SL02: Regression & Classification

Regression:

- Regression maps continuous (not discrete) inputs to outputs. In regression, we try to find a mathematical relationship based on measurement points. In some sense, Regression means “falling back to the mean”.
- Linear regression: Finding a linear relationship between some data points. The slope of that line will always be less than 1, which makes it regression to the mean. The best linear function would be the one that minimizes the squared error between the data points and their corresponding mathematical measurements.
- How to find the best linear fit?
  - Using calculus, we can calculate the least squared error.

$$f(x) = c$$

$$E(c) = \sum_{i=1}^{n} (y_i - c)^2$$

  - $E(c)$ → Sum of Squared Errors. It’s used because it’s easy to find it’s minimum using calculus (Squaring the differences makes this error function differentiable).
  - n → Number of data points.
  - y_i → The correct value of data point i.
  - c → The approximation function we’ll use to predict the best fit. In the simplest case it would be just a constant c, which will result in a horizontal line across the data points.
  - So, this function will calculate the sum, over all data points, of squared difference between the result of our prediction function, and the actual correct value.
  - To find the best c, we take the derivative of $E(c)$ (with respect to c) and set it equal to 0.

$$\frac{dE(c)}{dc} = \sum_{i=1}^{n} 2(y_i - c).(-1)$$

$$-\sum_{i=1}^{n} 2(y_i - c) = 0$$

$$\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} c = 0$$

$$\sum_{i=1}^{n} c = n.c$$

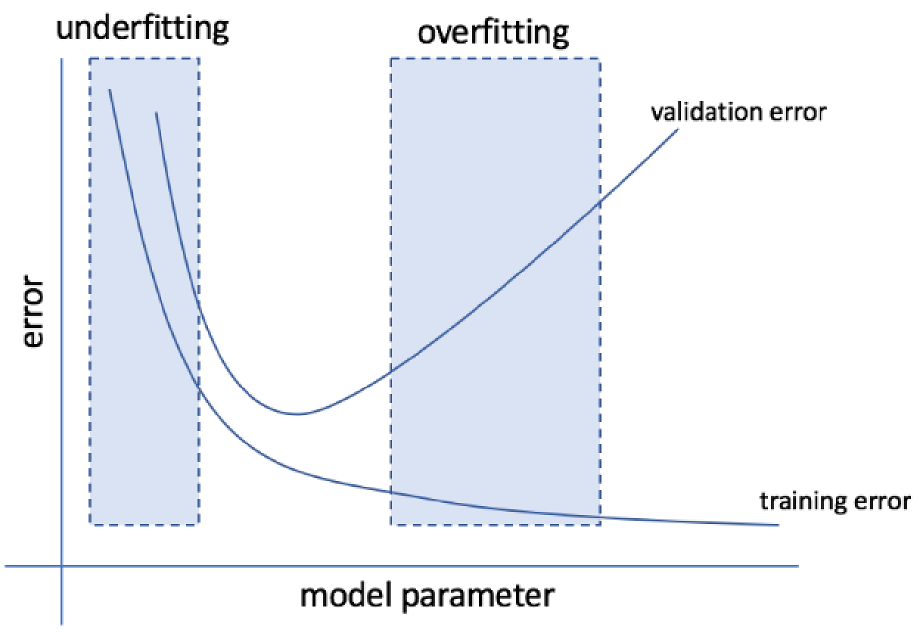
$$c = {\sum_{i=1}^{n} y_i}/{n}$$

  - It turned out that the best approximation function will be to take the mean of all the data points.

- Polynomial regression:
  - For a set of values $X = (x_1, x_2, \dots, x_n)$, the goal is to find a set of coefficients (weights) $C = (c_0, c_1, c_2, \dots, c_n)$ that will produce the best approximation function $f(x)$ where:
  $$f(x) = c_0 + c_1x_1 + c_2x_2 + \dots + c_nx_n$$
  - A higher order of polynomial will always model the data better (maximum order is the number of data points), but this will result in overfitting (Training error will decrease, and testing error will increase).
  - Arranging these sets in matrices will help us solve for the coefficients:
  $$c_0 + c_1x + c_2x^2 + c_3x^3 \approx y$$
  $$\begin{bmatrix} 1 & x_1 & {x_1}^2 & {x_1}^3 \\ 1 & x_2 & {x_2}^2 & {x_2}^3 \\ 1 & x_3 & {x_3}^2 & {x_3}^3 \\ . & & . & . \\ . & & . & . \\ 1 & x_n & {x_n}^2 & {x_n}^3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_n \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ y_n \end{bmatrix}$$
  $$Xw \approx Y$$
  $$X^TXw \approx X^TY$$
  $$(X^TX)^{-1}X^TXw \approx (X^TX)^{-1}X^TY$$
  $$(X^TX)^{-1}X^TX = 1$$
  $$w \approx (X^TX)^{-1}X^TY$$
  - w is the vector of weights, which tells us how important the corresponding x is in predicting the output.
- Regression errors:
  All the datasets include error points that we don't want to model in our algorithm. The error sources include:
  - Sensor error: When a HW device misinterprets physical values.
  - Malicious data: Putting error points intentionally.
  - Transcription error: Human errors that can happen during the process of data gathering/transformation.
  - Unmodeled influences: Missing other features that can affect our data.

  We need to avoid fitting the error itself, we want to fit the underlying relationship.
- Cross Validation:
  - One way to have a better-fitting model is to use higher polynomial (Up until the number of features), but this might lead to overfitting (Perfectly model the training data but fail to generalize on real world data).
  - The goal here is to use a model that is complex enough to model the training set without causing problems when generalizing to a test set.
  - Cross Validation is a process to quantify a model's ability to generalize. We assume that the data Independent and Identically Distributed (IID), which means that there's no inherent difference between training, testing and real-world data.

- Cross Validation steps:
  1. Randomly partition the training data into k folds of equal size.
  2. Train the model on all the folds except for one $(k-1)$.
  3. Validate the model's performance using the fold that was not used in training.
  4. Repeat steps 2 and 3 using different combinations of these folds.
  5. Average the error in predicting the polynomial trained on the training folds.
- Notes on Cross Validation:
  1. Average cross validation error will be higher than training error for 0 order.
  2. Average cross validation error decreases by increasing the order of polynomial (similar to training error).
  3. After a certain point, increasing the degree of polynomial will result in overfitting, and the Cross Validation error will start to increase.



- Increasing the order of polynomial allows for better fit to the data, however after a certain point, the ability to better fit the data comes at the expense of model generalization.

- Other input spaces:
  So far, we've seen only scalar/continuous input. Other scenarios could be:
  - Vector/continuous input: This will include more input features, and it will generalize by going from a line on a dimensional plot, to a plan where the lower dimensions describe the vector of inputs, and the highest describes the output.
  - Discrete/continuous input: Certain types of attributes are difficult to encode/quantify (e.g. hair color). Here are some ways to encode these attributes:
    1. Enumerating, which means just giving a number to each possible value of this attribute. This, however, can be misleading because a correlation between the ordering of the enumeration and the its value can be interpolated.
    2. Represent each possible value of the attribute as a Boolean.