# Applications and Development of Machine Learning

Tarik Onalan

*SCHOOL, PLANET EARTH*

**In the last decade, machine learning has taken centre stage in the computing field. With the advent of increasingly powerful computers, giga-, tera-, and even peta-scale computations have become possible. In this paper, we will investigate the applications of machine learning in these large-scale computations—big data—and how they can affect future development.**

## 1. INTRODUCTION

The term "Machine Learning" was coined in 1959 by Arthur Samuel, in his paper *Some studies in machine learning using the game of checkers*, described as "a field of study that gives computers the ability to learn without being explicitly programmed". The definition still holds, but machine learning, as a science, is now leaps and bounds ahead of where it was in 1959. Computers are now capable of analyzing giga- and tera-scale sets of data in hours or minutes instead of days, weeks, or even months or years.

It is essentially impossible to understand big data without machine learning. The computational power required to parse and analyze modern data-sets is well beyond the power of all human minds ever to be created. With computers, however, we can run calculations faster and with higher precision than any human ever could.

The applications of machine learning are vast, spanning from health care, to finance, to social media. Anywhere where there is a large pool of data to be analyzed, machine learning becomes a viable option.

## 2. LEARNING METHODS

### 2.1. Supervised Learning

Supervised learning is achieved by training a model by optimizing it to fit a set of input-output pairs. Given a set of inputs and outputs $\{(x_1, y_1), ..., (x_n, y_n)\}$, the model that is being trained will seek a function $h : X \to Y$, where $X$ is the input and $Y$ is the output. The key fact to remember with this learning method is that there is an expected output for every input against which the model's predictions can be compared and tuned.

### 2.2. Unsupervised Learning

Unsupervised learning is, as the name suggests, the opposite of supervised learning. Instead of constructing a model from input-output pairs, unsupervised learning consists of constructing a model solely from an input set. Essentially, given unlabeled data, the objective in unsupervised learning is to find any hidden structures that may be present in the data, and use those hidden structures to construct a classification or regression model.

### 2.3. Reinforcement Learning

Reinforcement learning functions in the following manner: model is presented with an input space, and a function to calculate reward; however, in this case, there are no explicit input-output pairs, and if the model begins to stray to a sub-optimal path, there will be no outright correction of the digression. This method of learning is preferred for on-line (streamed) learning, as a locked dataset is not required for reinforcement learning. Instead, there must be a balance between exploration of new knowledge and exploitation of old knowledge.

## 3. ALGORITHMS

### 3.1. *k*-NN

*k*-nearest neighbors, commonly shortened as *k*-NN, is a popular machine learning algorithm because of its relative simplicity to implement. It is non-parametric—meaning it does not assume that the input data has any particular structure, making it an ideal tool for "testing" the structure of a dataset and identifying any basic patterns. This property also makes it perform well in unsupervised learning (2.2).

When classifying an input $x_n$, for example, it takes the "majority vote" of the $k$ nearest neighbors to the input $x_n$ to determine the class of the input.

$$x_\theta = \underset{\theta}{\operatorname{argmax}} |k_\theta \in \{k^*\}| \tag{3.1.1}$$

Equation (3.1.1) demonstrates the "majority vote" method. The class of an input vector $x$, represented here by $x_\theta$ is defined as the class—$\theta$—for which the number of nearest neighbors $k_\theta$ in in all of the possible nearest neighbors $\{k^*\}$ is maximized.

Variations of the algorithm also vary the "weight" of the vote with the inverse of the distance. This helps counteract any bias incurred by skewed data; for example, if a point were to have three neighbors, one very near and two very far, the "weighted vote" would ideally have the nearer neighbor have more influence over the class of the input vector than the further neighbors.

$$x_\theta = \operatorname*{argmin}_\theta \frac{\sum_n \text{dist}^m(k_n \in \{k^*_\theta\})}{|k_\theta \in \{k^*\}|} \tag{3.1.2}$$

Equation (3.1.2) demonstrates the "weighted vote" method. The class of an input vector $x$, again represented by $x_\theta$, is defined by the $\theta$ for which the sums of the distances between the $k$ nearest neighbors of class $\theta$ divided by the cardinality of the set $k_\theta \in \{k^*\}$ is minimized. Important to note is that the function dist is taken to the power of $m$; the weighting need not be inverse-linear: it could be inverse-square—arguably the most popular, inverse-cubic, inverse-quartic, et cetera.

### 3.2. Support Vector Machines

Support vector machines are supervised learning models (2.1) that are commonly used for classification and regression tasks. With a linear kernel (which is all we will go over, for the sake of simplicity), a support vector machine will fit a linear model to separate a dataset of two classes (while more than two can be classified, it requires extra computation).

A linear model for a support vector machine is as follows:

$$\omega \cdot x_i + b = y_i \tag{3.2.1}$$

where $\omega$ is the matrix of weights, $x_i$ is the input vector, $b$ is the bias, and $y_i$ is the class. The support vector machine finds the parameters $\omega$ and $b$ for the hyperplane that separate the input dataset into two classes: $\{-1, 1\}$. This is accomplished by maximizing the distance to elements of either class on both sides of the hyperplane. Maximizing the distance is accomplished with the aid of two solutions to the hyperplane defined in equation (3.2.1):

$$\begin{cases} \omega \cdot x_i + b \geq 1 & \text{Class above line} \\ \omega \cdot x_i + b \leq -1 & \text{Class below line} \end{cases} \tag{3.2.2}$$

As $y_i$ is always either $-1$ or $1$, we can simplify this further:

$$y_i(\omega \cdot x_i + b) \geq 1 \quad \forall i \in [1, n] \tag{3.2.3}$$

Finally we can take the equations in (3.2.2) to derive our objective function:

$$\begin{cases} \omega \cdot x_i + b = 1 \\ \omega \cdot x_j + b = -1 \end{cases}$$

What we are assuming here is that $x_i$ represents the element of class 1 nearest to our line, and that $x_j$ represents the element of class $-1$ nearest to our line.

$$\omega \cdot (x_i - x_j) = 2$$

$$(x_i - x_j) = \frac{2}{\parallel \omega \parallel} \tag{3.2.4}$$

Now, if we remember, we were trying to maximize the difference between the elements of the respective classes $\{-1, 1\}$ that were closest to our line; we can use equation (3.2.4) to define our objective function with respect to $\omega$:

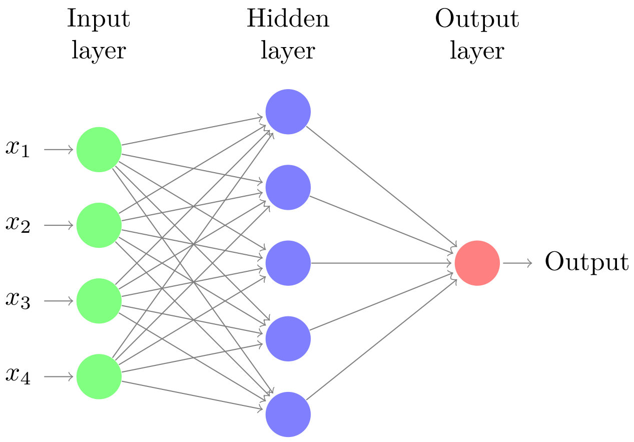
$$argmax_{(i,j)} x_i - x_j$$

$$\operatorname*{argmin}_\omega \frac{2}{\parallel \omega \parallel} \tag{3.2.5}$$

With the objective function, we can now solve for $\omega$ by putting equation (3.2.5) under the constraint of equation (3.2.3), and solving for the weights.

### 3.3. Neural Network

Neural networks are arguably the most publicized of the machine learning algorithms, likely due to their functionality. In essence, they function the same way a small brain would, weighing inputs and providing an output based on a threshold function. Each perceptron—the neural network's equivalent of a neuron—weighs one metric, and passes on the value to the next perceptron.



Neural networks are rather popular in practical use as well, mostly because they are able to relatively accurately map unknown functions of multiple parameters. Additionally, they are capable of learning with all three learning methods shown in section (2). This makes them a versatile tool in any data scientist's toolbox.

Let us do the derivation for a simple single hidden layer perceptron network with backpropagation. The first step is to determine an activation function; we will choose the sigmoid function:

$$\varphi(x) = \frac{1}{1+e^{x}} \tag{3.3.1}$$

Now that we have an activation function, we can define the system of equations describing a neural network with one hidden layer:

$$\begin{cases} S_i(x) = \sum_n \varphi(\omega_{ni} x_n) & \text{Input layer} \\ S_j(x) = \sum_n \varphi(\omega_{nj} S_i(x)) & \text{Hidden layer} \\ S_k(x) = \omega_k S_j(x) & \text{Output Layer (bias)} \end{cases} \tag{3.3.2}$$

From there, we can describe our error function:

$$E(x_i^*) = \sum_{x_i \in x_i^*} (S_k(x_i) - y_i)^2 \tag{3.3.3}$$

The error function described in equation (3.3.3) can then be used to update the individual weights

$$\begin{cases} \omega_i = \omega_i - \alpha \frac{\partial E}{\partial \omega_i} \\ \omega_j = \omega_j - \alpha \frac{\partial E}{\partial \omega_j} \\ \omega_k = \omega_k - \alpha \frac{\partial E}{\partial \omega_k} \end{cases} \tag{3.3.4}$$

where $\alpha \in (0, 1]$, usually between 0.02 and 0.05, is used to control the speed of the gradient descent.

## 4. FINANCE

Machine learning—specifically, neural networks—are heavily used in the field of finance, as they are exceedingly good at mapping unknown functions of multiple variables (3.3). A neural network could predict the rise and fall of certain stocks given a large feature vector containing, say, the profit margin of a company, the national inflation rate, the number of jobs created the previous quarter, et cetera.

Supercomputers are now used to calculate the trends in stocks on-line, given not only quantitative data like that described in the paragraph above, but qualitative data gathered from news sources like sentiment analyses or possible product launches, among others.

## 5. HEALTH CARE

Machine learning in the health care sector is centred more around clustering; support vector machines (3.2), decision trees (not derived), k-nearest neighbor (3.1), and k-means (not derived) are some of the more popular algorithms in use in the computational biology fields, specifically. With the advent of genome research, scientists have to search for genetic anomalies that could be signs of future disease. Clustering algorithms like k-NN are useful in this application because the computational power required is not as demanding as an unsupervised neural network, while they are still able to identify any important structures in the data.

## 6. SOCIAL MEDIA

Machine learning in social media is a trend that has developed fairly recently. The common algorithms in use in social media—for the purpose of news displays, post displays, et cetera—are neural networks, support vector machines, and the like. Algorithms that can map complex functions implicitly are the preferred models in this application, as encoding, say, a sentence with the proper information to make it useful for a computer requires a rather large feature vector. Neural nets and support vector machines, while not the only method of handling large feature dimensionality, are effective in simple implementations.

## 7. CONCLUSION

Hello, world!