# Machine Learning Algorithms Illustrated in Python

Cameron P. Dart

February 21, 2016

**Abstract**

In order to gain a deeper understanding of artificial intelligence algorithms, I write on commonly used supervised and unsupervised learning algorithms. While researching these algorithms, I will create my own implementations of each in Python, describe and prove the mathematical reasoning, and analyze the space-time complexities of my own to open source implementations of these same algorithms in Big $\mathcal{O}$, $\theta$, and $\Omega$ notation.

## 1 Introduction

In this paper, I will strictly define machine learning and give mathematical descriptions and intuition on different subsets of algorithms that exist inside of the domain. Additionally, I will provide an explanation of the differences between Big $\mathcal{O}(n)$,$\Theta(n)$, and $\Omega(n)$ in regards to what they mean in run time and space complexity. Provide valuable insight on how we use these notations to analyze algorithms for their space and time complexity. Lastly, I will combine the mathematical descriptions of algorithms and space/time complexity to analyze the differences between algorithms I have written to open source implementations in order to see how either I can be more efficient, or how to contribute to the implementations' efficiencies. Due to the nature of these algorithms being heavy in mathematics and computation, I believe it would be beneficial to first look at the algorithm from a purely mathematical standpoint before writing any code. Run through the computations by hand a few times and pinpoint exactly what parts in the algorithm could potentially be inefficient, or if it works well, deduce what makes it so quick. After this analysis, I will move forward to writing my own algorithm in Python.

## 2 Method of Analysis

In order to get a better understanding of the algorithms used, I will use modules built into Python such as *memory* and *os*, and *resource* [2] to gather data on programs, functions, and processes run time and memory usage. I will provide line by line theoretical analysis with run times of functions and operations provided by Python documentation so I can further see the short comings and advantages of each implementation. Additionally, theory can only get you so far, I will run test cases with controlled input size and examine the results to confirm my hypothesizes. With all of the theoretical and experimental data at my finger tips I will be able to decide more formally which implementation is scientifically and mathematically proven to be the most efficient.

## 3 Space-Complexity Analysis

### 3.1 Big $\mathcal{O}(n)$

**Formal Definition** Let $C$ and $n_0$ be positive constants
$f(n) \leq Cg(n)$ whenever $n > n_0$
$f(n)$ is $\mathcal{O}(g(n)) \equiv \ \exists C \ \exists n_0 \ \forall n \ (n > n_0 \rightarrow f(n) \leq Cg(n))$ [1]

**Informal Definition** A function $f$ is considered to be $(O)g$ if it is never larger than some constant multiplied by $g$
In short, there exists an upper bound on $f$.

### 3.2 Big $\Omega(n)$

**Formal Definition** Let $C$ and $n_0$ be positive constants
$f(n)$ is $\Theta(g(n)) \equiv \ \exists C \ \exists n_0 \ \forall n \ (n > n_0 \rightarrow f(n) \geq Cg(n))$ [1]

**Informal Definition** A function $f$ is considered to be $\Omega(n)$ if it is always larger than some constant multiplied by $g$

### 3.3 Big $\Theta(n)$

**Formal Definition** Let there exists positive constants $c_1$, $c_2$ , $n_0$.
the function $f$ is considered to be $\Theta(g(n)) \iff c_1 g(n) \leq f(n) \leq c_2 g(n)$ [1]

**Informal Definition** A function $f$ is considered to be $\Theta(g)$ if it is bounded by two separate functions.

## 4 Machine Learning

The field of study that gives computers the ability to learn without being explicitly programmed. These types of algorithms are deeply rooted in mathematical functions and theory. The computational power of a computer greatly exceeds that of a human. Allowing them to process multitudes more data than humanly possible. Machine learning algorithms are being used by a wide variety companies such as Google, Tesla, Goldman Sach's, and Amazon.

## 5 Supervised Learning Algorithms

### 5.1 Definition

Supervised Learning is the task of deducing a function from a *training set.* [4] A *training set* is a vector that is used for the initial discovery of relationships between variables; to fit the weights of the classifier. In order to prevent overfitting, a *validation set* is used in case any classification parameter needs to be adjusted. Further on, a *test set* is used to gauge the efficiency of a given model. One classic example of a supervised learning algorithm is a *linear regression*.

## 5.2 Example: Linear Regression

### 5.2.1 Definition

A *regression* is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable it is called a *linear regression*.

### 5.2.2 Mathematical Description

The *regression line* can be determined by finding the minimum *sum of squares* (SSE) error. a *linear regression line* is $y = mx + b$ where $n$ is the number of data points in the *training set*, $m$ and $b$ are computed values such that

$sx = \Sigma x = x_1 + x_2 + ... + x_n$
$sy = \Sigma y = y_1 + y_2 + ... + y_n$
$sxy = \Sigma xy = x_1 y_1 + x_2 y_2 + ... + x_n y_n$
$sxsq = (\Sigma x)^2 = \Sigma x * \Sigma x$
$m = \left( \frac{n(sxy) - (sx)(sy)}{n(sxsq) - (sxsq)} \right)$
$b = \left( \frac{sy - m(sx)}{n} \right)$ [3]

### 5.2.3 Example Use Cases

You are a home owner and you are trying to decide when you should best sell your home and for what price.
Let $x$ be the time in years and $y$ be the price of your house. Given you want to sell your house at a certain time $x$, estimate how much you might get for it given a vector containing its value over the past 20 years in yearly increments.

# 6 Unsupervised Learning Algorithms

## 6.1 Definition

## 6.2 Example K-Means Clustering

### 6.2.1 Defintion

### 6.2.2 Mathematical Description

## References

[1] https://www.cs.auckland.ac.nz/courses/compsci220s1t/lectures/lecturenotes/GG-lectures/220handout-lecture03.pdf

[2] https://wiki.python.org/moin/TimeComplexity

[3] http://www2.isye.gatech.edu/ sman/courses/6739/SimpleLinearRegression.pdf

[4]