Regression

John Whitamore

Abstract

Regression models clearly explained with examples in Python.

1 Introduction

This paper develops an understanding of regression - what it is and how it is approached in widely-used modelling frameworks.

Regression is the task of learning a mapping $h : \mathcal{D} \rightarrow \boldsymbol{y}$ from independent data $\mathcal{D}$ to dependent data $\boldsymbol{y}$ in a way that optimises some objective function. Optimising an objective function might involve minimising a loss function, maximising a utility function or - for probabilistic models - maximising likelihood or marginal likelihood.

In practice, the independent data $\mathcal{D}$ might be unstructured, incomplete and not in a form which we want to use directly for modelling. We very often decompose the function h into a data pipeline function g, which transforms our data into a form X that is amenable for modelling and a regression function f which maps onto the observed output data $\boldsymbol{y}$. This yields the compositional structure $g : \mathcal{D} \rightarrow X$, $f : X \rightarrow \boldsymbol{y}$.

Indeed, it is useful to think of a regression pipeline as having three stages. The first stage is responsible for extracting and collecting raw data and transforming it into a shape that is sufficiently clean and structured for the purpose at hand. We can write $g : \mathcal{D} \rightarrow \mathcal{D}^*$ in which $\mathcal{D}$ represents raw data, $\mathcal{D}^*$ represents clean, structured data and g represents the function (or ETL pipeline) responsible for data extraction, cleaning and structuring.

The second stage transforms structured data into the form that we want to use within our regression model. We can write $\phi : \mathcal{D}^* \rightarrow X$. An example might be using scalar x data to create values for $x^0, x^1, x^2, \ldots$ for use in polynomial regression. In practice, this stage involves multiple functions, each transforming structured data into a distinct feature representation. These functions are called *basis functions*.

Finally, the third stage is the regression itself, in which the model learns the mapping $f : X \rightarrow \boldsymbol{y}$. This is where we will focus our attention.

Overall, we have a compositional pipeline $g : \mathcal{D} \rightarrow \mathcal{D}^*$, $\phi : \mathcal{D}^* \rightarrow X$, $f : X \rightarrow \boldsymbol{y}$ that goes from raw to structured data, from structured data to transformed data and - only then - from transformed data to observed outputs. It is clear that the data pipeline mapping(s) g and the transformation function(s) ϕ are at least as important as the regression mapping f. It is, perhaps, also clear that g and ϕ could, in principle, be learned (by optimising an objective function) rather than fixed. This last point is somewhat in the spirit of multi-layer neural network architectures which include both ϕ and f in their processes. A useful task in a world of AI agents would be also to learn and optimise the data pipeline g rather than treating it as fixed.

2 Linear regression

A linear regression model is one in which the function f is linear, so that $f(X) = X\boldsymbol{w}$, where $X \in \mathbb{R}^{N \times M}$ is called the *design matrix* and $\boldsymbol{w} \in \mathbb{R}^M$ is a vector of regression weights (or coefficients). Note that this means that a linear regression model is linear in its parameters, $\boldsymbol{w}$. It does not mean that a linear regression model can only be used to fit straight lines to data.

We will also use $\boldsymbol{\epsilon} \in \mathbb{R}^N$ to denote a vector of residuals $\boldsymbol{\epsilon} \equiv \boldsymbol{y} - X\boldsymbol{w}$ and $\boldsymbol{\theta}$ to denote the parameters of a model.

Before we put our heads down into the details, it is worth looking up to see the bigger picture. We are describing a world in which data pipelines are designed by humans, basis functions and functional forms are carefully selected by a human and the mapping from design matrix X to observed data y is linear. It is very far from a world in which machines are learning or displaying artificial intelligence.

2.1 Design matrix and basis functions

Each column of the design matrix X encodes a hypothesis about the data generating process. The values in each column can be raw data values or - more typically - they can be values obtained by applying basis functions ϕ to raw data values. Columns are often referred to as *features*.

The requirement is (from an interpretative point of view) that the hypotheses are distinct from each other and (from a linear algebra point of view) that the columns of the design matrix are linearly independent. The requirement for linear independence means that the columns of the design matrix form a basis for their span. The term *basis function* reflects the idea that each function contributes to a spanning set over the space of possible input transformations.

Basis functions map a column of (clean, structured) data $\boldsymbol{d}^*$ onto a column of design matrix values $\boldsymbol{x}$ so that we have $\phi : \boldsymbol{d}^* \to \boldsymbol{x}$. Commonly-used basis functions include 1) $\phi(\boldsymbol{d}^*) = \mathbf{1}$ (for bias), 2) $\phi(\boldsymbol{d}^*) = \boldsymbol{x}^k$, 3) trigonometric functions, 4) radial basis functions, 5) one-hot encoding and so on. Although the design matrix enables a linear mapping from features to outputs, it can encode a very wide range of features and behaviours.

The principal drawbacks of the design matrix mechanism are 1) that the mapping from features to outputs is always linear in its parameters, 2) that the mapping is fixed at the time of designing the model and 3) that we often do not know how many hypotheses we should encode in the design matrix. This last point gives rise to the question of sparsity, which we will discuss in detail.

2.2 Probabilistic regression

We have a vector of observed data $\boldsymbol{y}$ of length N and a design matrix of features with N rows. We therefore have N points of observed data $\{y_1, \ldots, y_N\}$ and N rows of the design matrix $\{\boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_N^T\}$ which we can match up with each other to give combined feature and observed data $\{\boldsymbol{x}_n^T, y_n\}$ for row n .

For some probability density or probability mass function p, we have for each row $p(y_n) = p(y_n | \boldsymbol{x}_n^T \boldsymbol{w}, \theta)$ where θ represents the parameters of the probability function. Our choice of a suitable probability density or probability mass function will depend on the data which we are modelling. Common choices include the Poisson distribution for modelling non-negative integers, the Gaussian distribution for modelling real values, the Gamma distribution for modelling strictly positive real values and so on.

It is almost always the case in regression modelling that identically the same probability density or mass function is used for all values of n. It is also commonly assumed that the probabilities for each value of n are independent of each other. These two assumptions taken together are referred to as *identically and independently distributed* or, for short, *i.i.d.*.

Under the i.i.d. assumptions, we can write the joint probability of observing all the y_n values, given the feature values $\boldsymbol{x}_n^T$ and the parameter values $\boldsymbol{\theta}$, as

$$p(\boldsymbol{y}|X, \boldsymbol{w}, \boldsymbol{\theta}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n^T\boldsymbol{w}, \boldsymbol{\theta}) \tag{1}$$

2.3 Maximum likelihood

What we'd like to do is to find regression coefficients $\hat{\boldsymbol{w}}$ and associated parameter values $\hat{\boldsymbol{\theta}}$ that maximise the joint probability of the observed data. The subtlety is that, in the probabilistic formulation, these coefficients are treated as fixed. To optimise them, we require a function that shares the same numerical values as the joint probability for given inputs, but treats the parameters as variables.

This leads us to define the *likelihood function* $\mathcal{L}$, which takes $\boldsymbol{w}$ and $\boldsymbol{\theta}$ as arguments. We write it as $\mathcal{L}(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{\theta})$, and typically seek the values of $\boldsymbol{w}$ and $\boldsymbol{\theta}$ that maximise it.

Note that $\mathcal{L}(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{\theta}) \equiv p(\boldsymbol{y}|X, \boldsymbol{w}, \boldsymbol{\theta})$. Both expressions yield the same numerical value for the same inputs. The distinction lies in interpretation. In the likelihood formulation, the data is fixed and the parameters vary; in the joint probability, the parameters are fixed and the data varies. This reversal is what justifies maximising the likelihood rather than the joint probability.

2.4 Normally distributed residuals

We define *residuals* as the differences between observed values and the values of the fitted model. Selecting a probability distribution is a key choice when building a model. A common choice for real valued observed data is the Gaussian or Normal distribution.

Under the assumption of normally distributed residuals, we can write

$$\begin{aligned}
\mathcal{L}(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{\theta}) &= p(\boldsymbol{y}|X, \boldsymbol{w}, \boldsymbol{\theta}) \\
&= \prod_{n=1}^{N} \mathcal{N}(y_n|\boldsymbol{x}^T\boldsymbol{w}, \sigma^2) \\
&= \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(y_n - \boldsymbol{x}^T\boldsymbol{w})^2}{2\sigma^2}\right)
\end{aligned}$$

We would like to maximise this expression with respect to the regression weights $\boldsymbol{w}$. In order to do this, it is convenient to take logs of both sides. Note that the natural logarithm function increases monotonically, so the location of the maximum is unaffected by taking logs. This gives us an expression for the log likelihood $\ell(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{\theta})$, denoted using a curly lower case ℓ:

$$\begin{aligned}
\ell(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{\theta}) &= \sum_{n=1}^{N} -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y_n - \boldsymbol{x}_n^T\boldsymbol{w})^2 \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \boldsymbol{x}_n^T\boldsymbol{w})^2
\end{aligned}$$

The shape of derivation that we have used is very common across probabilistic models. We have selected a probability mass or density function (as a modelling choice) and used it to write down an expression for the joint probability of observing the data under the model using the i.i.d. assumptions. From there, we have written a likelihood function and then a log likelihood function which we can optimise with respect to the regression coefficients $\boldsymbol{w}$.

Ordinary Least Squares

Inspection of the expression for log likelihood reveals that $\boldsymbol{w}$ appears in only one place. It means that we just need to find the $\boldsymbol{w}$ that minimises the sum of squared residuals, $\sum_{n=1}^{N}(y_n - \boldsymbol{x}^T\boldsymbol{w})^2$. For this reason, a linear regression model with normally distributed residuals is often referred to as the method of *Ordinary Least Squares*.

Defining a vector of residuals $\boldsymbol{\epsilon} \equiv \boldsymbol{y} - X\boldsymbol{w}$, we can write the sum of squared residuals as $\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$

$$\boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w})$$

Differentiating with respect to $\boldsymbol{w}$ and setting the gradient equal to (vector) $\boldsymbol{0}$ yields the maximum likelihood estimate of the regression weights:

$$\begin{aligned}\frac{d\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}}{d\boldsymbol{w}} &= 2X^TX\boldsymbol{w} - 2X^T\boldsymbol{y} = \boldsymbol{0}\\ \hat{\boldsymbol{w}} &= (X^TX)^{-1}X^T\boldsymbol{y}\end{aligned}$$

We then use these maximum likelihood weights to obtain our fitted values, $\hat{y} = X\hat{\boldsymbol{w}}$. The fitted values are conditional expectations: what value of y do we expect given some particular value of $\boldsymbol{x}^T$? It means that we can predict a value of y given a new (out-of-sample) value of $\boldsymbol{x}^T$: $\hat{\boldsymbol{y}}^* = \boldsymbol{x}^{*T}\hat{\boldsymbol{w}}$.

2.5 Geometric interpretation

The mechanism by which OLS regression works is to project N dimensional observation space data into a lower ($M < N$) dimensional feature space in which structure can be identified and then to re-project the low dimensional feature structure back into high (N) dimensional observed data space to obtain N fitted values. Let's take a look at the details in a conventional way. And then let's pursue the details in a somewhat unconventional way to see what useful insights we can obtain.

2.5.1 The conventional interpretation

We have $\hat{\boldsymbol{w}} = (X^TX)^{-1}X^T\boldsymbol{y}$ and $\hat{\boldsymbol{y}} = X\hat{\boldsymbol{w}}$. Putting these together yields

$$\begin{aligned}\hat{\boldsymbol{y}} &= X(X^TX)^{-1}X^T\boldsymbol{y}\\ &= H\boldsymbol{y}\end{aligned}$$

where $H = X(X^TX)^{-1}X^T$ and is called the "hat" matrix because it takes observed $\boldsymbol{y}$ and puts a hat on it to give the fitted values $\hat{\boldsymbol{y}} = H\boldsymbol{y}$. It is made up of an outer product XX^T transformed by a matrix $P = (X^TX)^{-1}$ to give XPX^T. We can interpret the hat matrix by looking at the outer product XX^T and the transformation matrix P separately.

XX^T has dimensions $N \times N$ and operates in observation space and not feature space. Remember that the design matrix X has N rows. The matrix XX^T encodes similarity between pairs of rows of the design matrix, so that element i, j of XX^T encodes the similarity between rows i and j of the design matrix and - to the degree that rows of the design matrix successfully model variation in the observed data - also encodes the similarity between data points.

By contrast, X^TX has dimensions $M \times M$ and operates in the lower dimensional feature space. It encodes how similar columns - or features - of the design matrix are to each other. Its inverse, $P = (X^TX)^{-1}$ represents an operation of correcting for feature covariance, acting as a kind of normalising transformation in feature space.

Putting these pieces together, the hat matrix $H = XPX^T$ represents the pairwise similarity between observations (XX^T) being corrected by $P = (X^TX)^{-1}$ for pairwise similarity between features. In short, we use XX^T to capture similarity in observation space and use $P = (X^TX)^{-1}$ to correct for similarity in feature space, obtaining fitted values that respect both the geometry of the data and the structure of the features.

2.5.2 Going deeper

We earlier derived the OLS maximum likelihood estimator for regression weights $\hat{\boldsymbol{w}}$ as

$$\hat{\boldsymbol{w}} = (X^TX)^{-1}X^T\boldsymbol{y}$$

An obvious problem is that the algorithm will fail if X^TX cannot be inverted. Overt failure is, in a sense, a good type of problem because it is obvious when it has occurred and corrective steps can be taken. A more insidious problem is when the columns - or features - of X are sufficiently different from each other that matrix inversion succeeds - making it look as though the model has "worked" - whilst also being similar enough to each other that the resulting regression weights are unstable and dangerous to use. These two problems are manifestations of *multicollinearity*, meaning that the columns of the design matrix are too closely aligned with each other.

As a thought experiment, let's stipulate that the columns of X must be mutually orthogonal. And - why not? - let's also stipulate that the columns must be normalised to unit length. (Normalising a column simply results in the corresponding regression weight being multiplied by the normalisation term, so the effect comes out in the wash). A matrix with mutually orthogonal columns, each normalised to unit length, is said to have *orthonormal* columns.

If X has orthonormal columns then $X^TX = I_M$, the $M \times M$ identity matrix. If X is square we also have $XX^T = I_N$, the $N \times N$ identity matrix. Let's use these properties in the context of our OLS regression results. We have

$$\begin{aligned}
\hat{\boldsymbol{y}} &= X\hat{\boldsymbol{w}} \\
&= X(X^TX)^{-1}X^T\boldsymbol{y} \\
&= XI_M^{-1}X^T\boldsymbol{y} \\
&= XX^T\boldsymbol{y}
\end{aligned}$$

If X is square then we have $XX^T = I_N$ and we obtain the beautiful but useless result $\hat{\boldsymbol{y}} = \boldsymbol{y}$. This is not a model but a tautology: "If you tell me the true value of y, I'll tell you my estimate of y". This demonstrates that X must not be square. We need to project the observed data into a lower dimensional feature space in order to obtain useful results.

On the other hand, if X is "tall", with the number of rows N being larger than the number of columns M, we have a result which is useful ($\hat{\boldsymbol{y}} = XX^T\boldsymbol{y}$) and in which the matrix X does not appear directly but only as the outer product XX^T. This suggests that we could define a new matrix $K = XX^T$, subject to K having certain required properties, and re-cast our OLS regression simply as $\hat{\boldsymbol{y}} = K\boldsymbol{y}$.

Let's think carefully, because this formulation seems to contain some interesting possibilities. The first is that we might be able to write OLS regression directly in terms of a matrix K, rather than needing to use design matrix X (painstaking work requiring domain knowledge that we might not have). The second is that we might be able to construct links between OLS regression and other widely-used models, particularly non-parametric models. Both of these possibilities excitingly turn out to be true. However, it's also worth noting that we probably wouldn't want to apply this formulation naively: attempting to estimate $N(N+1)/2$ elements of symmetric matrix K is grossly inefficient compared to estimating M regression coefficients in the OLS formulation.

Here, K plays the role previously played by the hat matrix H but with the requirement that X now has orthonormal columns. We have $X^TX = I_M$ so that no feature space correction is made or required because the feature space has orthonormal columns. Nonetheless, we have already seen that we do need to project the observed data into a lower dimensional feature space in order to obtain useful results. Indeed, we require that matrix K is an *orthogonal projection matrix*.

In order to be an orthogonal projection matrix, K must be symmetric (which implies square), positive semi-definite and idempotent (so that $K^2 = XX^TXX^T = XIX^T = XX^T = K$ with $X^TX = I$). We require that the eigenvalues are either 0 or 1, which guarantees idempotency ($K^2 = K$) and implies that the determinant of K is either 0 or 1 depending on rank.

Let's decompose K so that we can see what is going on more clearly. We can write $K = Q\Lambda Q^T$ such that a) Q is $N \times N$ and orthogonal, with $Q^TQ = I_N$ and b) Λ is $N \times N$ and diagonal with diagonal entries in $\{0, 1\}$. The number of 1s in Λ determines the rank of K, i.e. the dimension of the subspace, which must always be less than N. These conditions are sufficient to guarantee that K is symmetric, positive semi-definite and idempotent.

Note that if all eigenvalues of Λ are one, then K becomes the identity matrix and we get the tautological $\boldsymbol{y} = \hat{\boldsymbol{y}}$ result that we already rejected. That means that at least one of the eigenvalues of Λ must be zero, which in turns means that K must specifically be singular rather than merely positive semi-definite. The requirement that at least one of the eigenvalues be zero means that the rank of K must be less than N, meaning that K projects into a feature space whose dimensionality is lower than that of observation space. In effect, the eigenvalues of K act as on / off switches encoding binary choices about which parts of observation space are projected into feature space.

Somewhere in the distance, we can see the possibility of an algorithm which automatically determines a) which parts (for relevance) and b) how many parts (for sparsity) of observation space to project into feature space. However, yet again, we can see that a naive application of the machinery we have so far would be grossly inefficient. We need to add something - which is regularisation - and, in doing so, let go of something - which is the requirement for K to be idempotent.

2.6 Ridge regression

One of the tell-tale signs of an OLS model over-fitting to data is large absolute values of weights. For example, using a polynomial to model linear observed data will tend to result in a fitted polynomial that matches the data very accurately at the observed data points but which, in between data points, soars very high and swoops very low. We observe high absolute values of weights as the polynomial over-fits the data.

Ridge regression penalises weights using the square of the L2 norm over weights, so that we write a loss function, L as the sum of squared residuals (as for OLS regression) plus a multiple λ of the sum of squared weights. Note that this loss function balances accuracy (from reducing the sum of squared errors) against model complexity (rather loosely), by penalising high values of regression weights.

$$\begin{aligned}L &= \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} + \lambda\boldsymbol{w}^T\boldsymbol{w} \\ &= (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda\boldsymbol{w}^T\boldsymbol{w}\end{aligned}$$

We find the regression weights that minimise the loss by differentiating the loss with respect to the weights $\boldsymbol{w}$ and setting the result equal to (vector) zero.

$$\begin{aligned}\frac{dL}{d\boldsymbol{w}} &= 2X^TX\boldsymbol{w} - 2X^T\boldsymbol{y} + 2\lambda\boldsymbol{w} = \boldsymbol{0} \\ (X^TX + \lambda I_M)\boldsymbol{w} &= X^T\boldsymbol{y} \\ \boldsymbol{w} &= (X^TX + \lambda I_M)^{-1}X^T\boldsymbol{y}\end{aligned}$$

An appropriate value of λ is typically found using cross-validation.

Note that the expression for the weights that minimise loss for some value of λ is very similar to its equivalent under OLS regression. Instead of $(X^TX)^{-1}$ for OLS we now have $(X^TX + \lambda I_M)^{-1}$ for ridge regression.

2.6.1 Going deeper

Let us again insist that X have orthonormal columns so that $X^TX = I_M$. This time we write $K = \frac{1}{1+\lambda}XX^T$ in order to incorporate our regularisation term. This enables us to write $\hat{\boldsymbol{y}} = K\boldsymbol{y}$, as before. And we perform the decomposition $K = Q\Lambda Q^T$ exactly as before but with one important difference. Whereas before, we required the diagonal elements of Λ to be either 0 or 1, we now require them to be in $[0, 1]$ so that we can accommodate the $\frac{1}{1+\lambda}$ term from regularisation, in which $\lambda \geq 0$.

Think of the diagonal terms of Λ (under OLS regression) as being binary switches that decide whether a feature of Q is excluded from (value 0) or included in (value 1) the feature subspace. The product of the non-zero diagonal terms represents the determinant of the features chosen to be included in the feature space. We now need to reduce that determinant from 1 (under OLS) to $\frac{1}{1+\lambda}$ under ridge regression. With $\lambda \geq 0$, this means that, for ridge regression, the diagonal elements must now be real values in $[0, 1]$ rather than the binary switches in $\{0, 1\}$ used for OLS regression.

And suddenly the door swings open to connect ridge regression with a wide variety of well-known models. Let's take a look.