Regression

Chapter 3 Machine Learning

Error metrics

- So unlike with the Bayesian approach from the previous chapter that looks to make classifications based on a discrete random variable, in most cases we want our output y to be continuous
- Naturally, we aim to fit a function to minimise the error in our output variable:

  $$f(x_i) = y_i + E_i$$

  In a lot of cases, in the univariate case we can assume this function fits a linear equation:

  $$f(x_i) = \beta_1 + \beta_2 x_i$$

  and we aim to find the parameters **β** to minimise the error.
- There are a couple of error metrics we can aim to minimise:
  1. The l2 *error metric, the one we use most often, that aims to minimise the average squared difference between predictions and true values: $$E_2=\sqrt(\frac{1}{m}\sum*{i=1}^{m}(f(x*i)-y_i)^2)*

     $$and since we only really care about minimising this term, we can drop the square root :$$

     E_2=\frac{1}{m}\sum{i=1}^{m}(f(x_i)-y_i)^2$$
  2. The l*1 error metric is more crude. It's also known as the Manhattan distance and looks at minimise the mean distance between predictions and true values: $$E_1 = \frac{1}{m}\sum*{i=1}^{m}|f(x_i)-y_i|$$*
  3. The l∞ *metric that minimises the maximum error: $$E*\infty=max|f(x_i)-y_i| $$
  4. Finally you have the l*p metric which is a robust version of the l_1 and l_2 that incorporates the attributes of both: $$l_p=[\sum*{i=1}^{m}(f(x_i)-y_i)^p]^\frac{1}{p}$$

Minimising Error Metrics

- Surprise surprise, we then look to minimise the error metrics via differentiation. Do some partial derivatives with respect to regression coefficients and equate to 0 then get it into matrix form of

  $$A\boldsymbol{\beta} = b$$

  where **β** is the vector of coefficients.

- With multi-dimensional data it's the same shabang where you're fitting for a function:

$$f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_x + \ldots \beta_n x_n$$

- When dealing with higher order polynomials, it is also possible to treat each order as a different parameter i.e.

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

becomes

$$f(\boldsymbol{x'}) = \beta_0 + \beta_1 x'_1 + \beta_2 x'_2$$

where x' = (x,x^2). Note this is computationally similar to just treating them as one variable, but is used in some context so is good to be aware of.

- Sometimes the natural fit is an exponential function instead of a linear one, e.g.

$$f(x_i) = \beta_1 e^{\beta_2 x_i}$$

in such a case we just take logs on each side so that we can still put crap into matrix form: e.g.

$$ln(y_i) = ln(\beta_1) + \beta_2 x_i$$

and as such we have for our matrix equation

$$\boldsymbol{\beta} = (ln(\beta_1), \beta_2)$$

Gradient Descent - Our old friend

- So what if we can't fit it to a linear or exponential function? Enter the old trusty:

$$\boldsymbol{\beta_{k+1}} = \boldsymbol{\beta_k} + \delta \nabla e(\boldsymbol{\beta_k})$$

where the gradient for each step can be calculated using a finite difference approximation (central, forward backward, whatevs, here is forward):

$$\frac{\partial E_2}{\partial \beta_i} \approx \frac{E_2(\beta_i + \delta_{\beta_j}) - E_2(\beta_i)}{\delta_{\beta_j}}$$

- You can also use **alternating gradient descent** where you descend on one parameter at a time in your hyperspace. This is best where there is independence between parameters.

Ridge Regularisation

- Of course we don't want to overfit to training data so we use **ridge regularisation** to penalise an increase in complexity when descending such that our error metric

becomes:

$$E_2 = \sum_i (f(x_i) - yi)^2 + \lambda \sum_j \beta_j^2$$

the selection of **λ**, the penalisation term, is based on the bias-variance trade-off that must be selected: