Delta Capita

Mathematics behind Machine Learning

Toby Alfred-Jones

Version v2.0
November 2019

What is Machine Learning?

The purpose of this white paper is to introduce machine learning and to explain the mathematics behind some of the most common techniques.

What is Machine Learning?

Machine learning is the process of analysing past data to find patterns and using computer models and algorithms to predict future outcomes independently of humans.

Types of Machine Learning

Machine learning can be categorised into 3 broad categories: **supervised learning**, **unsupervised learning** and **reinforcement learning**.

In all 3 categories, the goal is to learn a function $h: X \to Y$. Here, X is a set of inputs and Y is a set of outputs. Example inputs may include images, text or financial time series. Outputs may be either quantitative or qualitative, for example a stock price or yes/no.

In supervised learning, we know what the set Y is; i.e. we have access to some **training data** consisting of input/output pairs. This training data consists of n tuples (x_i, y_i), with $x_i \in X$, $y_i \in Y$. In unsupervised learning, we are just given input data and a task (such as clustering); our training data only consists of the values $x_i \in X$. Reinforcement learning is concerned with how an algorithm will take actions to maximise a notion of reward.

Supervised Learning: Regression

An example of supervised learning is **regression**. In regression, our set of outputs Y consists of quantitative values. The most well-known kind of regression is **linear regression**. In linear regression, we assume that the relationship between y_i and x_i is linear. If each input x_i consists of k features, we assume we have the equality

$$y_i = \sum_{j=1}^{k} m_j x_{i,j} + c. \qquad (1)$$

The regression problem consists of determining which values of $m = (m_1, \dots, m_k)$ and c result in the best fit for our training data. But how do we quantitatively define "best fit"?

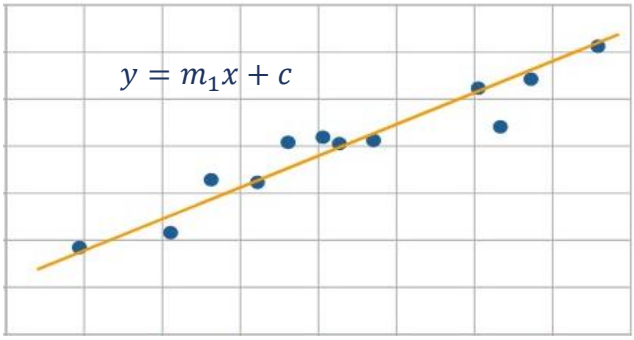


Figure 1: An example of linear regression with one feature ($k = 1$).

A **loss function** encodes the error between our predicted value and the observed value. **Ordinary least squares** (**OLS**) is the most common variant. In OLS, we aim to minimise the sum of squared errors

$$E(m, c) = \sum_{i=1}^{n} \left(y_i - \left(\sum_{j=1}^{k} m_j x_{i,j} + c \right) \right)^2. \qquad (2)$$

Observed value

Predicted value

Notice that if we do have a linear relationship, that is equation (1) holds exactly for each of our n data points, then the error $E(m, c)$ in (2) is zero. This is not the case in Figure 1, as the line of best fit does not pass precisely through each point.

Why do we square the errors? There are two reasons for this:

1. Squaring ensures that larger errors are punished accordingly; e.g. doubling the difference between a predicted and observed value results in a four-fold increase in the error. This is shown in Figure 2.
2. Squaring ensures that our function E is differentiable.

Supervised Learning: Regression

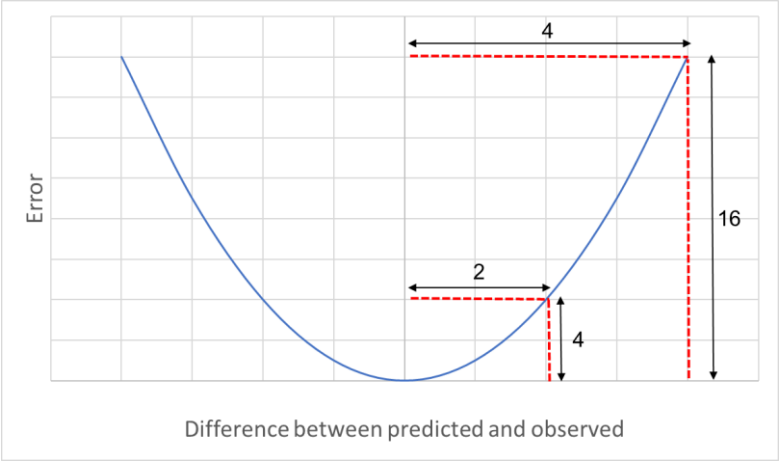


Figure 2: Squaring ensures larger errors are treated more harshly.

The second point requires some exposition. We could consider a loss function which simply sums up the differences, as in equation (3)

$$E(m,c) = \sum_{i=1}^{n}\left(y_i - \left(\sum_{j=1}^{k} m_j x_{i,j} + c\right)\right). \quad (3)$$

This certainly is differentiable; however it does not account for upside and downside errors correctly. We might try and remedy this by using an L^1 loss function, as in equation (4), which treats upside and downside errors equally by summing the absolute values of the difference between the predicted and observed values

$$E(m,c) = \sum_{i=1}^{n}\left|y_i - \left(\sum_{j=1}^{k} m_j x_{i,j} + c\right)\right|. \quad (4)$$

However, this is not differentiable, as the absolute value function is not differentiable. Why do we require a differentiable loss function? In order to minimise the loss function, we use **gradient descent**; this algorithm requires differentiability.

We employ a numerical optimisation algorithm, gradient descent, as for high-dimensional problems it is impossible to minimise our loss function algebraically. Gradient descent is a first order iterative optimisation algorithm which finds the minimum of a function.

Our aim is to find the values of m and c which minimise E. We start with an initial guess, $(m^{(0)}, c^{(0)})$, and move in the direction of fastest decrease, $\nabla E(m^{(0)}, c^{(0)})$[1]. How far we move in this direction is determined by a step-size δ_0. Our iterative procedure looks like this (and is illustrated in Figure 3):

$$\underbrace{(m^{(i+1)}, c^{(i+1)})}_{\text{Next guess}} = \underbrace{(m^{(i)}, c^{(i)})}_{\text{Previous guess}} \underbrace{-\delta_i}_{\text{Step size}} \underbrace{\nabla E(m^{(i)}, c^{(i)})}_{\text{Direction of fastest decrease}}. \quad (5)$$

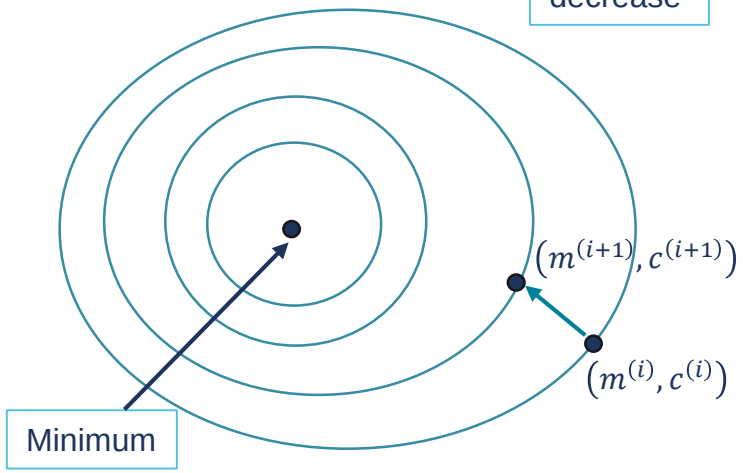


Figure 3: The iterative procedure of gradient descent.

It is important to choose our step size δ_i carefully. Too small a step size and the algorithm will converge to the minimum too slowly. Too large a step size and we may overshoot and miss the minimum completely. With certain assumptions on E and choices of step sizes, we can guarantee convergence to the minimum.

Supervised Learning in Finance

The well-known capital asset pricing model (CAPM) is an often-used regression model in finance. Under certain market assumptions (these are quite strong and do not often match reality), the expected return R_π of a portfolio π can be expressed in the form

$$\mathbb{E}[R_\pi] = r + \beta_\pi(\mathbb{E}[R_m] - r), \quad (6)$$

[1]: This is the grad vector of partial derivatives of E.

Unsupervised Learning: Clustering

where r is the risk-free interest rate and R_m is the return of the market portfolio m. This can be phrased as a linear regression problem to determine the coefficient β_π; by rearranging and setting $y = \mathbb{E}[R_\pi] - r$ and $x = \mathbb{E}[R_m] - r$ we can write equation (6) as $y = \beta_\pi x$.

Unsupervised Learning: Clustering

An example of unsupervised learning is **clustering**. Clustering is the task of grouping objects together such that objects in the same group are more similar to each other than those in other groups. This is unsupervised as we do not define the clusters beforehand, we let the algorithm define them. The most well-known clustering algorithm is ***k*-means clustering**.

Our input data consists of a set of n d-dimensional observations $\{x_1, \dots, x_n\}$. k-means clustering aims to partition this set into k clusters $C_1, \dots, C_k$, where we aim to minimise the distance from the mean within each cluster. The mean μ_i of cluster C_i is defined below in equation (7),

$$\mu_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j \,, \qquad (7)$$

where $|C_i|$ is the number of points in cluster C_i. We want to minimise the sum of the distances of each point in each cluster from the mean of the cluster. The algorithm we use to solve this problem uses an iterative refinement technique. We initialise this algorithm with a set of k initial means $\mu_1^{(1)}, \dots, \mu_k^{(1)}$. A common initialisation technique is to randomly generate the initial means uniformly from within the data set range.

Suppose we are at the t-th step of the algorithm, so our current means are $\mu_1^{(t)}, \dots, \mu_k^{(t)}$. The clusters at this step are

$$C_i^{(t)} = \left\{ x_j : \sum_{m=1}^{d} \left| x_{j,m} - \mu_{i,m}^{(t)} \right|^2 \leq \sum_{m=1}^{d} \left| x_{j,m} - \mu_{l,m}^{(t)} \right|^2 \ \forall\, 1 \leq l \leq k \right\}. \qquad (8)$$

That is, the cluster $C_i^{(t)}$ consists of those points which are closer (in Euclidean distance) to $\mu_i^{(t)}$ than any other mean.

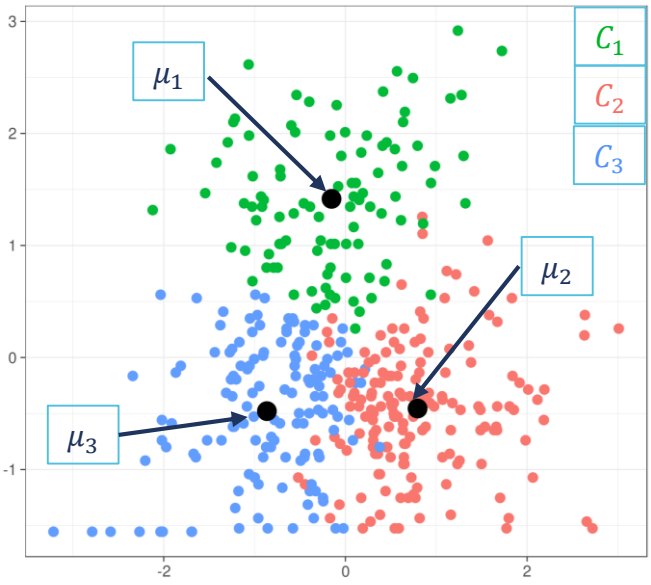


Figure 4: Clustering with 3 clusters.

We then update the means by setting

$$\mu_i^{(t+1)} = \frac{1}{\left|C_i^{(t)}\right|} \sum_{x_j \in C_i^{(t)}} x_j \,. \qquad (9)$$

The procedure stops when the means are no longer updated.

This algorithm has some limitations. It can be difficult to choose the parameter k and poor choices yields poor results. So some preprocessing may be required in order to determine the number of clusters. Furthermore, it can be shown that the algorithm performs arbitrarily badly under certain configurations. More complex methods, such as k-means++, address these issues.

Unsupervised Learning in Finance

Clustering is commonly used in portfolio diversification. Stocks that have similarly correlated returns will be clustered together; this ensures that separate clusters have low correlation, reducing the total risk.

Artificial Neural Networks

Artificial neural networks (ANNs) simulate the firing of neurons in biological neural networks, such as those in our brains. They consist of an input layer, one or more hidden layer/s and an output layer. Information is passed from layer to layer, with the output of the previous layer becoming the input into the next layer.

Each input x_i has a weight associated to it, w_i. This quantifies how important each input is to the output. We want a small change in weights to cause a small change in outputs. A typical output might be a sigmoid function; this is an S-shaped function that can be thought of a continuous version of an indicator function. An example of a sigmoid function is the logistic function; inputs x_1, x_2, x_3 with weights w_1, w_2, w_3 and bias b into the logistic function produces output

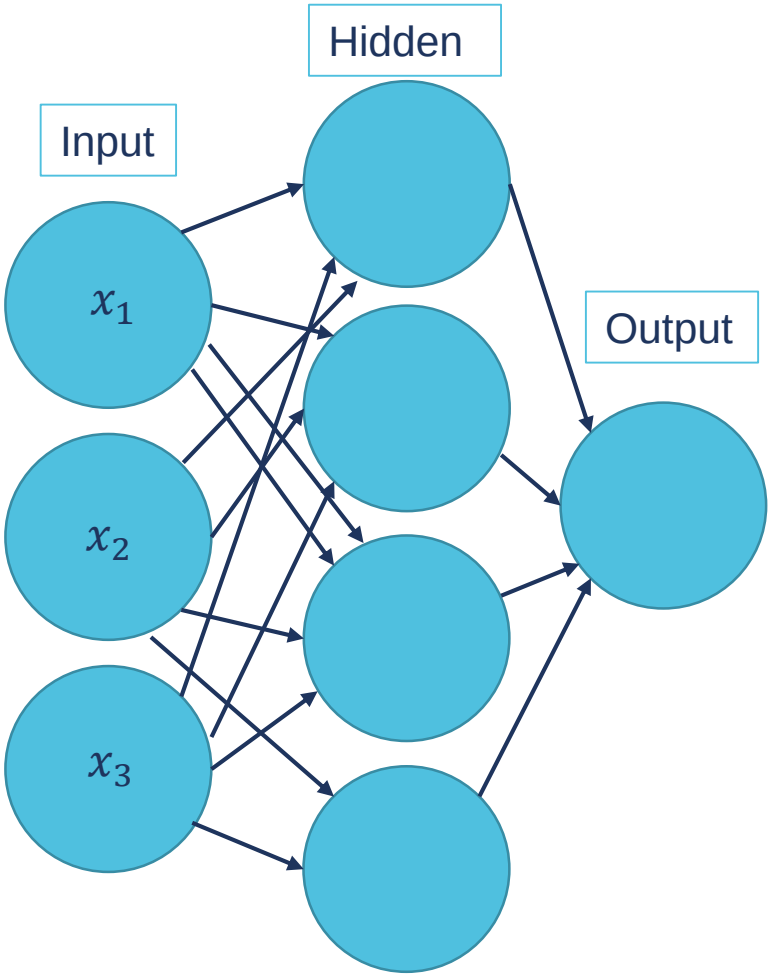


Figure 5: An ANN with 3 inputs and one hidden layer consisting of 4 neurons.

$$\frac{1}{1+\exp(-(w_1x_1+w_2x_2+w_3x_3)-b)}. \qquad (10)$$

The bias b is a threshold parameter. As with linear regression, we have a loss function that quantifies the accuracy of the output from the network,

$$E(w,b)=\sum_{x\in X}\|y(x)-y\|^2. \qquad (11)$$

Here w is the vector of weights, $y(x)$ is the output from the network with input $x \in X$ and $y \in Y$ is the actual output from the training data. The notation $\|y(x)-y\|$ is the Euclidean distance between the vectors $y(x)$ and y.

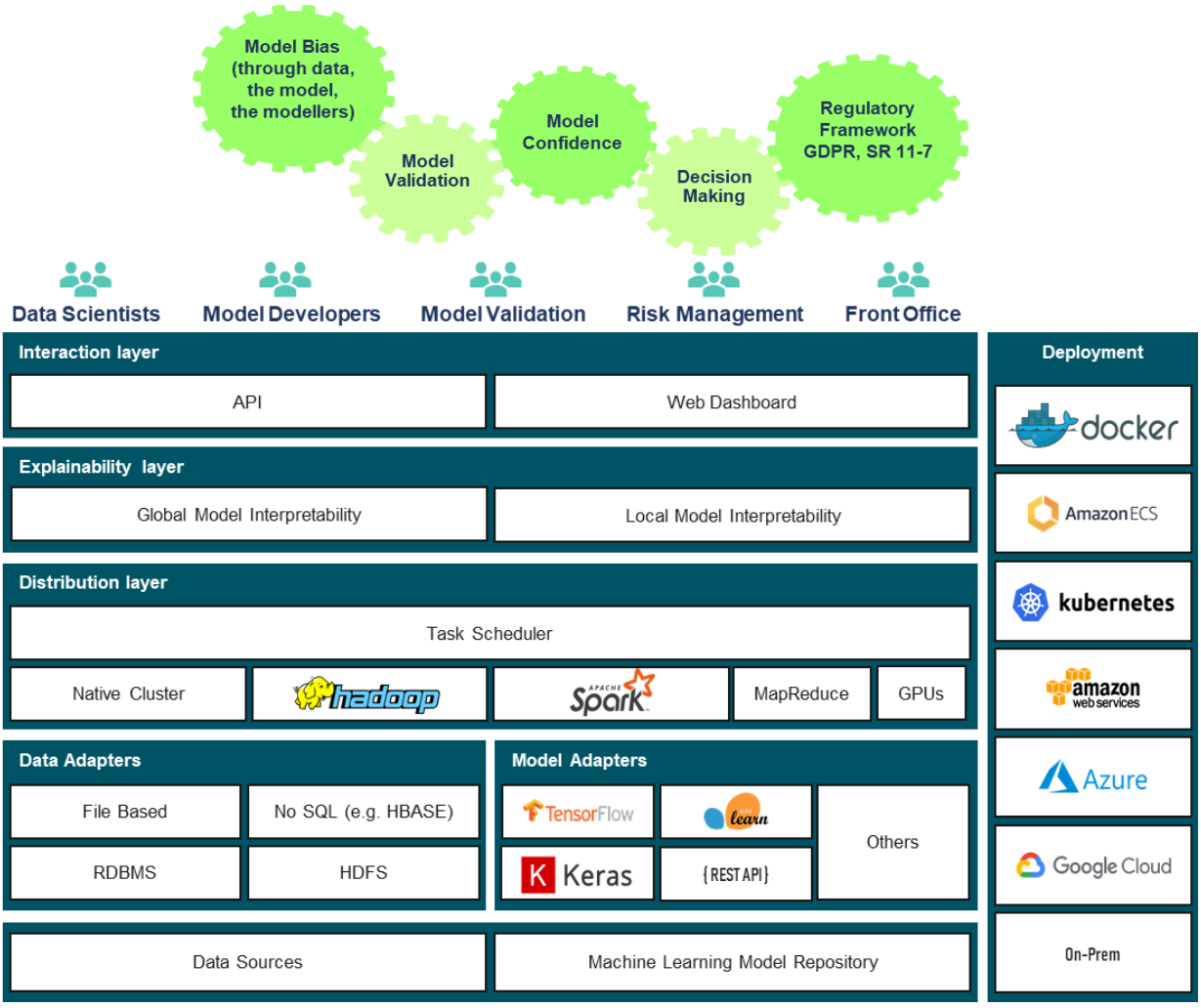
We can use the gradient descent algorithm that we saw earlier to find the combination of weights and bias that minimises E in equation (11).

ANNs in Finance

Delta Capita has produced an ANN that predicts whether a loan application will result in a default or fully paid. This ANN has 2 hidden layers of 100 neurons each and takes 133 input features. It can predict whether a loan will result in default or fully paid with an accuracy of 98.3%.

The size of these ANNs bring complications; if we were to use our loan default model to decide whether to accept or reject a customer, how do we explain our decision to them? Under new data protection laws, the applicant has a right to an explanation. This is the challenge that DC MINT solves; it provides a model explainability and interpretability layer on top of existing machine learning models, to explain a decision that was made. But this problem is recursive… how do you explain the explainer?

DC MINT



Toby Alfred-Jones, Quantitative Consultant

Quantitative consultant specialising in functional analysis with knowledge of stochastic calculus, mathematical finance and Fourier analysis.

Other contributors:
Sylvia Smit, Managing Partner
Ricardo Cruz, Senior Consultant
Khrystyna Andronova, Senior Consultant

Please contact Delta Capita at *capitalmarketsdelivery@deltacapita.com* if you are interested in any of the following areas:

- **DC MINT - Model Interpretability**
- **DC COMPRESSION - Model Compression and Model Optimisation**
- **DC DOCS - Automated Model Documentation**
- **DC VOICE - Voice/NLP on large data sets and Wealth applications**
- **DC RISK - Contagion Risk Models**

DC MINT