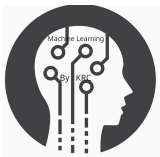
Machine Learning (Regression Model)

Prof K R Chowdhary

CSE Dept., MBM University

November 29, 2024



Introduction

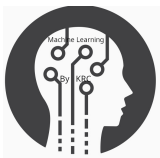
$\Rightarrow$ The regression is supervised learning. We consider an example, where we would like to build a model that approximates the relationship f between the number of years of experience in software industry $\mathbf{x}$ and corresponding annual income $\mathbf{y}$.

$$\mathbf{y} = f(\mathbf{x}) + \varepsilon \qquad (1)$$

where $\mathbf{x} = (x_1, x_2, ..., x_n)$ is (input) years in software industry and $\mathbf{y} = (y_1, y_2, ..., y_n)$ is predicted (output) annual income, f is function describing relationship between $\mathbf{x}$ and $\mathbf{y}$.

$\Rightarrow$ Machine learns f given $\mathbf{x}$, $\mathbf{y}$. The ε is *random error term* either positive or negative with mean zero, and represents irreducible error in the model, which is the theoretical limit around the performance of the algorithm due to inherent noise.

$\Rightarrow$ Two tasks of supervised learning are: 1) Regression, which predicts a continuous numerical value, and 2) to assign a label, for example, to predict whether a given picture is of a "cat" or "dog?"



Regression Model

$\Rightarrow$ In regression, data is split into *training data* set and *test data* set. Goal is: to learn linear model using *ordinary least squares regression* that predicts a new y given a previously unseen x, with as little error as possible.

$\Rightarrow$ It is a parametric method, to find a function that predicts $\hat{y}$ for given specific x:

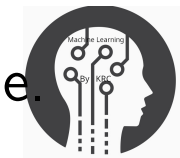
$$\hat{y} = \beta_0 + \beta_1 * x + \varepsilon, \qquad (2)$$

here β_0 is y-intercept (point where line cuts y axis), $\beta_1 =$ slope of line, i.e., how much it increases or decreases by one year of experience.

$\Rightarrow$ Goal: learn model parameters β_0 and β_1 that minimizes *error*.

$\Rightarrow$ To find best value of parameters:

1. Define a *cost function* (or "loss function"), that measures how inaccurate the predictions are?
2. Find the parameters that minimize cost, i.e., make this as accurate as possible.



Regression Model

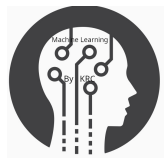
$\Rightarrow$ In 2D it is a line of best fit, in 3D it is plane, ...

$\Rightarrow$ Mathematically, we look at difference between each real data point (y) and this model's prediction ($\hat{y}$).

$\Rightarrow$ Differences are squared to avoid negative numbers, and we penalize large differences. At end, all squares are summed up and averaged: a measure of how well our data fits a line. For n number of observations:

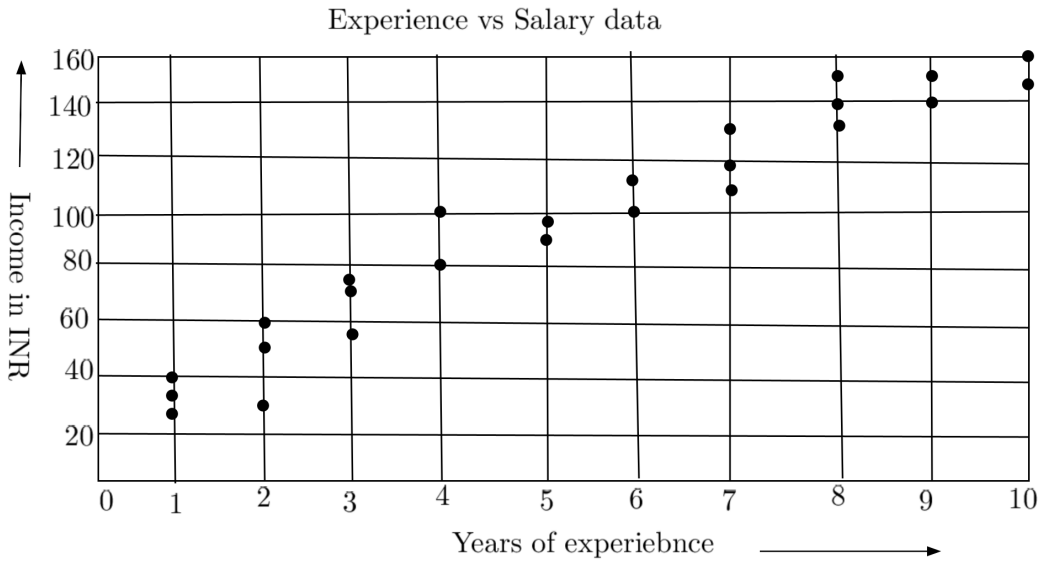
$$Cost = \frac{\sum_1^n((\beta_1 x_i + \beta_0) - y_i))^2}{2 * n} \quad (3)$$

"Cost" should be minimum possible. Using $2 * n$, instead of n makes the mathematics workout more cleanly when taking derivative to minimize loss (page no. 7). The random error term ε is not accounted for in equation (3), its mean value is zero.



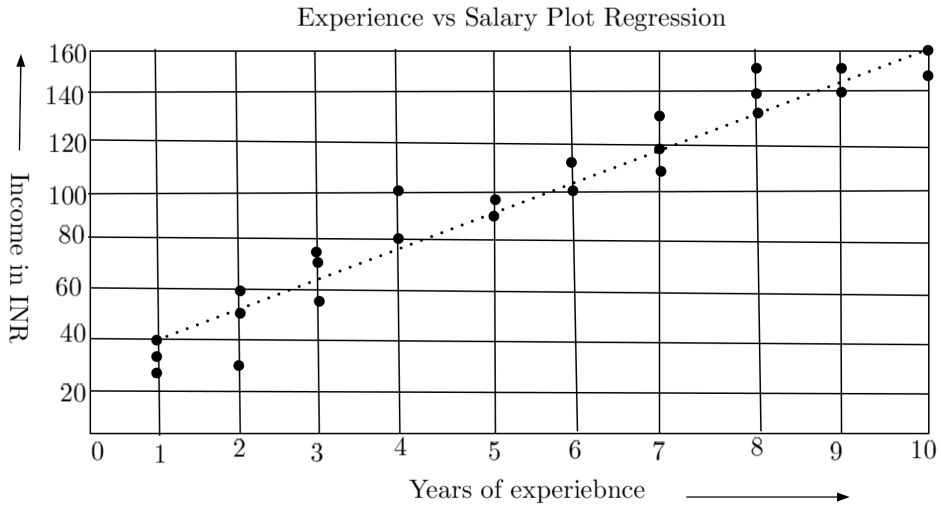
Regression Model

$\Rightarrow$ *Example.* Let an attribute vector $\mathbf{x} = [1, 1, 1, 2, ..., 10, 10]$ is years of experience of software developers, and vector $\mathbf{y} = [26, 33, 40, 30, ..., 144, 160]$ is corresponding annual income in thousands of INR. The values are plotted in Fig-1 below, as "Experience vs Salary."



$\Rightarrow$ Linear regression curve is obtained in Fig. (next slide) for experience vs salary data shown in Fig. 1, using cost equ. (3). The β_0 and β_1 are chosen such that *cost* is minimum, for all input data of experience vs salary.

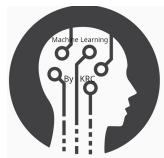
Fig.2 (Experience vs Salary Plot Regression).



$\Rightarrow$ Values of β_0, β_1 are substituted in equation (2) to get linear relation, which is *induced classifier*, and can classify new data in similar way we did in liner classifier in equation (4).

$$2.5 - 0.8x_1 - x_2 = 0 \tag{4}$$

For this simple problem we can compute a closed form solution using calculus to find optimal β parameters that minimize cost function. As cost function grows in complexity, finding a closed form calculus is difficult (requires different method).



Regression Model

Goal of gradient descent: Find minimum of regression model's loss function through iteration, to achieve better approximation.

$\Rightarrow$ It is like walking through a valley with a blindfold on. Goal: find bottom of a valley.
Approach: to find steepest slope of ground, and move and repeat this.

$\Rightarrow$ Bottom is reached, if we end up at the same elevation. (it is opposite to hill-climbing).

Cost function:

$$Cost = \frac{\sum_1^n((\beta_1 x_i + \beta_0) - y_i))^2}{2 * n} \quad (5)$$

Objective: Minimize this cost function (loss function).

$\Rightarrow$ Let $f(\beta_0, \beta_1) = z$. To begin process, initial guess is required for β_0 and β_1, that *minimize* the function.

$\Rightarrow$ Next, partial derivatives is to be computed w.r.t each beta parameter: $[dz/d\beta_0, dz/d\beta_1]$[1].

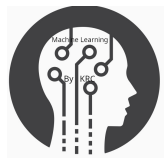
[1]Partial derivative: how much it is increased or decreased if we increase β_0 or β_1 by a very small amount.

Regression Model...

$\Rightarrow$ It means, how much would increasing your estimate of annual increase, assuming zero experience (β_0), increases the cost of the model? (You want to go to opposite direction so that you end up walking downhill and minimizing the cost).

$\Rightarrow$ Similarly, if you increase your estimate of how much each incremental year of experience effects the income (β_1), how much does this increase the loss z. If the partial derivative $dz/d\beta_1$ is a negative number, then increasing β_1 is good because it will reduce the total cost. If it is positive number, you want to increase β_1. If zero, do not change β_1 because it means you have reached the optimum.

$\Rightarrow$ Repeat this until you reach to bottom, i.e., the algorithm has converged and loss has been minimized. (There are many more tricks in accomplishing this, but this largely we do for finding the optimal parameters for this method).



Overfitting and Underfitting

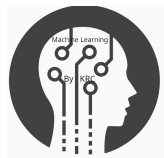
$\Rightarrow$ *Overfitting* is a common problem in ML where "learning function perfectly explains the training data through which the model has learned, but the model does not generalize well to unseen test data."

$\Rightarrow$ *Overfitting* happens when model over-learns from training data to the point that it starts picking up idiosyncrasies[2] that are not representative of pattens in the real-world. This makes the model increasingly complex.

$\Rightarrow$ *Underftting* is just the reverse: model is not complex enough to capture underlying trend in data.

$\Rightarrow$ As model increases in complexity and becomes more flexible, its bias decreases. By this, model does a better job of explaining the training data, but, variance increases. Due to which model does not generalize well.

$\Rightarrow$ Hence, for an ideal model, there is need of low bias and low variance.



[2] an unusual characteristic

Overfitting and Underfitting ...

$\Rightarrow$ Effects of bias and variance on model accuracy are shown in Fig. 1, for solar cell exposed to sunlight vs. current produced in milli-amps. High bias causes underfit, high variance overfit and both low is just a fit.

$\Rightarrow$ Overfitting can be limited: 1). Using more *training data*: due to this, it is harder to overfit data, by learning too much from any single trg. example. 2). Using *regularization*: Penalty added to cost function helps in building a model that assigns too much explanatory power to any one feature or allows too many features to be considered.

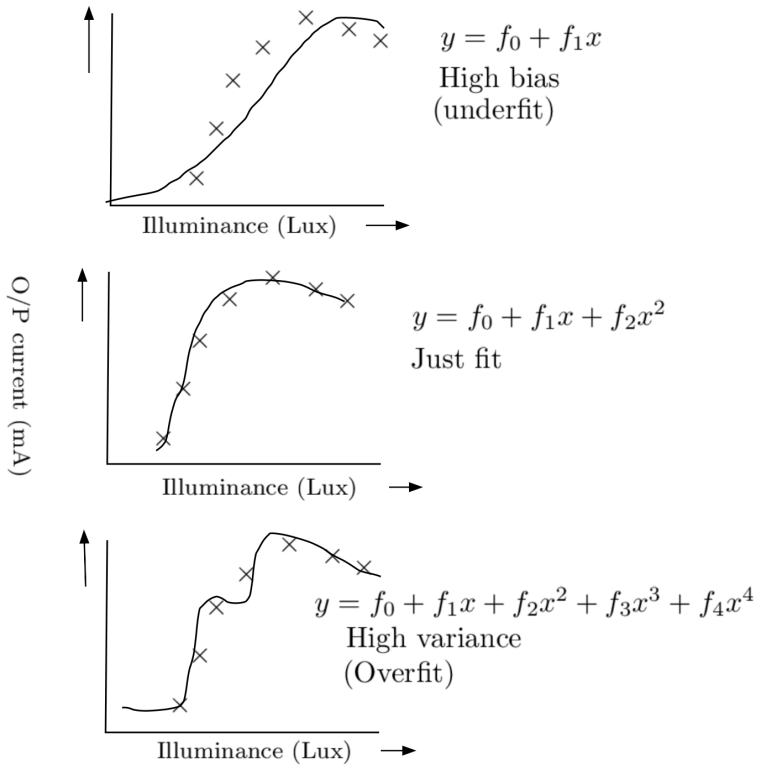


Figure 1: Effect of bias and variance on a model's accuracy

Overfitting and Underfitting ...

$$cost = \frac{\sum_{i=1}^{n}((\beta_1 x_i + \beta_0) - y_i)^2}{2 * n} + \lambda(\beta_0^2 + \beta_1^2) \qquad (6)$$

$\Rightarrow$ 1st term in (6) is normal cost function, 2nd is regularization: adds penalty for large beta coefficients that give too much explanatory power to any specific feature.

$\Rightarrow$ Due to these terms, cost function balances between two priorities: explaining training data vs. preventing that explanation from becoming overly specific.

$\Rightarrow$ λ coefficient of regularization term is a *hyper-parameter*. In a general setting (of model), λ can be increased or decreased (tuned) to improve performance.

$\Rightarrow$ Higher value of λ will be more aggressive to penalize large β coefficients which could result to potential overfitting.

$\Rightarrow$ For best value of λ, use *cross-validation*: it holds out a portion of training data during training, and then seeing how well model explains portion of data that has been held out.