Further notes on regression

Bernard Butler and Kieran Murphy

2026-02-12

Contents

1 Background

Regression is all about predicting a fitted value (a number, usually represented as $\hat{y}$) for a given set of features $p(x)$.

The simplest case is fitting a line to data, in which case $p(x) = \{1, x\}$.

The equation of a line has two parameters: the *slope* and the *intercept.* Although it can take time to become familiar with labels and symbols, they are very handy in machine learning. The labels we use in regression are β_0 (for the intercept term) and β_1 (for the slope term).

2 Training

For x and $\hat{y}$ to be on the line with intercept β_0 and slope β_1, we must have

$$\hat{y} = \beta_0 + \beta_1 x \tag{1}$$

This is the old $y = mx + c$ formula that students first met in Junior Cert/O Level Maths, recast in terms of a vector β instead of m and c... For the training set, we generally have many points (x_i, y_i), where i is an index that ranges from 1 to n.

Typically, in mathematical notation we write x_i but in Python we would write $x[i]$. They both represent the same data value.

We want

$$\hat{y}_i = \beta_0 + \beta_1 x_i. \tag{2}$$

In other words, all the points $(x_i, \hat{y}_i)$ should lie on the fitted line.

However, we do not know $\hat{y}_i$ yet, so we approximate it with the y_i we were given in the training set. Thus we have

$$y_i = \beta_0 + \beta_1 x_i + e_i, \tag{3}$$

where e_i represents the error term (residual of the fit: $e_i \equiv y_i - \hat{y}_i$).

If we write the equations explicitly, we find

$$\begin{aligned} y_1 &= 1 \times \beta_0 + x_1\beta_1 + e_1 \\ y_2 &= 1 \times \beta_0 + x_2\beta_1 + e_2 \\ \ldots &= \ldots \\ y_n &= 1 \times \beta_0 + x_n\beta_1 + e_n \end{aligned} \tag{4}$$

Hopefully students can see a pattern emerging:

1. we always multiply β_0 by 1;
2. we always multiply β_1 by x_i;
3. we always add the result to the error term e_i.

We write

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdot\cdot & \cdot\cdot\cdot \\ 1 & x_n \end{pmatrix}, \tag{5}$$

where X is a matrix with those rows, and we write

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}, \tag{6}$$

where $\boldsymbol{\beta}$ is a single column vector with those values, and

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \cdot\cdot \\ y_n \end{pmatrix}, \tag{7}$$

and

$$\boldsymbol{e} = \begin{pmatrix} e_1 \\ e_2 \\ \cdot\cdot \\ e_n \end{pmatrix}, \tag{8}$$

are column vectors, the first being the y values from the training set and the second the computed errors for the fit. We can rewrite the point-by-point equations 3 or 4 above, cleanly, in matrix-vector form as

$$\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{e} \tag{9}$$

This is based on how matrix-vector multiplication works. Students are advised to try this themselves using pencil and paper.

3 Analysis

Note that $\boldsymbol{y}$ and X come directly from the training set. $\boldsymbol{\beta}$ is the parameter vector that we are looking for, so that we can predict $\hat{y}$ for any x (see Equation 1) in the training or test set.

Note that X generally has more rows than columns, which is why it is *overdetermined.*

If there are fewer rows than columns (X is *underdetermined*), we cannot solve for a *unique* slope and intercept. That makes sense: if we wish to fit a line to just one point, there are many lines that pass through that point.

If there are exactly the same rows and columns (2), and the 2 points do not coincide, there is a single line that passes through both points. More generally, there are many points in the training set and it is impossible to pass through all the points simultaneously, so regression helps us to find the line that is "most representative" of all the data. There is a more formal statement of this, but you can imagine that the fitted line is the one that

- averages out the errors (so the mean of $\boldsymbol{e}$ is zero), and
- minimises the "size" of $\boldsymbol{e}$. Usually, the size of $\boldsymbol{e}$ is represented by its Euclidean norm (square root of the sum of the squares of e_i, when $i = 1, \ldots, n$).

The Normal equations show how $\boldsymbol{\beta}$ can be derived from X and $\boldsymbol{y}$, on the assumption that the size of $\boldsymbol{e}$ is represented by its Euclidean norm. Other measures of distance can be used instead, but then the Normal equations do not apply and we need to use different methods to solve for $\boldsymbol{\beta}$.

To solve for $\boldsymbol{\beta}$ using python, you generally supply the X and $\boldsymbol{y}$ from the training set and let the `scikit-learn` or `statsmodels` solver (whichever one you chose) do the rest. You can then take that object and use it to predict $\hat{y}$ for any instance in the test set. This is known as the *prediction* phase, which follows the *training* phase of supervised learning techniques such as linear regression.

Internally, the solver does not use the Normal equations directly. Instead it uses an alternative formulation that is derived from (and equivalent to) the Normal equations, using more advanced linear algebra that we do not cover in this module. The advantage of the alternative formulation is that it is more efficient (faster, uses less memory), more accurate in the presence of measurement and roundoff errors, and can even provide metrics on the collinearity of the features in X and its own precision.

4 Extensions

Lastly, the features $p(x)$ do not have to be restricted to 1 and x. They could be $1, x$ and x^2 say (so you are fitting a quadratic function - a parabola - to data). You can also use higher powers of x if needed to model more curvature.

Likewise, you might wish to fit a plane rather than a line, in which case y in the equations above should be interpreted as the height (usually represented by z) above the $xy-$plane, so you have three features $1, x$ and y where you had just 1 and x for fitting a line. Clearly this could be extended to higher dimensions if needed.

More generally still, the features could be any function of x ($\sin(x)$, $\log(x)$, etc.) and regression can still handle such models, providing they can be written according to the pattern mentioned above, that is, as a linear combination of features.

As more features are added, it is important that each $p_j(x), p_k(x)$ should be non-collinear, so that the corresponding β_j, β_k can be distinguished from each other.

Lastly, the target variable y_i could itself be vector-valued $\boldsymbol{y}_i$, leading to *multivariate linear regression*, which is supported by both statsmodels and scikit-learn.