalise the deviation from the correct outputs and seek to *minimise* this penalty. Therein lies the physics: ML is a minimisation problem. Exactly like how a theorist would minimise an action, the data scientist must minimise the so-called 'cost function' which computes the penalty associated to an ill-fit.

No one has ever learnt anything by reading an overview so now that we are done with it let's move on to the single most important *ansatz* in all of ML: Linear Regression.

[1] a mathematician would say they invented them first but we won't be paying any attention to them.

# 3 Linear Regression

The aim of LR is to take some inputs, $\mathbf{x}$, and then fit the line

$$\beta_0 + \beta_1 \mathbf{x} \tag{1}$$

to a set of outputs $\mathbf{y}$ by suitably choosing $\beta_0, \beta_1$. In fully-fledged ML $\mathbf{x}$ is often a matrix where each column is a separate data set to be fitted. If each data set has $n$ points in it then it will be an $n \times m$ matrix. For now we will work with just one set of data and so $\mathbf{x}$ is to be understood as the vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \tag{2}$$

(and similarly for $\mathbf{y}$).

For any choice of the parameters $\theta = (\beta_0, \beta_1)$ we can write

$$\mathbf{y} = \beta_0 + \beta_1 \mathbf{x} + \epsilon, \tag{3}$$

i.e. we have tried to fit the data but perhaps this fit isn't very good, in which case we have an error associated to each of the data points with vector $\epsilon$. Naturally, one would like to minimise these errors by an appropriate choice of $\theta$.

First, let us perform some elementary manipulations on Eq. (3) to write

$$\epsilon(\theta) = \mathbf{y} - (\beta_0 + \beta_1 \mathbf{x}) \,. \tag{4}$$

The temptation is probably all to great at this stage to write

$$\nabla \epsilon = 0, \tag{5}$$

and solve, where our notation involves the gradient operator taken in the *two-dimensional space* parameterised by the coordinates $\beta_0, \beta_1$, i.e. $\partial_{\beta_0} \epsilon_i, \partial_{\beta_1} \epsilon_i, = 0, 0$ for every $i \in (1, n)$. It should be easy to convince you this isn't a good idea: there are now a whopping $n$ equations for just 2 unknowns, none of which know anything about each other. A simple remedy is to instead minimise

$$S(\theta) = \frac{1}{n} \epsilon^T \epsilon, \tag{6}$$

otherwise known as the sum of the squares of the errors ($n$ is there for normalisation later). This is the so-called *cost function*. We thus wish to find where

$$\frac{\partial S}{\partial \beta_0} = 0, \quad \frac{\partial S}{\partial \beta_1} = 0. \tag{7}$$

Calling $S$ an action and $\beta_0, \beta_1$ coordinates we have what amounts to simple classical mechanics. We remind that for a non-relativistic particle of unit mass moving in $2D$ we have the following relations

$$\dot{\beta}_0 = \partial_{\beta_0} S = p_{\beta_0}, \quad \dot{\beta}_1 = \partial_{\beta_1} S = p_{\beta_1}, \tag{8}$$

having introduced conjugate momenta $p_\theta$. The minimisation constraint implies $p_\theta = 0$ and thus the linear regression problem becomes one of finding the *turning points* in phase-space of some classical system with action $S(\theta)$. One can also phrase this problem in terms of quantum mechanics and the path integral. Feynman's sum over paths tells us to consider the expression

$$\mathcal{A} = \int \mathcal{D}\theta e^{-S(\theta)/\hbar}, \tag{9}$$

where $|\mathcal{A}|^2$ is the probability amplitude or likelihood for the particle to move between two points[2]. In the semi-classical limit ($\hbar \to 0$) this path-integral is dominated by the stationary points of its action giving back Eq. (7). For the choice Eq. (6) the action is quadratic in which case the stationary phase approximation is exact. For other choices of $S(\theta)$ this might not be the case and the stationary phase analysis becomes invaluable.

When written explicitly we find

$$p_{\beta_0} = -\frac{2}{n}\sum_i^n (y_i - \beta_0 - \beta_1 x_i) \tag{10}$$

$$p_{\beta_1} = -\frac{2}{n}\sum_i^n x_i\,(y_i - \beta_0 - \beta_1 x_i)\,. \tag{11}$$

Thus

$$\dot{\beta}_0 = -2\left[\langle y\rangle - \beta_0 - \beta_1\langle x\rangle\right], \tag{12}$$

$$\dot{\beta}_1 = -2\left[\langle xy\rangle - \beta_0\langle x\rangle - \beta_1\langle x^2\rangle\right], \tag{13}$$

describing the trajectory of the particle or its flow. We now need the two expressions above to vanish resulting in

$$\beta_1 = \frac{\langle xy\rangle - \langle x\rangle\langle y\rangle}{\langle x^2\rangle - \langle x\rangle^2} \tag{14}$$

$$\beta_0 = \langle y\rangle - \beta_1\langle x\rangle. \tag{15}$$

These are the coordinates of the turning points and with them we have completed the minimisation process.

[2]the keen theorist may have noticed a conspicuous absence of i in the path integral. This is because we are interested in the saddlepoints found through steepest descent and not stationary phase

For the interested theorist we now evaluate the 'probability amplitude'. We rewrite it

$$\mathcal{A} = e^{-\langle y^2 \rangle} \int \mathcal{D}\theta \exp\{-\theta A \theta^T + 2B\theta^T\} \tag{16}$$

where

$$A = \begin{pmatrix} 1 & \langle x \rangle \\ \langle x \rangle & \langle x^2 \rangle \end{pmatrix}, \quad B = \begin{pmatrix} \langle y \rangle & \langle xy \rangle \end{pmatrix} \tag{17}$$

Elementary manipulations provide

$$\mathcal{A} = e^{-\langle y^2 \rangle + BA^{-1}B^T} \int \mathcal{D}\theta \exp\{-\left(\theta - BA^{-1}\right) A \left(\theta^T - A^{-1}B^T\right)\}. \tag{18}$$

It is now obvious that the minimum of the function lies at $\theta = BA^{-1}$ which one can check is the answer we had already arrived at. We then have

$$\mathcal{A} = \sqrt{\det(\pi A^{-1})} e^{-\langle y^2 \rangle + BA^{-1}B^T}. \tag{19}$$

# 4 Gradient Descent

The above was instructive but simple and it is easy to imagine that by introducing more complexity it might not be possible to analytically find $\theta$. Fortunately we don't have to because computers are pretty good these days. The most popular method is gradient descent. It involves approximating the flow equations as

$$\theta_{t+\alpha} = \theta_t - \alpha \nabla S. \tag{20}$$

This is the discretised version of (12, 13). $\alpha \sim dt$ is known as the learning-rate but it is really just the amount of time we are evolving the system for. Unituitvely we actually want to evolve the system backwards in time hence the minus sign in front of $\alpha$. This is the 'descent' part of gradient descent. One can then plug in their initial guess $\theta_0$ and then keep iteratively updating $\theta$ according to the above equation. The hope is that eventually one will descend to the (global) minimum of $S$. I say hope because it depends upon the specifics of $S$ and perhaps most importantly the choice of $\alpha$. Too small and it will not converge quickly enough, too large, and one may just miss the minimum completely. Most agree that $\alpha \sim 0.01 - 0.1$ is not a bad place to start [3].

[3] There are some clever algorithms out there which use a variable learning rate. One that slows the descent if the 'velocity' or gradient is too steep and speeds up when it is shallow. If we want $\theta$ to have the dimensions of a coordinate then $\alpha$ must have units of time/mass for dimensional consistency. We can then think of a variable learning rate as like a particle with variable mass.