VIT-AP UNIVERSITY

Locally Weighted Regression

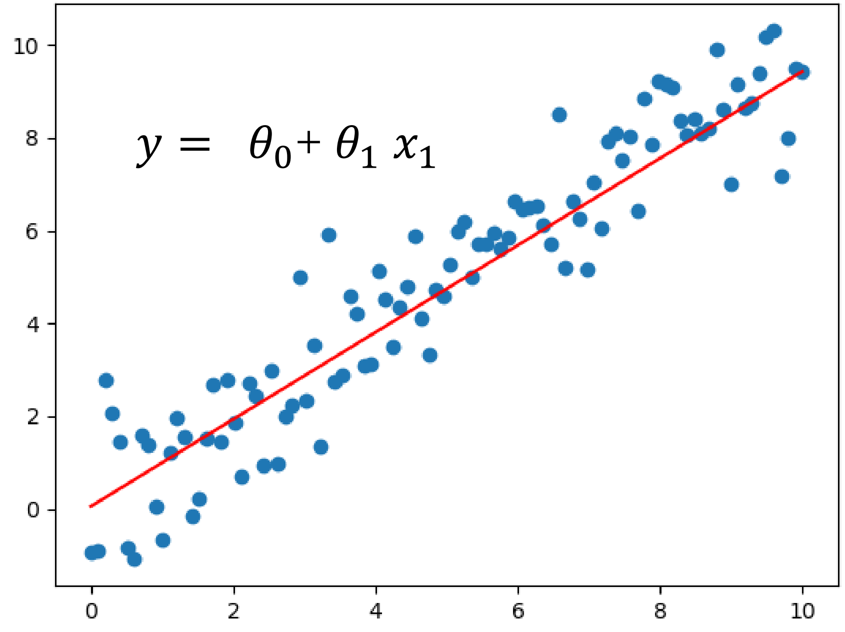
Dr. Kuppusamy .P
Associate Professor / SCOPE

Locally Weighted Linear Regression

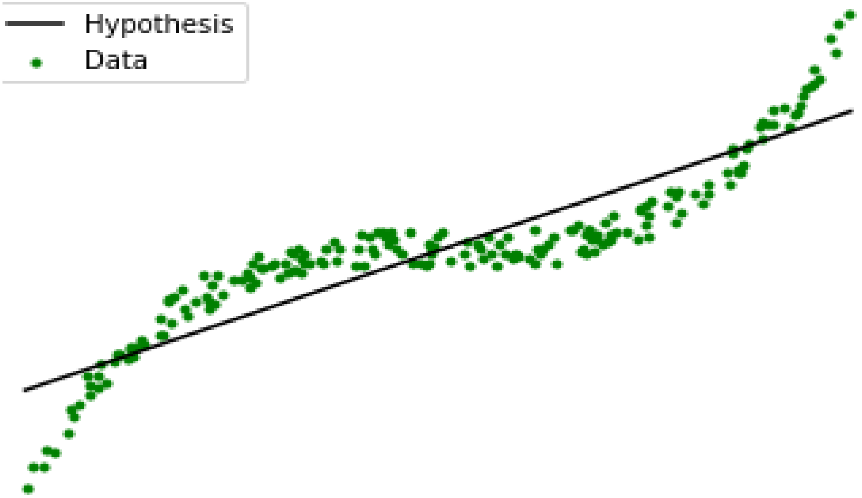
- Simple linear regression algorithm performs better during the relationship between the data is linearly separable.
- In parametric learning, fixed number of parameters that fit the training data.
- The plot shows the normally distributed data with a specific and limited number of independent variables (parameters).
- If data is non-linear then Linear regression fails to find a best-fit line.
- It is addressed by **Locally weighted algorithm.**

Non-Parametric Learning Algorithm

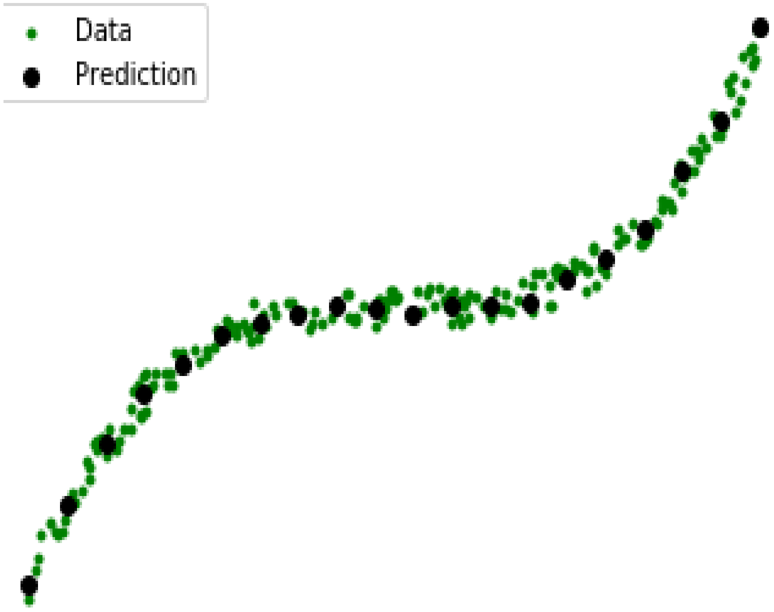
- Number of parameters grows with the size of the training dataset i.e., learning algorithm needs to keep around an entire training set, even after training.



Simple linear regression

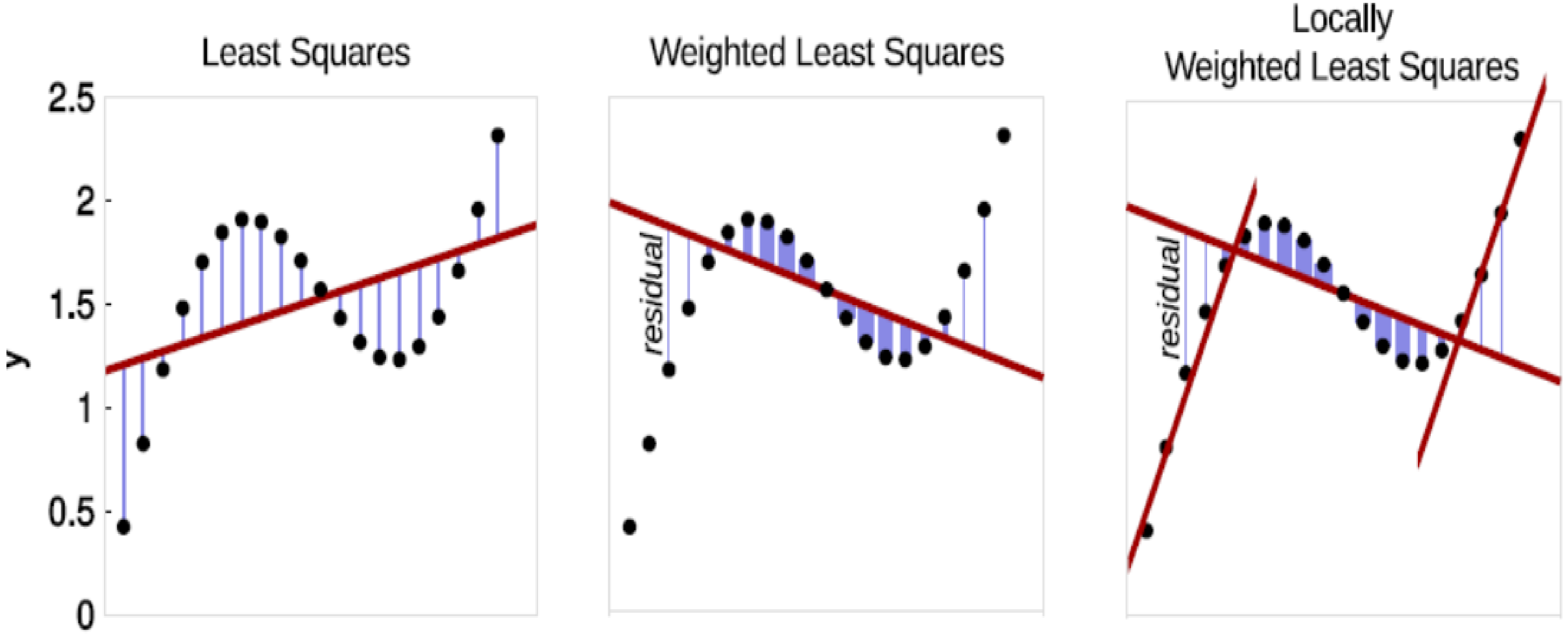


Locally Weighted Regression (LWR)



- Locally weighted regression is instance-based learning algorithm.
- It is a non-parametric algorithm i.e., model does not learn a fixed set of parameters.
- **Local** denotes that functions is approximated using datapoints that are close to the test datapoint.
- **Weighted** represents contribution of each training sample is weighted by its distance from the test datapoint x.
- Parameters θ are computed individually for each new data point x.
- When computing $\theta,$ a more "**weightage**" is given to the data points in the training set closer to test datapoint x than far away from x.
- In this case, each data point becomes a weighting factor that states the influence of the data point for the prediction.
- LWR is lazy learning because the processing of the training data is shifted until a new test data point to be answered.
- This approach makes LWR a very accurate function approximation method where it is easy to add new training points.
- More expensive computations.

Locally Weighted Regression (LWR)



The thickness of the lines indicates the weights.

Locally Weighted Regression (LWR)

- Let consider for new data (query) point $\boldsymbol{x}_{\mathbf{q}}$**,** build an approximation function f' that fits the training dtapoints close to the new datapoint $\boldsymbol{x}_{\mathbf{q}}$
- It computes the predicted value for $\boldsymbol{x}_{\mathbf{q}}$

$$f'(x_q) = \theta_0 + \theta_1\, a_1(x) + \theta_2 a_2(x) + \ldots\ldots + \theta_n a_n(x)$$

- $a_r(x)$ denotes the value of the r[th] **attribute** of instance x. θ are **weights**.
- Initially, assign the random weights to the θ. Then fine-tune the weights.

Parameters tune by Gradient Descent of Artificial Neural network.

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - y'^{(i)}\right)^2$$

Update the weights using Gradient Descent

- $\theta_{new} = \theta_{old} + \Delta\theta_j$
- $\Delta\theta_j = -\eta \frac{\partial E}{\partial \theta_j}$

Update the weights in Linear Regression

- Update the weights

$$\theta_{new} = \theta_{old} + \Delta\theta_j$$

$$\Delta\theta_j = -\eta \frac{\partial E}{\partial \theta_j}$$

- $\Delta\theta_j = -\eta \frac{\partial}{\partial \theta_j}\left(\frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - y'^{(i)}\right)^2\right)$ $\qquad$ $y'^{(i)} = \theta_i x_i$, i= 0 …. n

- $\Delta\theta_j = -\eta \left(\frac{1}{2}\sum_{i=1}^{m} 2\left(y^{(i)} - y'^{(i)}\right)(0 - \frac{\partial \theta_j x_j}{\partial \theta_j}\right)$

- $\Delta\theta_j = -\eta \sum_{i=1}^{m}\left(y^{(i)} - y'^{(i)}\right)(x_{ij})$

- $\theta_{new} = \theta_{old} - \eta \sum_{i=1}^{m}\left(y^{(i)} - y'^{(i)}\right)(x_{ij})$

Repeat until minimize the error.

Modifying cost function for LWR

Modified cost function is

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{m}(f(x) - f'(x))^2$$

It leads to the Gradient Descent chain rule

$$\Delta\theta_j = -\eta \sum_{x \in D}(f(x) - f'(x))\, a_j(x)$$

- It is a global error. But we need to derive local approximation error.
- So, $E(\theta)$ should emphasize fitting to the local (close to new data point) training examples.
- $\theta_{new} = \theta_{old} - \boldsymbol{\eta} \sum_{\boldsymbol{x \in k\ nearest\ neighbors\ of x_q}} \boldsymbol{K}(\boldsymbol{dist}(\ \boldsymbol{x_q}, \mathbf{x}))\ (\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f'}(\boldsymbol{x}))\ \boldsymbol{a_j}(\boldsymbol{x})$

Modifying cost function for LWR

For this three possible criteria

Minimize the squared error using **k nearest neighbors**

1. $E_1(x_q) = \frac{1}{2}\sum_{x \in k \text{ } nearest \text{ } neighbors \text{ } of x_q}(f(x) - f'(x))^2$

- Minimize the squared error over entire D training datset, while weighting the error of each training datapoint using decreasing function **K** of its distance from Minimize the squared error .

2. $E_2(x_q) = \frac{1}{2}\sum_{x \in D}(f(x) - f'(x))^2 \boldsymbol{K}(dist(x_q, \text{x}))$

- Combine both 1 and 2

3. $E_3(x_q) = \frac{1}{2}\sum_{x \in k \text{ } nearest \text{ } neighbors \text{ } of x_q}(f(x) - f'(x))^2 \boldsymbol{K}(dist(x_q, \text{x}))$

Apply Gradient Descent

$$\Delta\boldsymbol{\theta_j} = -\boldsymbol{\eta}\sum_{x \in \boldsymbol{k} \text{ } \boldsymbol{nearest} \text{ } \boldsymbol{neighbors} \text{ } \boldsymbol{of} x_q} \boldsymbol{K}(\boldsymbol{dist}(\boldsymbol{x_q}, \mathbf{x}))\,(\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{x}))\,\boldsymbol{a_j}(\boldsymbol{x})$$

Update the weights using $\theta_{new} = \theta_{old} + \Delta\theta_j$

$$\theta_{new} = \theta_{old} - \boldsymbol{\eta} \sum_{x \in \boldsymbol{k} \text{ } \boldsymbol{nearest} \text{ } \boldsymbol{neighbors} \text{ } \boldsymbol{of} x_q} \boldsymbol{K}(\boldsymbol{dist}(\boldsymbol{x_q}, \mathbf{x}))\,(\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{x}))\,\boldsymbol{a_j}(\boldsymbol{x})$$

References

1. Tom M. Mitchell, Machine Learning, McGraw Hill , 2017.

2. EthemAlpaydin, Introduction to Machine Learning (Adaptive Computation and Machine Learning), The MIT Press, 2017.

3. Wikipedia