# Regression I

*Part of the Smartia Machine Learning (ML) Essentials series*

### *Introduction to Smartia ML Essentials*

Machine learning is transforming the world we live in. Coverage of intelligent algorithms that will be able to drive us to work, fly us on holiday, diagnose us when we're ill - the list goes on - appear in our news feeds ever more frequently (machine learning drives those news feeds as well by the way!). We are swimming in a sea of data and the means to use that data for generating insights and understanding are becoming more and more sophisticated.

This field of machine learning (data analytics, artificial intelligence or whatever term you would like to call it) is one of the most interesting technology areas right now; and will continue to be so for the foreseeable future. However, for those just starting out in the field it can be overwhelming and difficult to know where to start. This combined with the pressures of everyday life means that some people simply don't get started. That's where we hope these posts will help.

Using relatable examples and plain language, each post will introduce key concepts in machine learning. We hope to provide the starting points that will encourage the reader to learn more and start using machine learning in their own applications (professional or hobby).

### *Some Terms*

Like with any discipline, the world of machine learning has its own vocabulary that practitioners in the field are accustomed to using. It will be useful to have an understanding of a few terms before we proceed:

- **Algorithm**: An algorithm is a process that carries out calculations in a defined order. In machine learning linear regression, logistic regression, random decision forests and k-nearest neighbours are just some of the algorithms that have been developed. Selecting the right algorithm for a given problem is a key skill for a data scientist.
- **Model**: The goal of machine learning is to produce insight into data. This often means that we want to predict some value (e.g. the number of sales a business will make in a month) or assign a category (e.g. classifying what objects are in an image) for new data that we have not seen before. These predictions are done with a model which takes data as an input (an image in the case of an image classifier) and outputs a prediction.
- **Training**: An algorithm on its own has no use for generating insight into our data. We must implement a training process to fit a model to our data (or usually a subset of it that we call the training set).

We can draw an analogy with baking a cake. The algorithm is our recipe, the data is the ingredients, the preparation and cooking would be the training process, and the model is the cake. We can see that using a different recipe will produce a different cake. Similarly, using better ingredients may produce a better cake.

## *Introduction to Regression*

Regression techniques are a set of well-established algorithms for finding the relationship between some input variables (usually called the independent variables) and some output variables (usually called the dependent variables).

There are a range of different regression types each one suited to different scenarios. In today's post we will start by giving a gentle introduction to simple linear regression.

## *A Simple Problem*

We will build towards the algorithm itself by starting with a problem; it's a bit of a contrived example but it should be fine for our purposes.

Let's say we have a sensor that measures force. The sensor outputs a voltage which varies with the applied force. We have long since lost the sensor's calibration certificate and cannot find the manufacturer datasheet anywhere online. With no way of making sense of the voltage readings the sensor is pretty much useless.

In an attempt to save the sensor from the dustbin we decide to run an experiment where we place known loads on the sensor and measure the voltage reading. We hope to find some kind of rule that will enable us to infer force from a measured voltage. The results of this experiment are tabulated below:

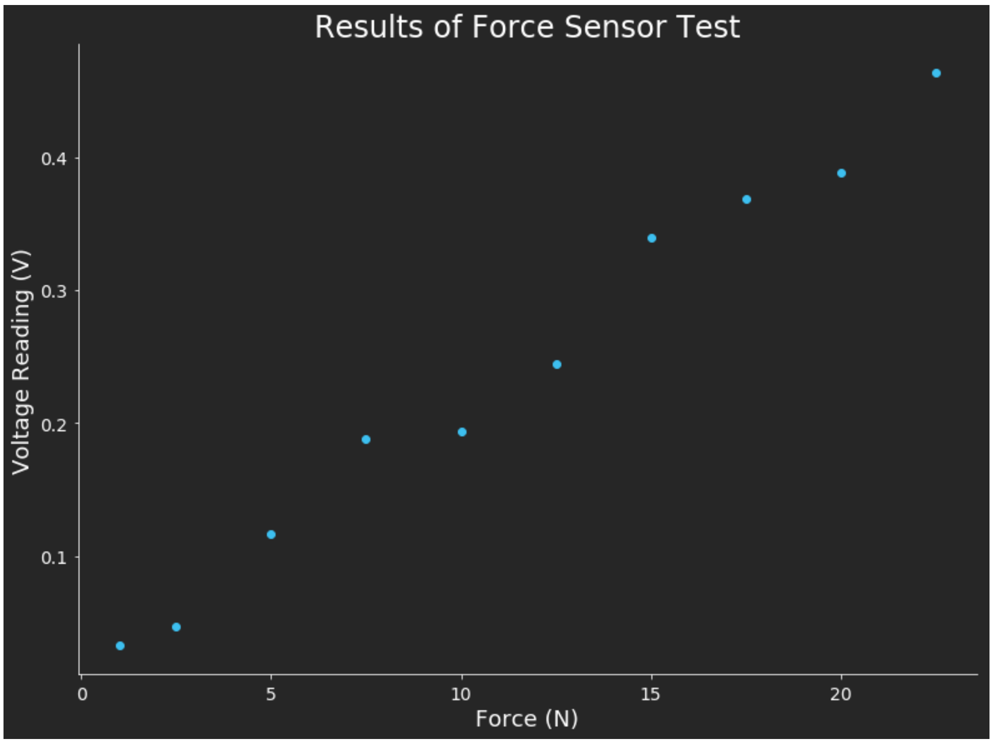
| Force Applied (N) | Measured Voltage (V) |
|---|---|
| 1 | 0.032 |
| 2.5 | 0.047 |
| 5 | 0.116 |
| 7.5 | 0.188 |
| 10 | 0.194 |
| 12.5 | 0.244 |
| 15 | 0.339 |
| 17.5 | 0.369 |
| 20 | 0.388 |
| 22.5 | 0.464 |

Can we make sense of this data and save the force sensor? The rest of this post will step through a process for solving this problem.

## *First Look*

In any machine learning problem, it's a good idea to visualise the data if possible. By doing this we are able to get to grips with the data we have and potentially spot any trends or problems. It also helps with deciding what algorithms might be most suitable.

In our simple sensor example the data can be completely represented in a single two-dimensional plot. The plot below shows applied force on the x-axis against measured voltage on the y-axis.

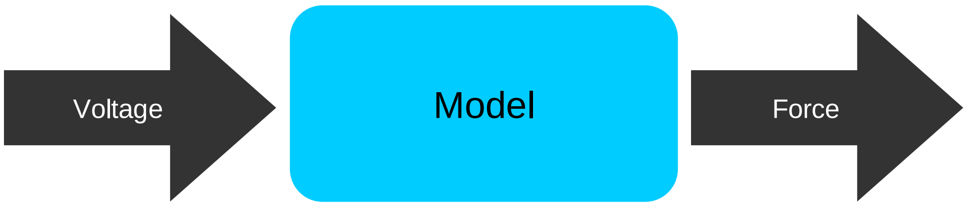


Just by looking at this data we can see that there is some kind of relationship between the force we apply and the measured voltage outputted by the sensor. Specifically, it seems that as the force increases the sensor outputs a higher voltage. We call this a positive correlation; the voltage is positively correlated with the applied force. The existence of a relationship is good news for saving the sensor!

All we can conclude from this initial examination is that there is a positive correlation between voltage and force. In practice we must resist the temptation to go further and say, for example, that the data is best described by a line. However, for the purposes of this article where we just want to introduce some key concepts, we will assume that a straight line is the best fit for the data.

## *Making the Model*

Our first look at the data has exposed a potential link between voltage (the dependent variable) and the force applied (the independent variable). To turn this observation into something useful we will create a model for predicting applied force from a given voltage reading. The figure below illustrates what we want to be able to do.



Going back to our definitions from before we need three things to make a model:

- **Data**: Collected from the force-voltage measuring experiment as explained above.
- **Algorithm**: This will give us the blueprint or recipe for making the model.
- **Training**: The process by which we arrive at the model.

## *The Recipe*

As we've said before, the selection of the correct recipe or algorithm for a given problem is a key skill for a data scientist. Choosing an algorithm will depend on many factors from the nature of the problem (e.g. is it a classification, a continuous prediction problem etc.) to the type of data (time series, images etc.). In this first post we will ignore this selection process and just choose simple linear regression.

Let's break this down by word:

- **Regression**: As said above, regression is a tool (or family of tools) that attempts to find the relationship between variables. This relationship (or model) can be expressed in the form of an equation containing terms that contribute to the value of a target or dependent variable.
- **Linear**: Refers to the way we combine terms in the model equation to give a prediction. Namely, by using addition and subtraction.
- **Simple**: regression can handle multiple independent variables. Simple linear regression is a special case where there is only one independent variable.

## *Introducing the Model*

A simple linear regression model can be represented by an equation. If we call the independent variable $x$ and the dependent variable $y$ we can write:

$$y = a_0 + a_1 x$$

Where $a_0$ and $a_1$ are the model parameters.

For our force sensor example then, simple linear regression will yield a model in the form:

$$V = a_0 + a_1 F$$

Where $V$ is the measured voltage, $F$ is the applied force.

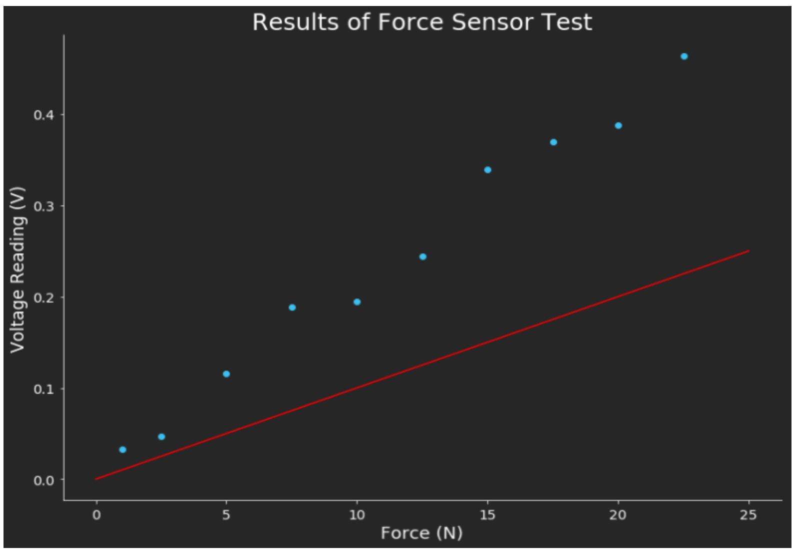
This equation may look very familiar as the equation for a straight line! Two parameters are required to define a straight line; the intercept on the y-axis (here called $a_0$) and the gradient (here called $a_1$) which is a measure of how much the output (the dependent variable) changes in response to a change in input (the independent variable). Initially we don't know what these parameters are. This is where the training process comes in.

***Training***

The process of training aims to find the parameter values that best fit the training data. So, how do we train the model?

In the figure below we've plotted the data together with a line. The red line has been drawn with the parameters $a_0 = 0, a_1 = 0.01$. It is very unlikely that such an initial guess would be a good fit but in machine learning we often need to initialise a model's parameters in order to work towards a solution.

As we can see, the red line doesn't appear to be a good fit. At low values of force the model is reasonably close to the data points but as force increases the line deviates more and more from the bulk of the data points.



Can we do better than this initial guess? The answer, of course, is yes!

The first step towards improving the model is to find some measure of how wrong we are with this model. Let's define the error, $E$ at the $i^{th}$ data point as the difference between the measured value, $V_i$ and the predicted value, $\widehat{V}_i$:

$$E_i = V_i - \widehat{V}_i$$

Where:

$$\widehat{V}_i = a_0 + a_1 F_i$$

If we sum up all the individual errors we will get a value for the total error between the model predictions and the data points. However, what if some of these errors are negative (i.e. $\widehat{V}_i > V_i$ )? We will end up with a lower value than we should as negative and positive errors cancel each other out. There are several ways we could modify our error expression to remove this problem but in our discussions here we will choose to square the errors; ensuring positive values for all individual errors.

Summing the squared errors gives the sum of squared errors, $SSE$:

$$SSE = \sum_{i=1}^{n} (V_i - \widehat{V}_i)^2$$

The equation above simply calculates the error for each data point $(V_i - \widehat{V}_i)^2$, squares it and then sums over the entire training dataset (all $n$ data points).

This quantitative measure for model inaccuracy is usually called the cost function. In this case we have defined a cost function that is the sum of squared errors between the predictions and the data. This is not the only cost function we can use in machine learning and is not even the only cost function we could have used for this simple problem. As we explore machine learning in further posts we will see how other types of cost functions can be used.

The cost function we have defined can be used to drive alterations to the model that will improve its fit to the data. Let's expand the SSE equation by substituting the model equation for $\widehat{V}_i$:

$$SSE = \sum_{i=1}^{n} (V_i - a_o - a_1 F_i)^2$$

This equation has four terms. The terms $V_i$ and $F_i$ represent the $i^{th}$ data point and are constant (the training data does not change). The remaining terms ($a_0$ and $a_1$) are the model parameters, and these are not constant; i.e. varying the model parameters will change the SSE.

Varying the model parameters in such a way that reduces the SSE will improve the fit of the model on the data. To do this we will use a simple algorithm called gradient descent.

Gradient descent is a fundamental algorithm in mathematical optimisation. It's used to find the parameters that minimise a given function. By calculating the gradient of a function at a given point (starting off with whatever initial parameters we specify) it updates the parameter values so that the function evaluates to a lower value than before. Repeating this process (gradient calculation followed by updating parameter values) will allow us to move along the function towards a minimum value.

*Gradient Descent - Step 1*

To implement gradient descent for our problem we will need to calculate the derivative of the SSE with respect to the model parameters $a_0$ and $a_1$. Differentiating the SSE equation with respect to each model parameter in turn gives:

$$\frac{\partial(SSE)}{\partial a_0} = -2\sum_{i=1}^{n} \quad (V_i - a_o - a_1F_i)$$
$$\frac{\partial(SSE)}{\partial a_1} = -2\sum_{i=1}^{n} \quad F_i(V_i - a_o - a_1F_i)$$

Notice that we have two expressions that represent the SSE equation differentiated first with respect to $a_0$ and then with respect to $a_1$. If we had three or four or more parameters in our model we would have a corresponding number of equations for the SSE derivatives.

*Gradient Descent - Step 2*

In the update step we simply subtract the derivatives from the current parameter values. The parameter value at iteration $k+1$ is equal to the parameter value at iteration $k$ minus the derivative multiplied by a small constant, $\alpha$.

$$(a_0)_{k+1} = (a_0)_k - \alpha\frac{\partial(SSE)}{\partial a_0}$$
$$(a_1)_{k+1} = (a_1)_k - \alpha\frac{\partial(SSE)}{\partial a_1}$$

The constant $\alpha$ is usually called the learning rate. It is chosen to be small enough to guarantee numerical stability in finding the minimum. It can be thought of as the step size we take as we traverse the function towards the minimum; too big a step size and we can overshoot the minimum, while too small a step size can unnecessarily delay the algorithm in finding the minimum.

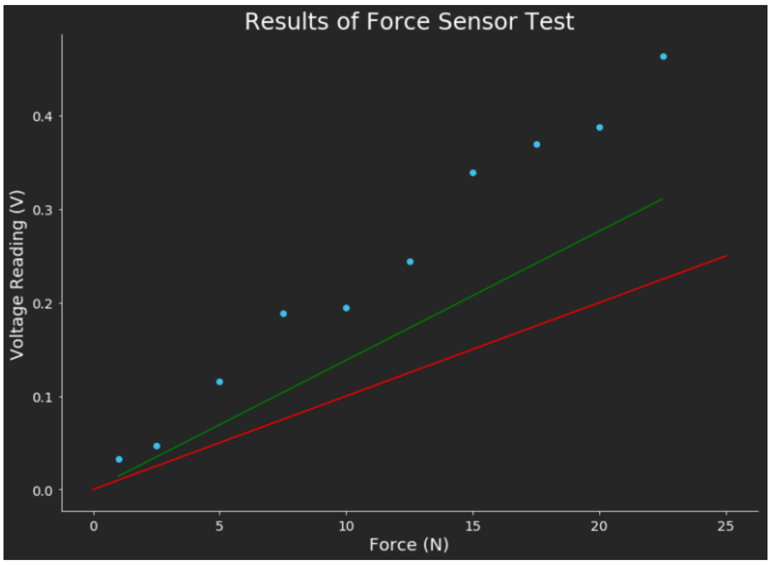
Let's try applying this to our sensor example. At $k = 0$ we choose initial values of:

$$(a_0)_0 = 0.01$$
$$(a_1)_0 = 0.01$$

Setting a learning rate of 0.0001 we now calculate the updated parameters for the next iteration step (i.e. $k = 1$):

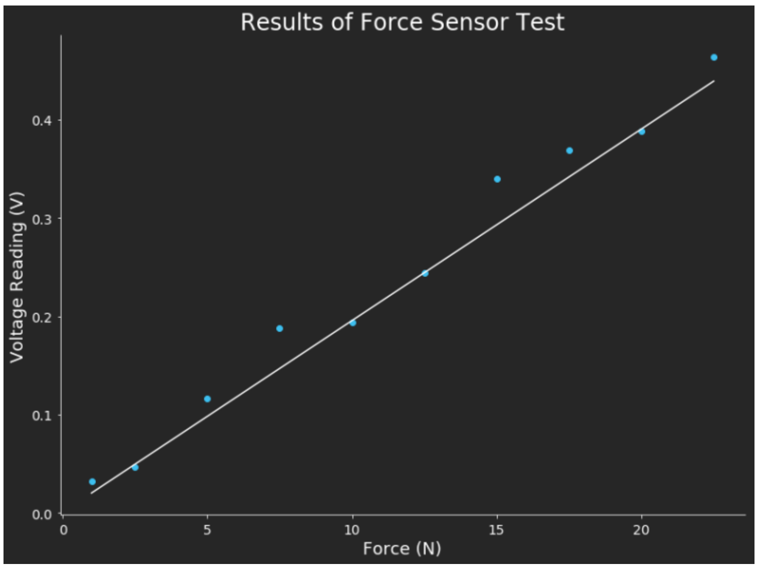
$$(a_0)_1 = 0.01 - \alpha \frac{\partial(SSE)}{\partial a_0}$$
$$(a_1)_1 = 0.01 - \alpha \frac{\partial(SSE)}{\partial a_1}$$

The figure below shows the plot of our data together with the initial model (at $k = 0$, in red) and the first updated model line ($k = 1$, in green).

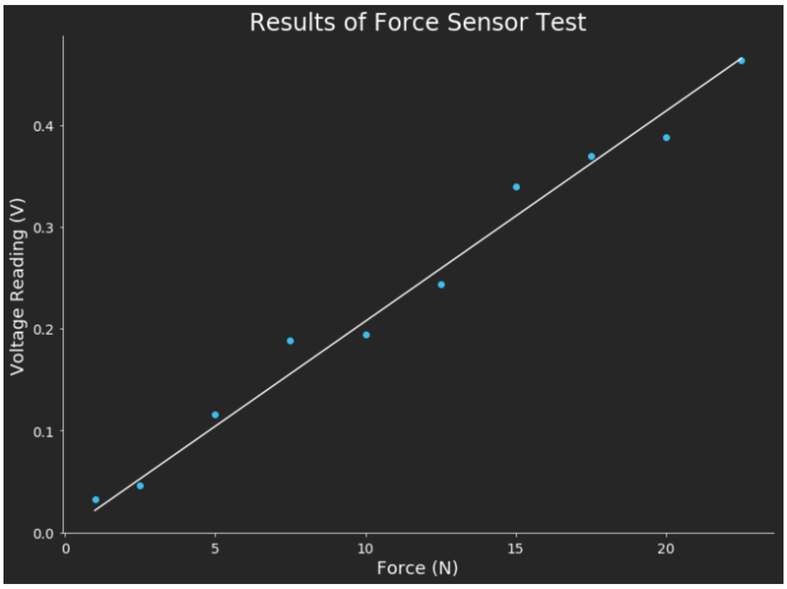


As we can see, the new updated parameters have given us a line that is much closer to the data and looks to be a better fit. It still does not look good enough though. Running more iterations should get us closer.

After five iterations we get the model shown in the figure below:

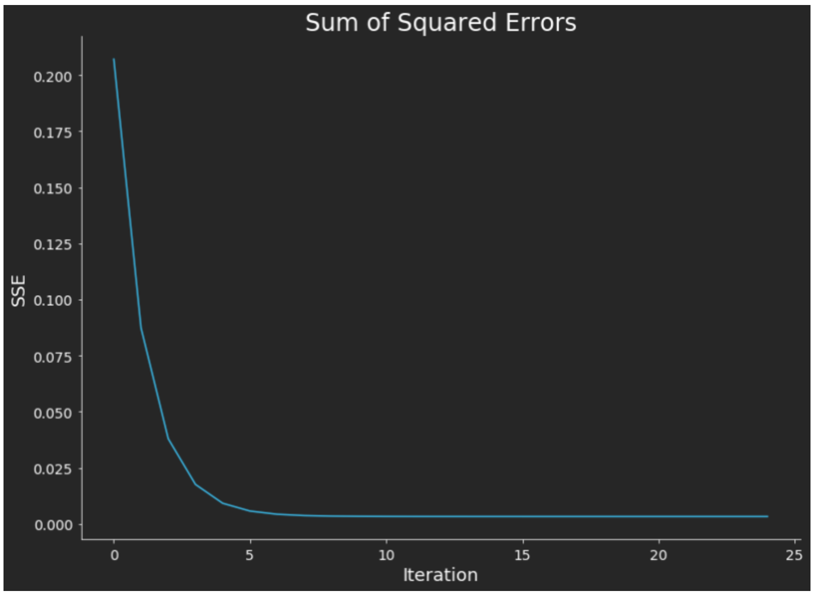


Twenty-five iterations gets:



This looks much better. We now have a line which looks like it could be the line of best fit. Should we keep going with the gradient descent? In this case we can see just by looking at the plot that it might be difficult to get a better fit with this particular approach (simple linear regression) so could make the decision to stop the gradient descent process and use the model parameters found so far.

Deciding based on just an observation of where the line is doesn't seem to be a very good approach. What is needed is a more robust way of deciding whether to keep going with gradient descent process or to stop. One possibility is to monitor the SSE as we iterate; this is shown in the figure below.



We can see that at a certain point the error stops reducing with extra iterations. Beyond about 10 iterations the reduction in error is so small that it's not visible on the above graph. With this numerical measure we are now in a better position to decide when to stop the gradient descent. All we need to do is have an exit condition in our iteration loop that exits gradient descent when the rate of reduction in SSE reaches some low value.

## ***The Result***

After running the gradient descent process we are left with two parameters for a simple linear regression model that best fits our data. The final model parameters are:

$$a_0 = 0.0008$$
$$a_1 = 0.0206$$

Substituting these values in to the model equation from before gets the following equation for voltage in terms of force:

$$V = 0.008 + 0.0206F$$

Remember that our sensor gives out voltages and we want to be able to infer what force caused that voltage. The equation above is not suitable for this. Rearranging the equation so that it is in terms of $V$ we get:

$$F = \frac{V - 0.008}{0.0206}$$

That's it! With this equation we can take a measured voltage and predict the applied force.

## *It's only the beginning*

Simple linear regression is a useful tool for very straightforward problems and is a great entry point to the world of machine learning. We have introduced some concepts, like cost functions and gradient descent, that are fundamental to machine learning and can be found in one form or another in all sorts of algorithms.

There is, however, much that we have not discussed including alternative algorithms, different cost functions, and performance metrics. We will save these and more for future posts; there's much to talk about but this has hopefully been useful to whet the appetite for some more machine learning!