Machine Learning
ICT-4261

By-
Dr. Jesmin Akhter
Professor
Institute of Information Technology
Jahangirnagar University

Contents

The course will mainly cover the following topics:

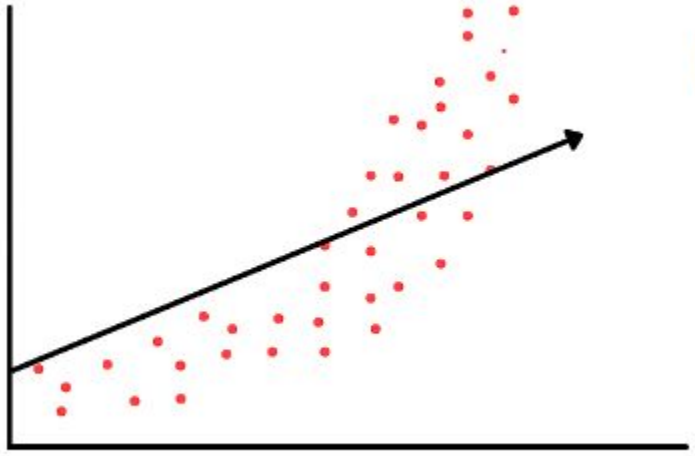
- ✔ A Gentle Introduction to Machine Learning
- ✔ Linear Regression
- ✔ Logistic Regression
- ✔ Naive Bayes
- ✔ Support Vector Machines
- ✔ Decision Trees and Ensemble Learning
- ✔ Clustering Fundamentals
- ✔ Hierarchical Clustering
- ✔ Neural Networks and Deep Learning
- ✔ Unsupervised Learning

Outline

- ✔ **Linear Regression**
  - Polynomial regression
  - Different Types of Gradient Descent Algorithms
  - Ridge, Lasso

Polynomial Regression

- ✔ A simple linear regression algorithm only works when the relationship between the data is linear. But suppose we have non-linear data, then linear regression will not be able to draw a best-fit line. Simple regression analysis fails in such conditions.
- ✔ Consider the below diagram, which has a non-linear relationship, and you can see the linear regression results on it, which does not perform well, meaning it does not come close to reality.
- ✔ Hence, we introduce polynomial regression to overcome this problem, which helps identify the curvilinear relationship between independent and dependent variables.



Linear Regression

$$Y = \theta_0 + \theta_1 x$$

Polynomial Regression

- ✔ It is also called the special case of **Multiple Linear Regression** in ML. Because we add some **polynomial terms** to the **Multiple Linear regression equation** to **convert** it into **Polynomial Regression**.
- ✔ It makes use of a **linear regression model** to fit the **complicated and non-linear functions and datasets**.
- ✔ ***"In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."***

How Does Polynomial Regression Handle Non-Linear Data?

- ✔✔ Polynomial regression is a form of Linear regression where only due to the Non-linear relationship between dependent and independent variables, we add some **polynomial terms to linear regression to convert it into Polynomial regression.**
- ✓ In polynomial regression, the relationship between the dependent variable and the independent variable is modeled as an nth-degree polynomial function. When the polynomial is of degree 2, it is called a quadratic model; when the degree of a polynomial is 3, it is called a cubic model, and so on.
- ✓ Now we have multiple features. Hypothesis can be written
  - $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 = \theta_0 + \sum_{i=1}^{m} \theta_i x_i$
- ✓ Before feeding data to a model in the preprocessing stage, we convert the input variables into polynomial terms using some degree.

How Does Polynomial Regression Handle Non-Linear Data?

- ✔ The degree of the polynomial determines the complexity of the curve that can be fit to the data.
  - Higher-degree polynomials can capture more intricate patterns, But they can also lead to overfitting,
  - For smaller values of degree, the model tries to underfit,
  - so we need to find the optimum value of a degree.

Equation of the Polynomial Regression Model

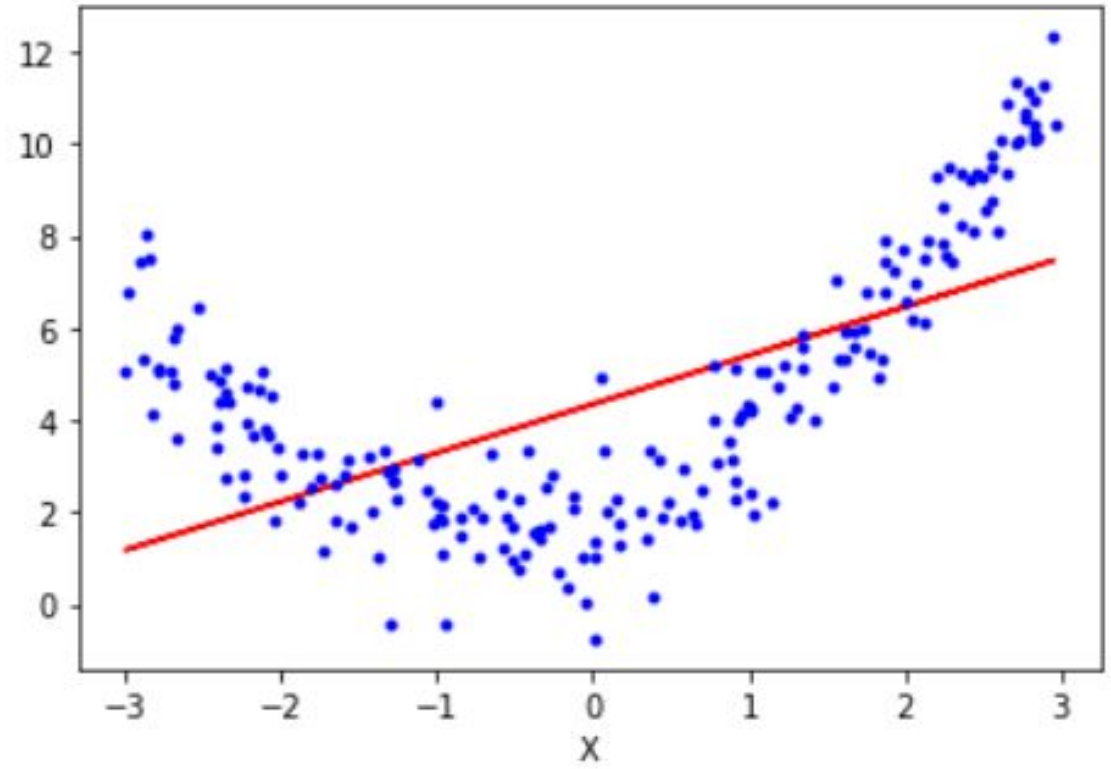
Simple Linear Regression equation: $y = b_0 + b_1 x$ .....(a)

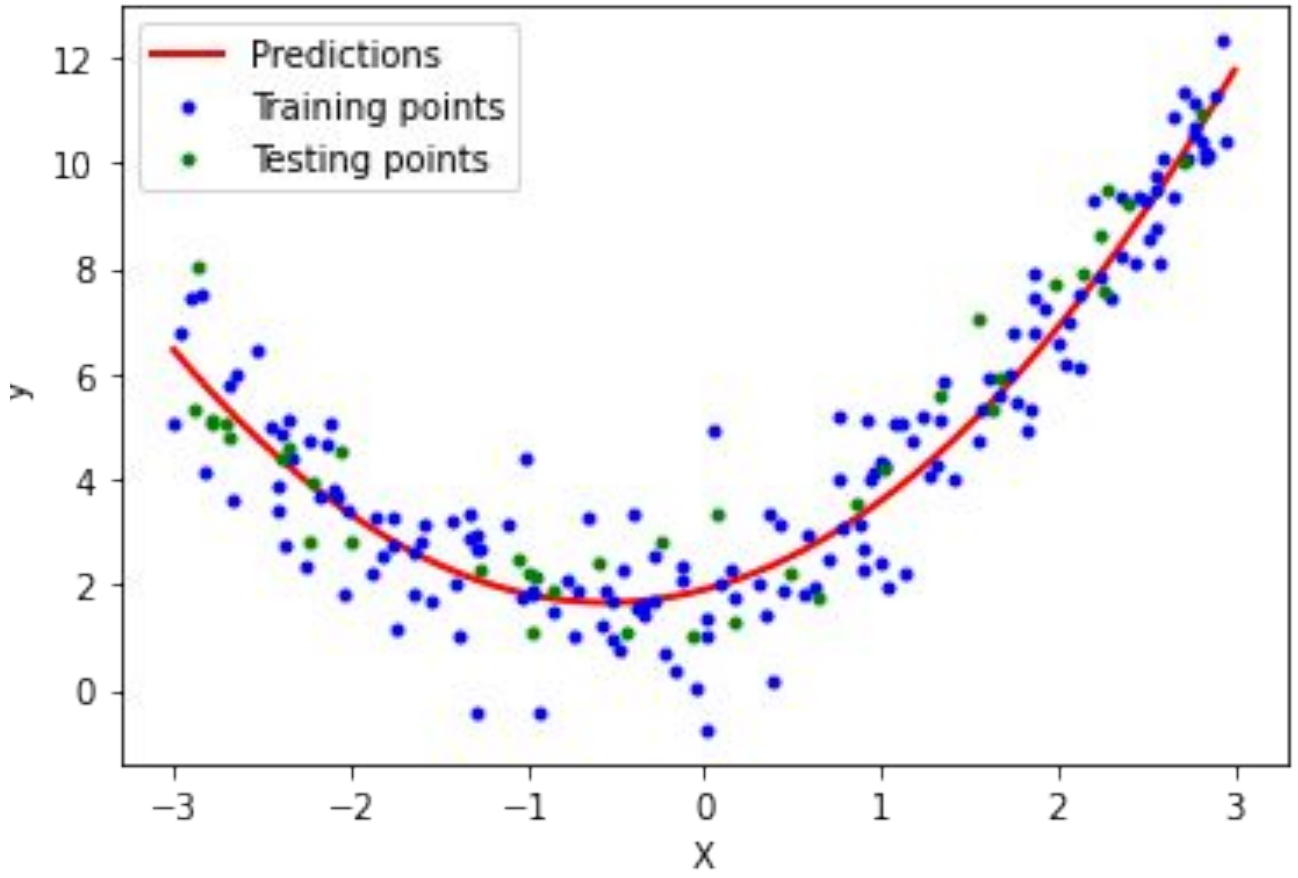
Multiple Linear Regression equation: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$ .....(b)

Polynomial Regression equation: $y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + \ldots + b_n x^n$ .....(c)

- ✔ When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables.
- ✔ The **Simple and Multiple Linear equations** are also Polynomial equations with a **single degree**,
- ✔ The **Polynomial regression equation** is Linear equation with the **nth degree**. So if we **add** a **degree to our linear equations**, then it will be **converted into Polynomial Linear equati**ons.

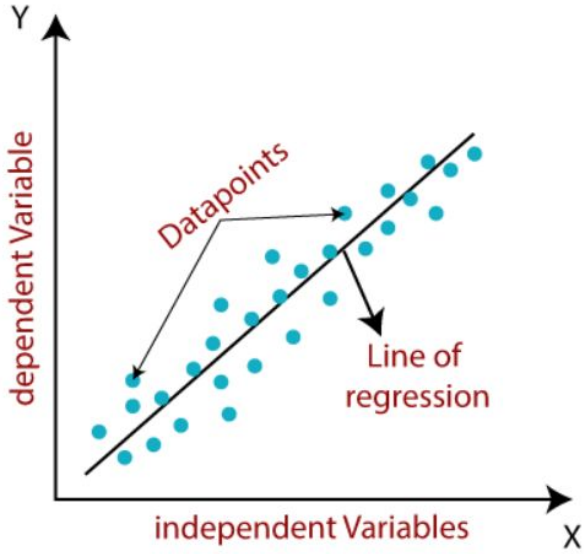
How Does Polynomial Regression Handle Non-Linear Data?

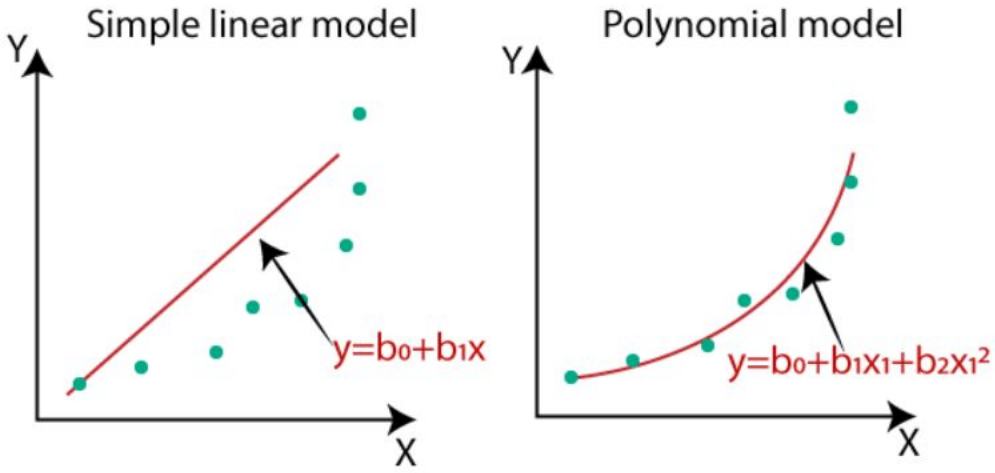




Need for Polynomial Regression

- ✔ If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in the Simple Linear Regression.
- ✔ If we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output.
  - Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- ✔ In the diagram, we have taken a dataset which is arranged **non-linearly.** So if we try to cover it with a **linear model**, then we can clearly see that it **hardly covers any data point**. On the other hand, a curve is **suitable to cover most of the data points**, which is of the **Polynomial model**.
- ✔ So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**.





Gradient Descent

Gradient descent is an optimization algorithm used to minimize the cost function by adjusting the model parameters (θ). It iteratively updates θ in the direction of the steepest decrease in the cost function. The update formula for gradient descent is given by:

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Where:

- θ_j is the j-th parameter.
- α is the learning rate, a hyperparameter that determines the size of the steps in the parameter space.
- m is the number of training examples.
- $h_\theta(x^{(i)})$ is the predicted value for the i-th example.
- $y^{(i)}$ is the actual value for the i-th example.
- $x_j^{(i)}$ is the j-th feature of the i-th example.

Gradient descent for multiple variables

- ✔ Fitting parameters for the hypothesis with gradient descent
  - Parameters are θ_0 to θ_n
  - Instead of thinking about this as n separate values, think about the parameters as a single vector (θ)
  - Where θ is n+1 dimensional
- ✔ Our cost function is

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- ✔ Where,
- ✔ Similarly, instead of thinking of J as a function of the n+1 numbers, J() is just a function of the parameter vector J(θ)

Gradient descent for multiple variables

- ✔ Gradient Descent is an iterative optimization process that searches for an objective function's optimum value (Minimum/Maximum).
- ✔ It is one of the most used methods for changing a model's parameters in order to reduce a cost function in machine learning projects.
- ✔ The primary goal of gradient descent is to identify the model parameters that provide the maximum accuracy on both training and test datasets.
- ✔ The algorithm might gradually drop towards lower values of the function by moving in the opposite direction of the gradient, until reaching the minimum of the function.

Gradient descent for multiple variables

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

} (simultaneously update for every $j = 0, \ldots, n$)

- ✔ $\theta_j = \theta_j$ - learning rate (α) times the partial derivative of $J(\theta)$ with respect to θ_j (j=0, 1, 2,...n)
- ✔ We do this through a simultaneous update of every θ_j value
- ✔ Implementing this algorithm
  - When n = **1**

$$\theta_0 := \theta_0 - \alpha * \frac{d}{d\theta_0} J(\theta_0, \theta_1)$$

–

$$\theta_1 := \theta_1 - \alpha * \frac{d}{d\theta_1} J(\theta_0, \theta_1)$$

where

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right)$$

where

$$\frac{d}{d\theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right) * x^i$$

Gradient descent for multiple variables

- ✔ We now have an almost identical rule for multivariate gradient descent

New algorithm $(n \geq 1)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$\frac{\partial}{\partial \theta_j} J(\theta)$

(simultaneously update θ_j for $j = 0, \ldots, n$) $\}$

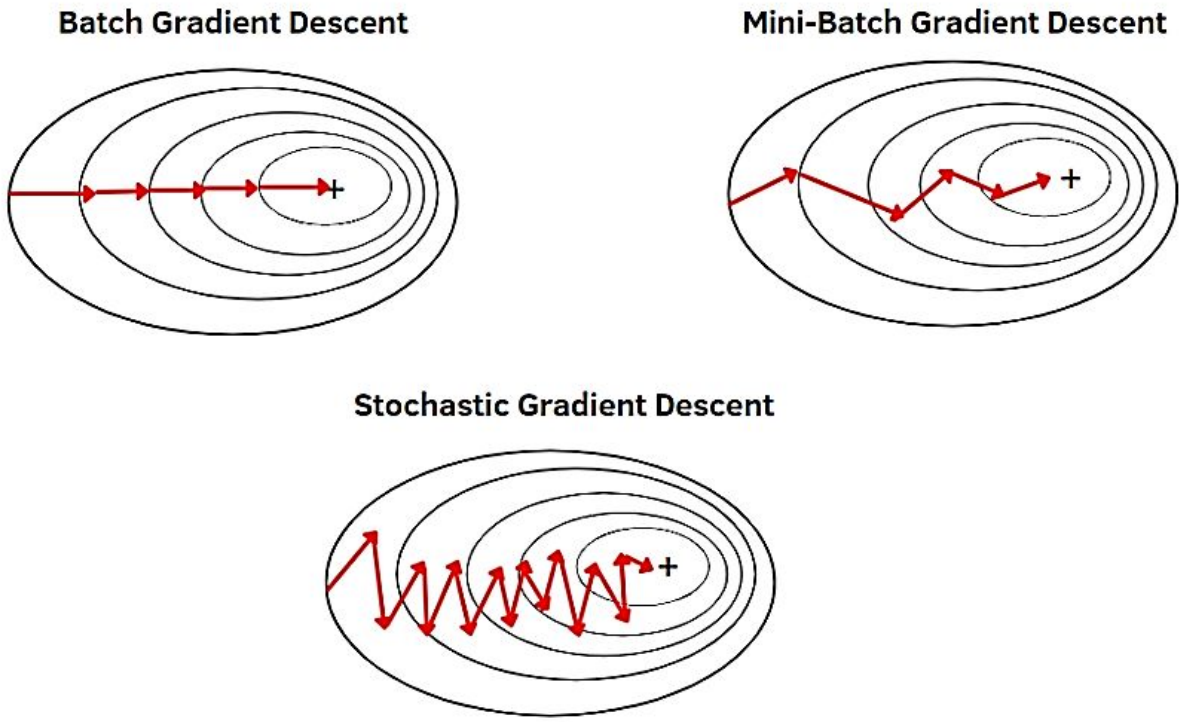
- ✔ It's important to remember that

$$\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} = \frac{\partial}{\partial \theta_j} J(\theta)$$

Different Types of Gradient Descent Algorithms

Gradient descent algorithm can be divided into three categories

1. **Batch Gradient Descent**
2. **Stochastic Gradient Descent (SGD)**
3. **Mini Batch Gradient Descent**



Batch Gradient Descent

- ✔ In batch gradient descent, the update of model parameters or weights is calculated using the entire training dataset at once. This means that for each step in the training process, the algorithm calculates the gradient of the cost function for the entire dataset.
- ✔ This algorithm is often used when the **training dataset is relatively small** and can fit into memory comfortably.
- ✔ For example, consider a linear regression model where you're predicting housing prices based on features like size and location. If your **dataset is small**, you could use batch gradient descent to train your model, updating the weights based on the error calculated from the entire dataset.

Repeat until convergence

{

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

Batch Gradient Descent

- ✔ Despite its accuracy and consistency, batch gradient descent has several drawbacks.
  - The batch gradient descent method typically requires the entire training dataset in memory
  - Large datasets can result in very slow model updates or training speeds.
  - It can be slow and computationally expensive(require more computational power) when dealing with large datasets and complex models.
  - Additionally, it can get stuck in shallow local minima or saddle points, where the gradient is either very small or zero.
  - Furthermore, it does not permit online or incremental learning, meaning new data cannot be added to the training set without restarting the algorithm.

Let's understand how the Gradient descent can be derived from cost function:

1. **Cost function**

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

2. **Compute the partial derivative of the cost function with respect to θ_j**

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

3. **Update θ_j using the gradient descent update rule:**

$$\theta_j := \theta_j - \alpha \frac{\partial \theta_j}{\partial J}$$

Substituting the partial derivative, we get:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

This update rule is applied simultaneously for all Θ_j until convergence is reached. The intuition behind this process is that we update each parameter in the direction that reduces the cost function, and the size of the update is determined by the learning rate (α). The sum term represents the average contribution of each training example to the update. The process is repeated iteratively until the algorithm converges to parameter values that minimize the cost function.

y_i

x

Stochastic Gradient Descent (SGD)

- ✔ Stochastic gradient descent updates the model's parameters using **only one training example or a small batch of examples at a time.** This means the gradient and hence the **parameter update is calculated and applied after each example or small batch**, making the process more incremental.
- ✔ SGD is particularly useful when dealing with **large datasets** that cannot fit into memory. For example, the model parameters are updated incrementally as each image (or a small batch of images) is processed, enabling the **model to learn progressively without the need to load the entire dataset into memory**.
- ✔ We update the values of parameters after each example of training set. It is relatively faster than other types but its accuracy is less.

$$for\ i\ in\ range\,(m):$$

$$\theta_j = \theta_j - \alpha\,(\hat{y}^i - y^i)\,x_j^i$$

Numerical Example

Data preparation

Consider a simple 2-D data set with only 6 data points (each point has x_1, x_2), and each data point have a label value y assigned to them.

Model overview

For the purpose of demonstrating the computation of the SGD process, simply employ a linear regression model: $y = w_1\ x_1 + w_2\ x_2 + b$, where w_1 and w_2 are weights and b is the constant term. In this case, the goal of this model is to find the best value for w_1, w_2 and b, based on the datasets.

Initial Weights:

The linear regression model starts by initializing the weights w_1, w_2 and setting the bias term at 0. In this case, initiate $[w_1, w_2]$ = [-0.044, -0.042].

Dataset:

For this problem, the batch size is set to 1 and the entire dataset of [x_1, x_2, y] is given by:

x_1	x_2	y
4	1	2
2	8	-14
1	0	1
3	2	-1
1	4	-7
6	7	-8

Gradient Computation and Parameter Update

$$\omega_1^{'} = \omega_1 - \eta \frac{\partial L}{\partial \omega_1} = \omega_1 - \eta \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \omega_1} = \omega_1 - \eta \left[2(\hat{y} - y) \cdot x_1\right]$$

$$\omega_2^{'} = \omega_2 - \eta \frac{\partial L}{\partial \omega_2} = \omega_2 - \eta \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \omega_2} = \omega_2 - \eta \left[2(\hat{y} - y) \cdot x_2\right]$$

$$b^{'} = b - \eta \frac{\partial L}{\partial b} = b - \eta \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = b - \eta \left[2(\hat{y} - y) \cdot 1\right]$$

Where the η stands for the learning rate and in this model, is set to be 0.05. To update each parameter, simply substitute the value of resulting $\hat{y}$.

Use the first data point $[x_1, x_2]$ = [4, 1] and the corresponding y being 2. The $\hat{y}$ the model gave should be -0.2. Now with $\hat{y}$ and y value, update the new parameters as [0.843, 0.179, 0.222] = $[w_1', w_2', b']$. That marks the end of iteration 1.

Now, iteration 2 begins, with the next data point [2, 8] and the label -14. The estimation , $\hat{y}$ is now 3.3. With the new $\hat{y}$ and y value, once again, we update the weight as [-2.625, -13.696, 1.513]. And that marks the end of iteration 2.

Keep on updating the model through additional iterations to output [w_1, w_2, b] = [-19.021, -35.812, -1.232].

x1	x2	y
4	1	2
2	8	-14
1	0	1
3	2	-1
1	4	-7
6	7	-8

Iteration – 01

$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

$\hat{y} = \theta_1 x_1 + \theta_2 x_2$

$\alpha = 0.05$

$\theta_0 = 0$ ধরি।

$\theta_1 = -0.044$

$\theta_2 = -0.042$

$\theta_1 = \theta_1 - \alpha \frac{\delta L}{\delta \theta_1}$

$= \theta_1 - \alpha \{2(\theta_1 x_1 + \theta_2 x_2 - y_1)\} x_1$

x_1	x_2	y
4	1	2
2	8	-14
1	0	1
3	2	-1
1	4	-7
6	7	-8

$= -0.044 - 0.05\{2(-0.044 \times 4 - 0.042 \times 1 - 2) 4\}$

$= -0.044 - 0.05\{2(-0.176 - 0.042 - 2) 4\}$

$= -0.044 - 0.05\{8(-.218 - 2)\}$

$= -0.044 - 0.05 \times 8(-2.218)$

$= -0.044 + .8872$

$= 0.8432$

$\theta_2 = \theta_2 - \alpha \frac{\delta L}{\delta \theta_2}$

$= \theta_2 - \alpha \{2(\theta_1 x_1 + \theta_2 x_2 - y_1)\} x_2$

$= -0.042 - 0.05\{2(-0.044 \times 4 + -0.042 \times 1 - 2) \times 1\}$

$= -0.042 - 0.05\{2(-0.176 - 0.042 - 2)\}$

$= -0.042 - 0.05\{-.218 - 2\} \times 2$

$= -0.042 - 0.05(-2.218) \times 2$

$= -0.042 + 0.2218$

$= 0.1798$

$\theta_0 = \theta_0 - \frac{\delta L}{\delta \theta_0}$

$= 0 - \alpha \{2(\theta_1 x_1 + \theta_2 x_2 - y_1)\}$

$= -0.05\{2(-2.218)\}$

$= +0.05 * 4.436$

$= 0.2218 \approx 0.222$

$=$

Iteration - 02

$x_1 = 2, x_2 = 8,$
$y_2 = -14$

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$
$$= 0.222 + 0.843 \times 2 + 0.179 \times 8$$
$$= 3.34$$

$$\theta_1 = \theta_1 - \alpha \{2(\hat{y} - y_2)\} x_1$$
$$= 0.843 - 0.05 \{2(3.34 - (-14))\} \times 2$$
$$= 0.843 - 0.05 * 2 (17.34) * 2$$
$$= 0.843 - 3.468$$
$$= -2.625$$

$$\theta_2 = \theta_2 - \alpha \{2(\hat{y} - y_2)\} x_2$$
$$= 0.179 - 0.05 \{2(3.34 - (-14)\} \times 8$$
$$= 0.179 - 0.05 \{16 \times 17.34\}$$
$$= 0.179 - 13.872$$
$$= -13.693$$

$$\theta_0 = .222 - 0.05 \{2(3.34 + 14)\}$$
$$= .222 - 0.05 \times 2 \times 17.34$$
$$= 0.222 - 1.734$$
$$= 1.512$$

Stochastic Gradient Descent (SGD)

Advantages

- ✔ You can instantly see your model's performance and improvement rates with frequent updates.
- ✔ This variant of the steepest descent method is probably the easiest to understand and implement, especially for beginners.
- ✔ Increasing the frequency of model updates will allow you to learn more about some issues faster.
- ✔ The noisy update process allows the model to avoid local minima (e.g., premature convergence).
- ✔ Faster and require less computational power.
- ✔ Suitable for the larger dataset.

Disadvantages

- ✔ Frequent model updates are more computationally intensive than other steepest descent configurations, and it takes considerable time to train the model with large datasets.
- ✔ Frequent updates can result in noisy gradient signals. This can result in model parameters and cause errors to fly around (more variance across the training epoch).
- ✔ A noisy learning process along the error gradient can also make it difficult for the algorithm to commit to the model's minimum error.

Mini Batch Gradient Descent

- ✔ A variation on stochastic gradient descent is the mini-batch gradient descent.
- ✔ In SGD, the gradient is computed on only one training example and may result in a large number of iterations required to converge on a local minimum.
- ✔ Mini-batch gradient descent offers a compromise between batch gradient descent and SGD by splitting the training data into smaller batches. Here the data set is divided into certain number of batches of equal size so that we can update the parameters of the equation after the completion of each batch.

Say $b = 10, m = 1000.$

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$) } }

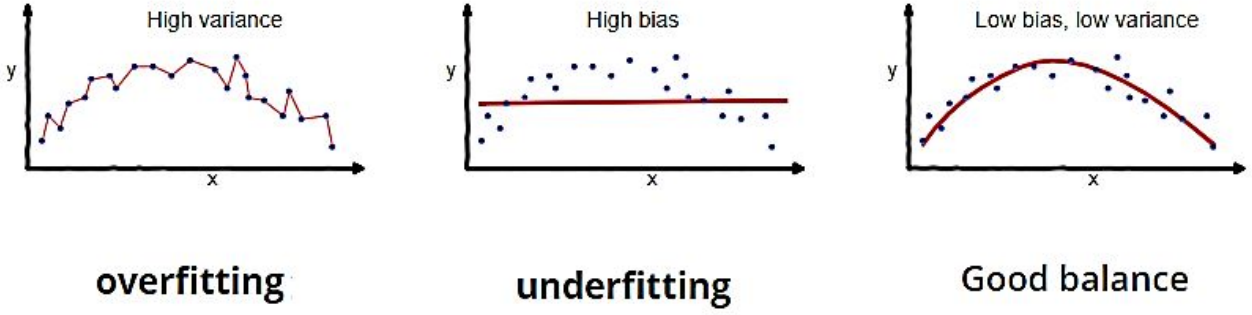
- ✔ here b is the batch size and m is number of training examples
- ✔ This has moderate accuracy and time consumption.

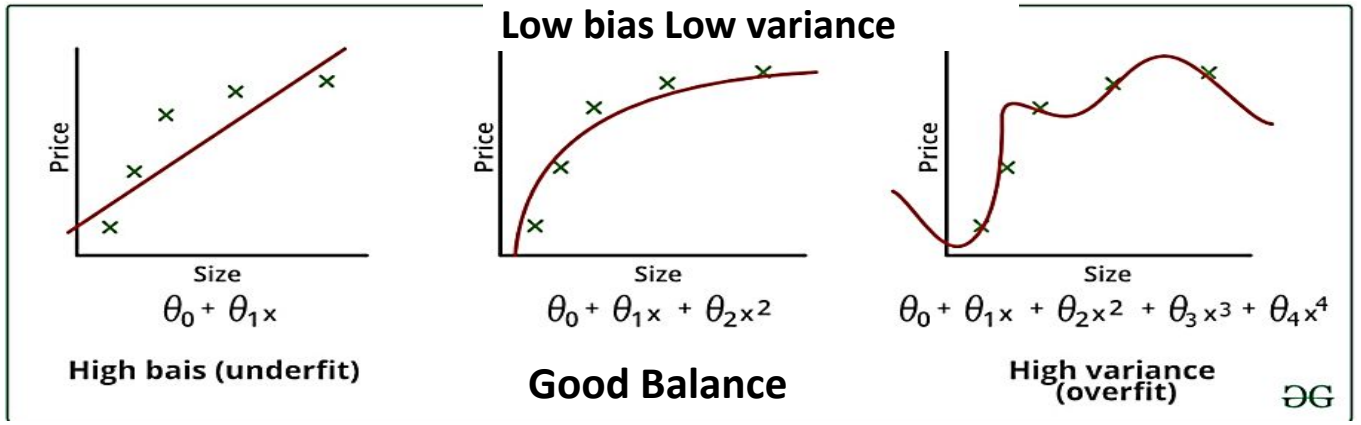
What is Bias?

- ✔ **Bias** refers to the errors which occur when we try to fit a statistical model on real-world data which does not fit perfectly well on some mathematical model. It may be defined by the differences or error occurring between the model's predicted value and the actual value.
- ✓ Let Y be the true value of a parameter, and let $\hat{Y}$ be an estimator of Y based on a sample of data. Then, the bias of the estimator $\hat{Y}$ is given by:
  - $\text{Bias}(\hat{Y}) = E(\hat{Y}) - Y$
- ✓ where $E(\hat{Y})$ is the expected value of the estimator $\hat{Y}$. It is the measurement of the model that how well it fits the data.
- ✓ The high-bias model will not be able to capture the dataset trend. It is considered as the underfitting model which has a high error rate. It is due to a very simplified algorithm.
- ✓ For example, a **linear regression** model may have a high bias if the data has a non-linear relationship.
- ✓ We can use a more complex model like **Polynomial regression** for **non-linear datasets**

Variance

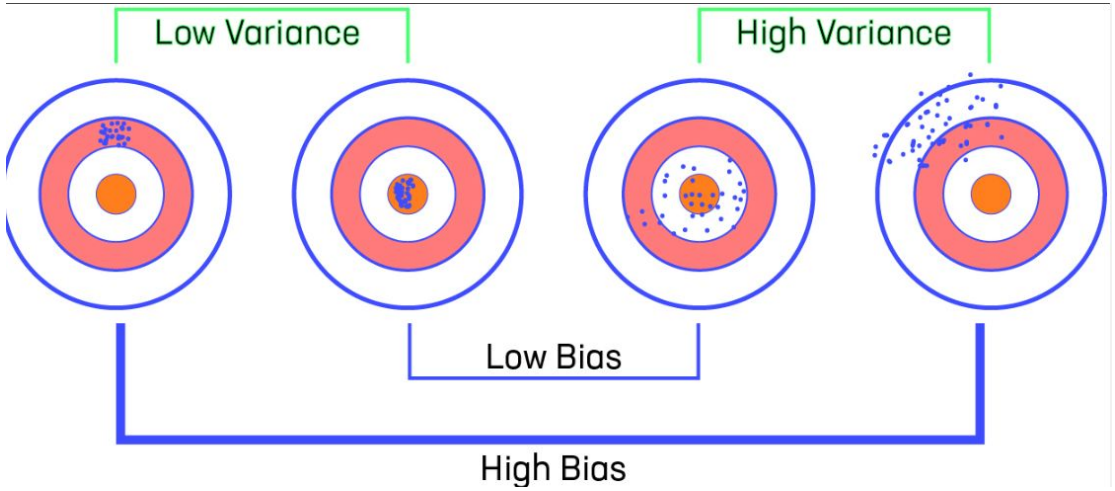
- **Variance** is the measure of spread in data from its mean position. In machine learning variance is the amount by which the performance of a predictive model changes when it is trained on different subsets of the training data. More specifically, variance is the variability of the model that how much it is sensitive to another subset of the training dataset.
  - i.e. how much it can adjust on the new subset of the training dataset.

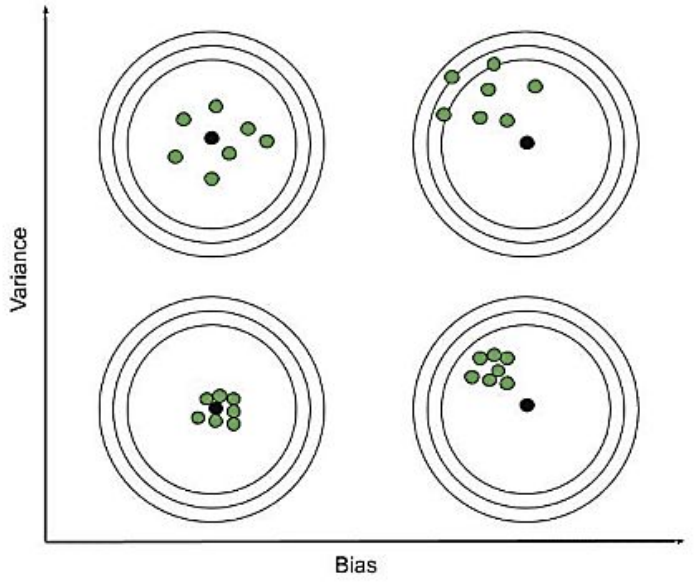




Bias vs Variance

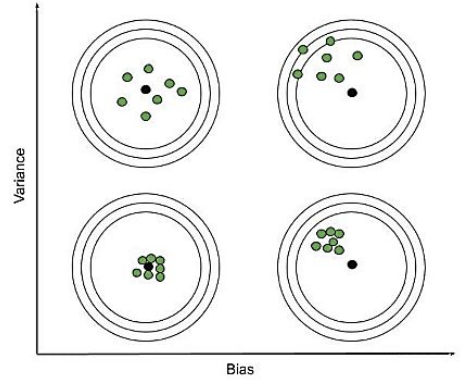
- ✔ **Low Bias:** The average prediction is very close to the target value
- ✔ **High Bias:** The predictions differ too much from the actual value
- ✔ **Low Variance:** The data points are compact and do not vary much from their mean value
- ✔ **High Variance:** Scattered data points with huge variations from the mean value and other data points.
- ✔ To make a good fit, we need to have a correct balance of bias and variance.





Bias-Variance Combinations

Bias vs Variance



Bias-Variance Combinations

- ✔ There can be four combinations between bias and variance.
- ✔ **High Bias, Low Variance:** A model with high bias and low variance is said to be underfitting.
- ✔ **High Variance, Low Bias:** A model with high variance and low bias is said to be overfitting.
- ✔ **High-Bias, High-Variance:** if the **predictions are scattered here and there then that is the symbol of high variance**, also if the **predictions are far from the target then that is the symbol of high bias**.

  A model has both high bias and high variance, which means that the model is not able to capture the underlying patterns in the data (high bias) and is also too sensitive to changes in the training data (high variance). As a result, the model will produce inconsistent and inaccurate predictions on average.

Bias vs Variance

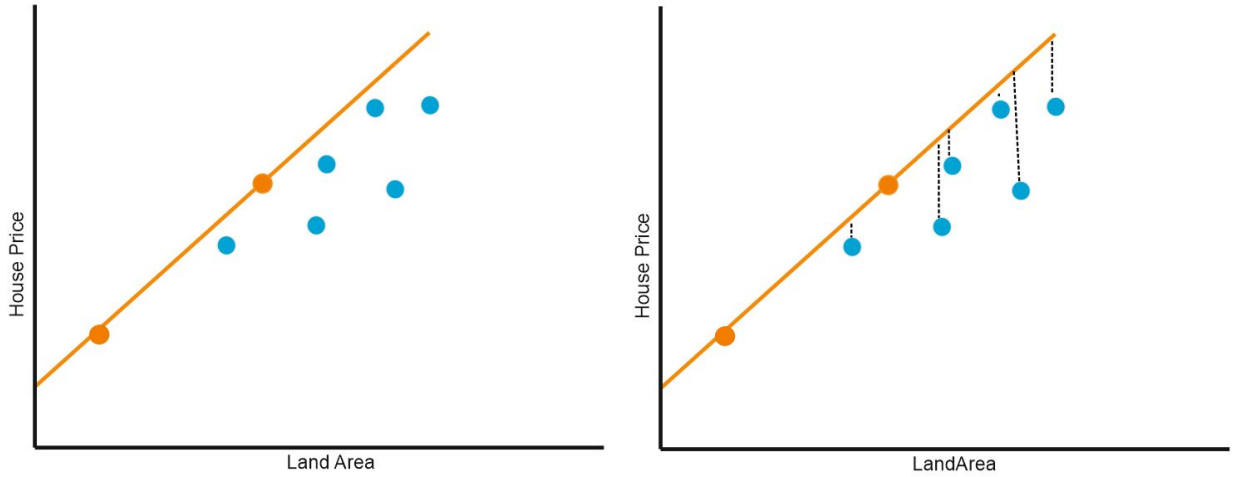
- ✔ **Low Bias, Low Variance:** A model that has low bias and low variance means that the model is able to capture the underlying patterns in the data (low bias) and is not too sensitive to changes in the training data (low variance). This is the ideal scenario for a machine learning model, as it is able to generalize well to new, unseen data and produce consistent and accurate predictions. But in practice, it's not possible. So, we trade off between Bias and variance to achieve a balanced bias and variance.
  - **A model with balanced bias and variance is said to have optimal generalization performance**. This means that the model is able to capture the underlying patterns in the data without overfitting or underfitting. The model is likely to be just complex enough to capture the complexity of the data, but not too complex to overfit the training data. This can happen when the model has been carefully tuned to achieve a good balance between bias and variance, by adjusting the hyperparameters and selecting an appropriate model architecture.
- ✔ Sometimes we need to choose between low variance and low bias. There is an approach that prefers some bias over high variance, this approach is called **Regularization**. It works well for most of the classification/regression problems.

What is Regularization?

- ✔ Regularization is a technique to **prevent the model from overfitting** by **adding extra information** to it.
- ✔ Sometimes the machine learning **model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data,** and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.
- ✔ It mainly **regularizes or reduces the coefficient of features toward zero**. In simple words, "In regularization technique, we reduce the magnitude of the features by keeping the same number of features.“

Overfitting

- ✔ if we want to predict house prices based on land area, but have only 2 data for training. **Model performs well with these training data (orange dot)**
- ✔ The problem is **Model does not perform well with the test data (**blue dot**)** and its distribution is like the blue dot below



- ✔ Figure above shows that if we have a regression line in orange, and want to predict test data like the blue dot above, the estimator has a very large variance even though it has a small bias value, and The regression model formed is very good for the train data but bad for the test data. When the phenomenon occurs, the model built is subject to overfitting problems.

Why regularization?

- ✔ Regularization is often used as a solution to the **overfitting** problem in Machine Learning. Overfitting occurs when a model is too complex and learns noise in the training data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. Underfitting occurs when a model is too simple to capture underlying data patterns. Regularization provides a means to find the optimal balance between these two extremes.
- ✔ The primary goal of regularization is to reduce the model's complexity to make it more generalizable to new data, thus improving its performance on unseen datasets.

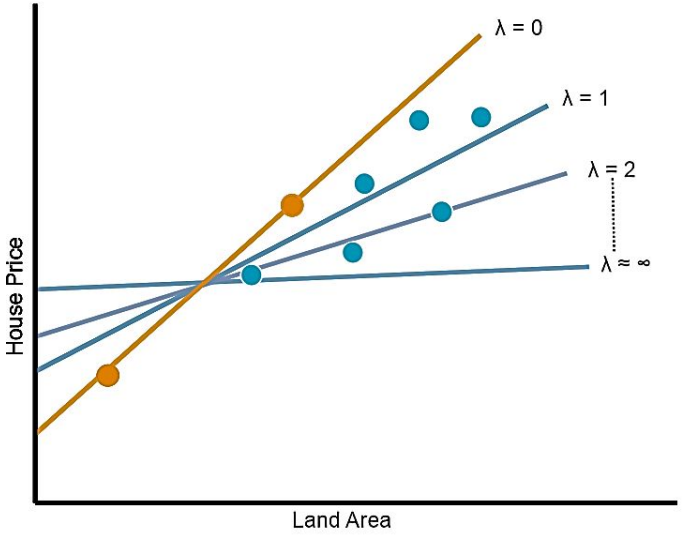
Techniques of Regularization

- ✔ There are mainly two types of regularization techniques, which are given below:
- ✔ **Ridge Regression**
- ✔ **Lasso Regression**

Ridge Regression

- ✔ Ridge regression is a **model tuning method** that is used to **analyze any data that suffers from multicollinearity**. Multicollinearity occurs when independent variables in a regression model are highly correlated, which can lead to unreliable and unstable estimates of regression coefficients.
- ✔ Ridge regression is a procedure for eliminating the bias of coefficients and reducing the mean square error by **shrinking the coefficients of a model towards zero** in order to solve problems of overfitting or multicollinearity that are normally associated with ordinary **least squares regression.**
- ✔ The cost function is altered by **adding the penalty term** to it where this penalty term is governed by a parameter denoted as lambda (λ), thus **lowering the variance** of the model and increasing its stability as well as the robustness of the prediction made by the model.
- ✔ It is also called as **L2 regularization**.

Ridge Regression



- The equation for the cost function in ridge regression will be:

$$\sum_{i=1}^{M}(h_\theta(x_i)-y_i)^2=\sum_{i=1}^{M}\left(\sum_{j=0}^{n}\theta_j x_{ij}-y_i\right)^2+\lambda\sum_{j=0}^{n}{\theta_j}^2$$

- In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
- As we can see from the above equation, if the values of **λ is zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model. A **general linear or polynomial regression will fail if there is high collinearity between the independent variables,** so to solve such problems, Ridge regression can be used.
- As we increase the value of λ, causes the value of the coefficient to tend towards zero.
- The greater the value of λ (lambda) the regression line will be more horizontal, so the coefficient value approaches 0.

Ridge Regression

- ✔ The basic difference between **Linear regression** and **Ridge regression** is, in case of linear regression we try to reduce the **cost function**. However, in **Ridge Regression** we add 2 more parameters to the cost function $\lambda * (Slope)^2$.

- ✔ The formula for **Ridge Regression** is $\sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 + [\lambda * (Slope)^2]$

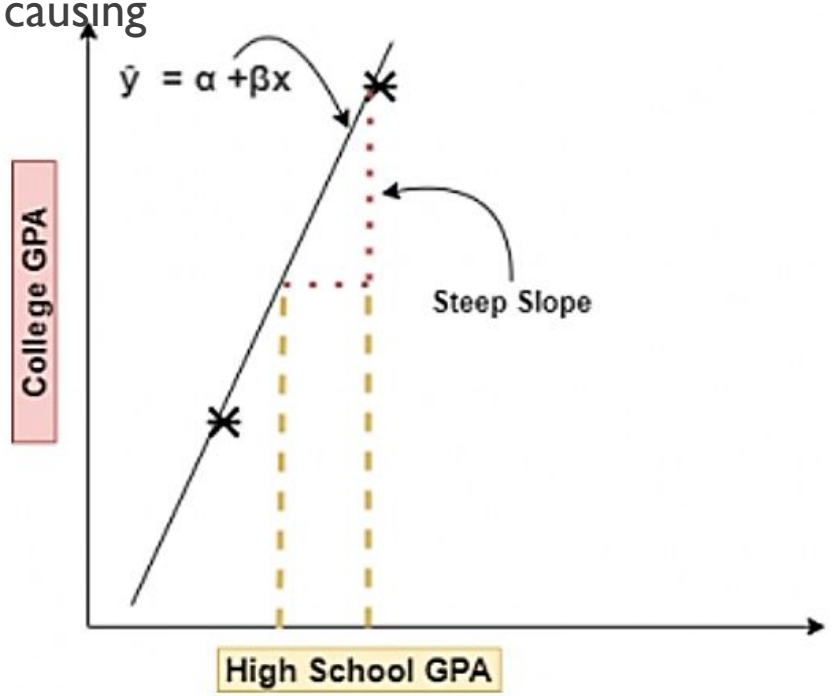
- ✔ Unlike **Linear Regression** where we used to reduce only the **cost function**, in this case we will try to reduce the whole function which includes the 2 parameters also

$$\sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 + [\lambda * (Slope)^2]$$

Iteration 01

- ✔ To understand the significance of these parameters let's give an example of **overfitting condition.**

- ✔ If you notice the graph, the slope that we have is very steep. This means with the unit increase in the X- axis, there will be a comparatively larger increase in Y-axis.
- ✔ Also, we have the best fit line aligning perfectly with the training data points causing an overfitting condition and making value for the cost function as 0.
- ✔ To counter all these problems, we use the **Ridge Regression** where the parameters $\lambda * (Slope)^2$ are used to penalize the steep **slope**.
- ✔ Let's understand this mathematically below:



- ✔ $\sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 = 0$, lue to the **overfitting condition.**
- ✔
- ✔ **Let λ=1**

Iteration 01

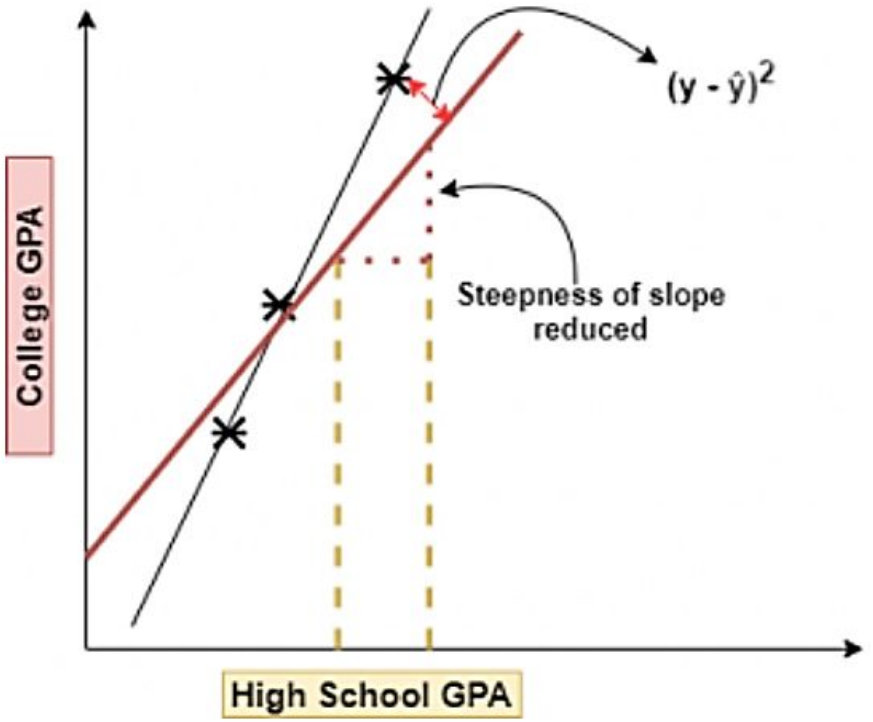
- ✔ **Slope** = High steep slope(Let's take 1.5 for our understanding purpose)
- ✔ Substituting, the values to the whole equation **= 0 + (1) * $(1.5)^2$ = 2.25**
- ✔ Now, the **cost function** that we got is 2.25.
- ✔ One thing we need to notice here is, in case of **linear regression**, we stopped when the cost function was zero because our best fit line perfectly aligned with the data points.
- ✔ However, in **Ridge Regression** we will not stop until we reach a particular value which would yield us a **generalized model.**

Iteration 02

- ✔ Now let's take another best fit line and calculate the cost function:
- ✔ In the graph, the value for **(y – ŷ)** which was zero in the overfit condition will now be a smaller value as our best fit line is not passing exactly through the data points.
- ✔ Also, if you notice, the steepness of the curve has reduced a bit, which means the slope has also reduced by a smaller value.
- ✔ Substituting the individual values to the cost function formula for **Ridge regression:**

$$: \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 + [\lambda * (Slope)^2]$$

$$\sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 = small\ value$$



Iteration 02

- ✔ **Slope** = Less steep slope (Let's take 1.3)
- ✔ Substituting, the values to the whole equation **= small value + (1) * $(1.3)^2$ => small value +1.69**
- ✔ The value that we will get will be less that the previous **cost function** output of **2.25** because we have a reducing slope in this case.
- ✔ So, as we got a smaller **cost function** now compared to the previous iteration, we will select the current one as our best fit line.
- ✔ We are basically **penalizing** those features which are having **steeper slopes**. This is the basic motive of **Ridge Regression.**
- ✔ One thing to note here is, we will usually choose the λ to be a small and as it increases, the slope of our best fit line also approaches towards 0, but never becomes 0.

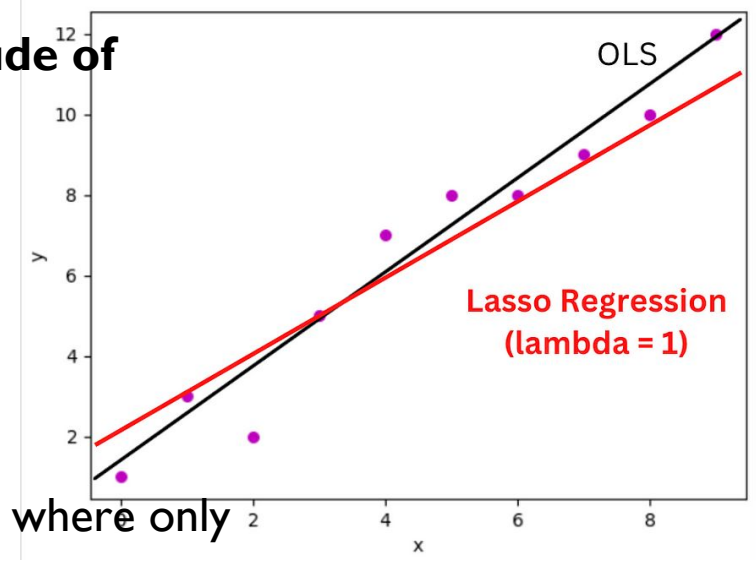
Lasso Regression

- ✔ Now that we have discussed about **Ridge Regression** and how it can be used to overcome **overfitting condition**, let's understand another technique which is called **Lasso Regression.** In this technique the overall slope of the best fit line would move towards zero over the period of multiple iterations. It might also reach zero.

Lasso Regression

- ✔ Lasso regression is another regularization technique to reduce the complexity of the model. It stands for **Least Absolute Shrinkage and Selection Operator.**
- ✔ It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square. Since it takes absolute values, hence, it **can shrink the slope to 0**, whereas **Ridge Regression can only shrink it near to 0.**
- ✔ It is also called as **L_1 regularization.** The equation for the cost function of Lasso regression will be:
- ✔ **Residual Sum of Squares + λ * (Sum of the absolute value of the magnitude of coefficients)**

$$\sum_{i=1}^{M}(h_\theta(x_i)-y_i)^2=\sum_{i=1}^{M}\left(\sum_{j=0}^{n}\theta_j x_{ij}-y_i\right)^2+\lambda\sum_{j=0}^{n}|\theta_j|$$



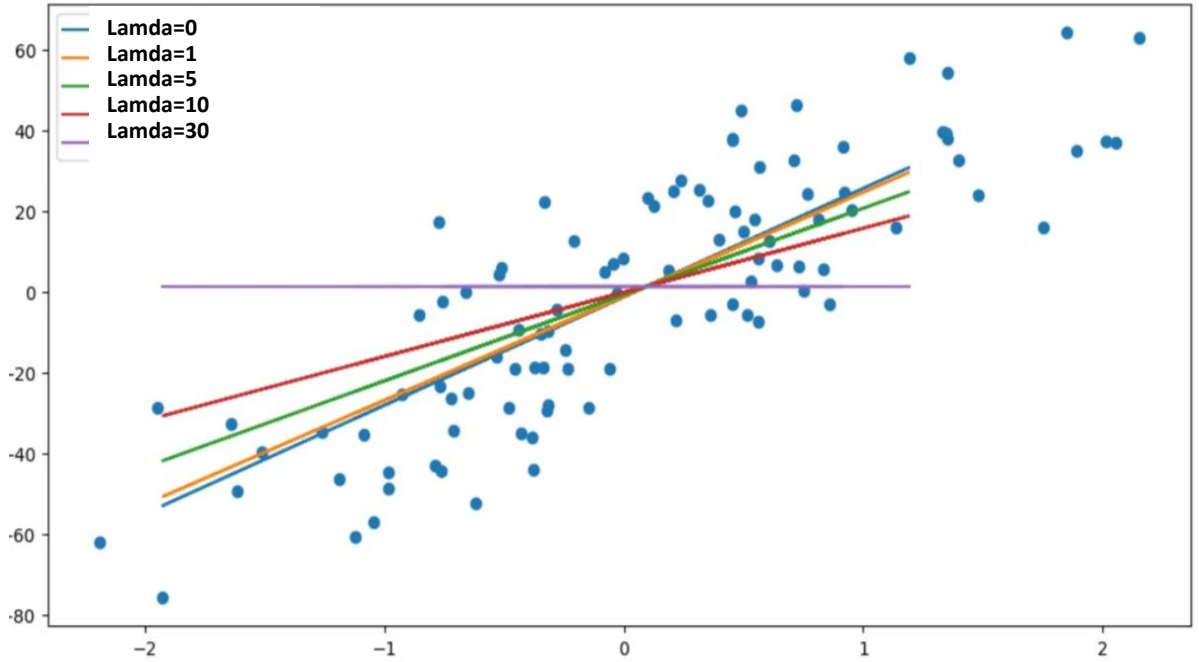
- ✔ Where, λ denotes the amount of shrinkage, controls the strength of the L_1 penalty
- ✔ λ = 0 implies all features are considered and it is equivalent to the linear regression where only the residual sum of squares is considered to build a predictive model
- ✔ λ = ∞ implies no feature is considered i.e, as λ closes to infinity it eliminates more and more features
- ✔ The bias increases with increase in λ and variance increases with decrease in λ

Lasso Regression

- ✔ LASSO regression encourages models with **fewer parameters** where some coefficients are forced to be exactly zero.
- ✔ This feature makes LASSO particularly useful for **feature selection**, as it can automatically identify and discard irrelevant or redundant variables.
- ✔ This particular type of regression is well-suited for models showing high levels of **multicollinearity** or when you want to automate certain parts of model selection, like variable **selection/parameter elimination.**
- ✔ Hence, the Lasso regression can help us to **reduce the overfitting** in the model as well as the feature selection.

- ✔ Lowering the **“Lambda” value towards zero makes Lasso Regression behave like standard “Linear Regression,”** potentially leading to overfitting.
- ✔ Conversely, **increasing the “Lambda” value significantly can result in underfitting**.
- ✔ The **“Lambda”** value must always be **non-negative and greater than** zero.
- ✔ When visualizing the graph, it’s evident that as we i**ncrease the Lambda value, the regression line begins to decrease**.
- ✔ The **coefficients also decrease as the Lambda value increases**.
- ✔ At an **alpha value of 30, the slope of the line becomes zero, indicating that “X” does not significantly contribute to calculating “Y,” resembling a case of underfitting**.
- ✔ Unlike Ridge Regression, where coefficients never reach zero but tend towards it, in Lasso Regression, increasing the **Lambda value can drive coefficients to become exactly zero**.

high-dimensional data or when there is multicollinearity among the



How does Regularization Work?

- ✔ Regularization works by adding a penalty term to the cost function of a linear regression model, which reduces the magnitude of the coefficients (in regression models) or weights (in neural networks).
- ✔ By adding regularization, you can prevent the model from fitting too closely to the training data and reduce the variance of the model. However, regularization also introduces some bias to the model, as it shrinks the coefficients towards zero or a constant.
- ✔ Let's consider the simple linear regression equation:

$$y= \theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_3+\cdots+\theta_nx_n$$

- ✔ In the above equation, Y represents the value to be predicted
- ✔ X_1, X_2, …X_n are the features for Y.
- ✔ $\theta_0,\theta_1,\ldots..\theta_n$ are the weights or magnitude attached to the features, respectively.

How does Regularization Work?

- ✔ Linear regression models try to optimize the coefficients to minimize the cost function. The equation for the cost function for the multiple linear model:

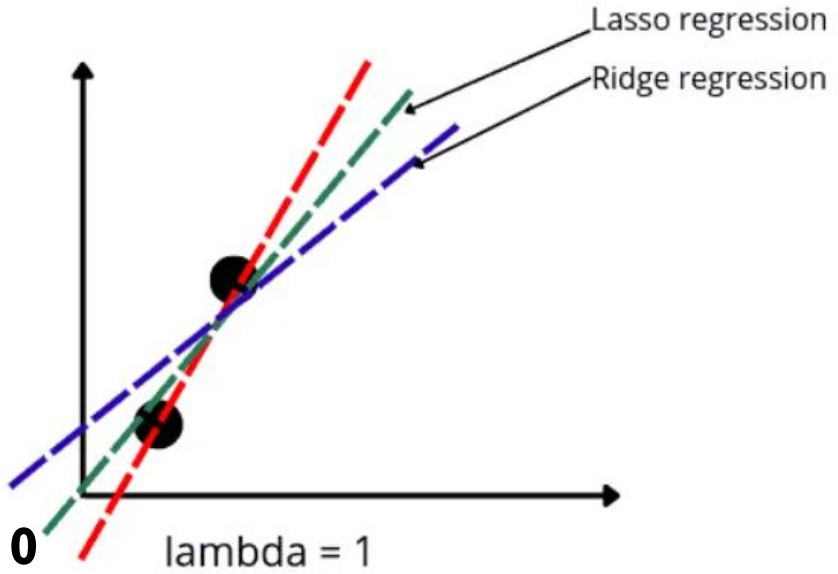
$$\sum_{i=1}^{M}(\hat{y}_i - y_i)^2 = \sum_{i=1}^{M}\left(\sum_{j=0}^{n}\theta_j x_{ij} - y_i\right)^2$$

- ✔ The equation for the cost function in ridge regression will be:

$$\sum_{i=1}^{M}(h_\theta(x_i) - y_i)^2 = \sum_{i=1}^{M}\left(\sum_{j=0}^{n}\theta_j x_{ij} - y_i\right)^2 + \lambda\sum_{j=0}^{n}{\theta_j}^2$$

- ✔ In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
- ✔ As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.

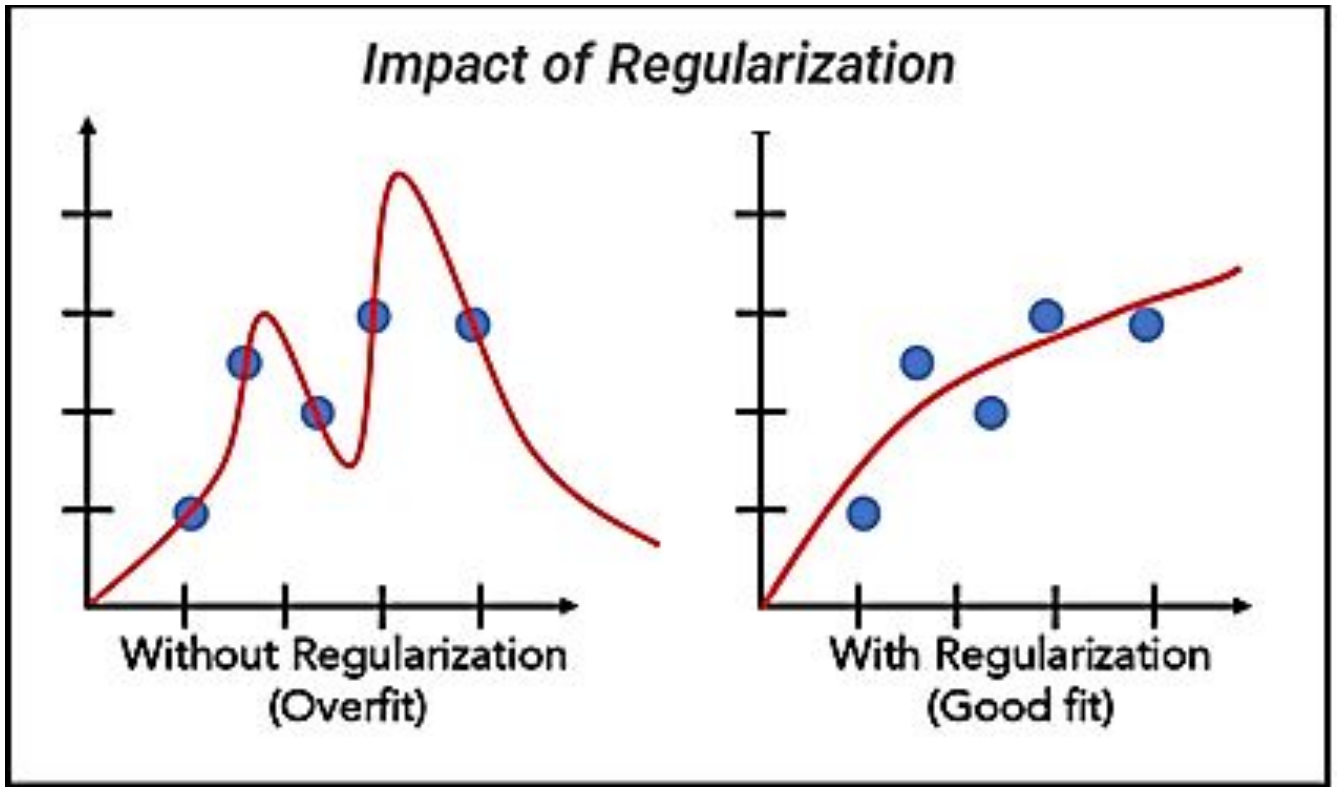
How does Regularization Work?



- ✔ The equation for the cost function of Lasso regression will be:

$$\sum_{i=1}^{M}(h_\theta(x_i)-y_i)^2=\sum_{i=1}^{M}\left(\sum_{j=0}^{n}\theta_j x_{ij}-y_i\right)^2+\lambda\sum_{j=0}^{n}|\theta_j|$$

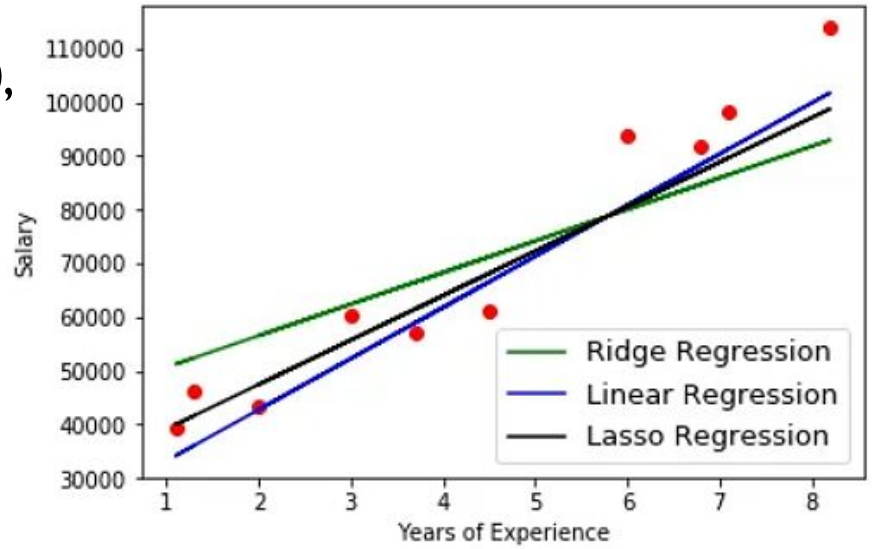
- ✔ Since it takes absolute values, hence, it **can shrink the slope to 0**
- ✔ This particular type of regression is well-suited for models showing high levels of **multicollinearity** or when you want to automate certain parts of model selection, like variable **selection/parameter elimination.**
- ✔ Hence, the Lasso regression can help us to **reduce the overfitting** in the model as well as the feature selection.



✔ So using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.

The big difference between Lasso and Ridge Regression

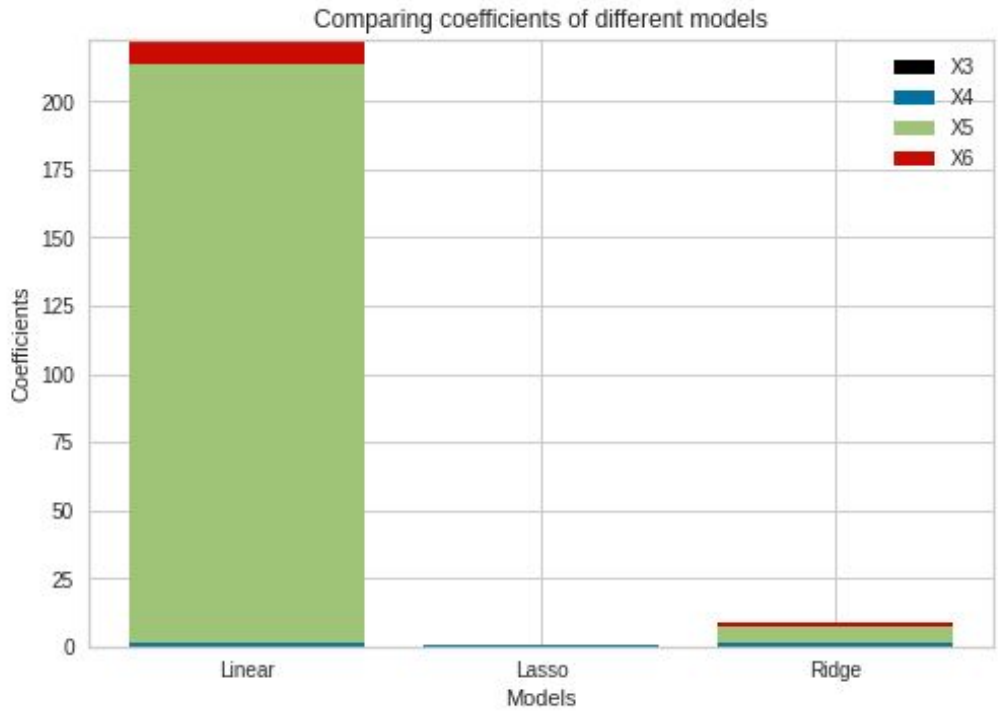
- ✔ Lasso Regression deals with the penalty — not on the squared value of the parameter, but the absolute value.
- ✔ So essentially meaning that your data can have 20 variables but you can exclude the ones that don't impact your regression equation from the penalty (because the absolute penalty can take a value of 0, which is not possible in Ridge Regression)
- ✔ Lasso Regression is good when you want to penalize your data to not overfit the training data and also if you want to exclude some unnecessary variables from your model. (If you only want to avoid overfitting and want your model has only the required variables)
- ✔ **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients.
- ✔ **Lasso regression** helps to reduce the overfitting in the model as well as feature selection.



big difference is that **Lasso Regression** can exclude useless variables from equations.

Comparison of Theta Coefficients

✔ Inspecting the coefficients, we can see that Lasso and Ridge Regression had shrunk the coefficients, and thus the coefficients are close to zero. On the contrary, Linear Regression still has a substantial value of the coefficient for the X5 column.



Comparison of Theta Coefficients

Thank You

Loss function for multiple linear regression

MSE, $L = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$

For m samples

x_1	x_2	y
x_{11}	x_{12}	y_1
x_{21}	x_{22}	y_2
x_{31}	x_{32}	y_3

$$= \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \theta_3 x_{i3} + \cdots \theta_n x_{in} - y_i)^2$$

For n Features

$$= \sum_{i=1}^{m}(\quad)^2$$

Now for simplicity For 2 Features and for $m = 2$ samples

$$L = \frac{1}{2}\left[(\theta_0 + \theta_1 x_{11} + \theta_2 x_{12} - y_1)^2 + (\theta_0 + \theta_1 x_{21} + \theta_2 x_{22} - y_2)^2\right]$$

let $h_\theta(x) = \hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

$\hat{y}_1 = \theta_0 + \theta_1 x_{11} + \theta_2 x_{12}$

$\hat{y}_2 = \theta_0 + \theta_1 x_{21} + \theta_2 x_{22}$

x_1	x_2	y
8.1	93	3.2
9.5	95	3.5

Again,

$$L = \frac{1}{2}\cdot\frac{1}{2}\left[(\theta_0 + \theta_1 x_{11} + \theta_2 x_{12} - y_1)^2 + (\theta_0 + \theta_1 x_{21} + \theta_2 x_{21} - y_2)^2\right]$$

$$\frac{\delta L}{\delta \theta_0} = \frac{1}{2}\cdot\frac{1}{2}\left[2(\theta_0 + \theta_1 x_{11} + \theta_2 x_{12} - y_1)(1) + 2(\theta_0 + \theta_1 x_{21} + \theta_2 x_{21} - y_2)(1)\right]$$

$$= \frac{1}{2}\cdot\frac{1}{2}\left[2(\hat{y}_1 - y_1) + 2(\hat{y}_2 - y_2)\right]$$

$$= \frac{1}{2}\cdot\frac{2}{2}\left[(\hat{y}_1 - y_1) + (\hat{y}_2 - y_2)\right]$$

Now for m samples

$$\frac{\delta L}{\delta \theta_0} = \frac{1}{2}\cdot\frac{2}{m}\left[(\hat{y}_1 - y_1) + (\hat{y}_2 - y_2) + (\hat{y}_3 - y_3) + \cdots + (\hat{y}_m - y_m)\right]$$

$$\frac{\delta m}{\delta \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

$$\theta_0 = \theta_0 - \alpha \frac{\delta L}{\delta \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\delta L}{\delta \theta_1}$$

$$\theta_2 = \theta_2 - \alpha \frac{\delta L}{\delta \theta_2}$$

$$\vdots$$

$$\theta_j = \theta_j - \alpha \frac{\delta L}{\delta \theta_j} \quad [j = 0, 1, 2, \cdots n]$$

Again,

$$L = \frac{1}{2} \cdot \frac{1}{2} \sum_{i=1}^{n=2} (\hat{y}_i - y_i)^2$$

$$\Sigma_{i=1}^{M} (y_i - y'_i)^2 = \Sigma_{i=1}^{M} (y_i - \Sigma_{j=0}^{n} \beta_j * Xij)^2$$

$$= \frac{1}{2} \cdot \frac{1}{2} \left[(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \right]$$

$$= \frac{1}{2} \cdot \frac{1}{2} \left[(\theta_0 - \theta_1 x_{11} - \theta_2 x_{12} - y_1)^2 + (\theta_0 - \theta_1 x_{21} + \theta_2 x_{22} - y_2)^2 \right]$$

$$\frac{\delta L}{\delta \theta_1} = \frac{1}{2} \frac{1}{2} \left[2 (\hat{y}_1 - y_1)(x_{11}) + 2 (\hat{y}_2 - y_2)(x_{21}) \right]$$

11, 21, 31, ... m1

$$\frac{\delta L}{\delta \theta_1} = \frac{1}{2} \cdot \frac{2}{m} \left[(\hat{y}_1 - y_1)(x_{11}) + (\hat{y}_2 - y_2)(x_{21}) + (\hat{y}_3 - y_3)(x_{31}) + \cdots + (\hat{y}_n - y_n)(x_{m1}) \right]$$

$$\therefore \frac{\delta L}{\delta \theta_1} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)(x_{i1})$$

$\theta_1 \rightarrow$ Value of 1st-

$$\therefore \frac{\delta L}{\delta \theta_2} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)(x_{i2})$$

x_1	x_2	y
x_{11}	x_{12}	y_1
x_{21}	x_{22}	y_2
x_{31}	x_{32}	y_3
⋮	⋮	⋮
x_{m1}	x_{m2}	y_m

$$\therefore \frac{\delta L}{\delta \theta} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i) x_{in}$$ column

x_{i1} means

~~Three steps:~~

Now, $\theta_j = \theta_j - \alpha \frac{\delta L}{\delta \theta_j}$ where, $j = 0, 1, 2, \cdots n$

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i) x_{ij}$$

$$\begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ \vdots \\ x_{m1} \end{bmatrix}$$ 1st- column

x_1	x_2 ⋯	x_n	y
x_{11}	x_{12}	x_{1n}	y_1
x_{21}	x_{22}	x_{2n}	y_2
x_{31}	x_{32}	x_{3n}	y_3
⋮		⋮	⋮
x_{m1}	x_{m2}	x_{mn}	y_m

~~Three steps:~~

Finally,

Repeat until convergence

{

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i) x_{ij}$$

}

where, $j = 0, 1, 2, \cdots n$

Time : Date : / /

or we can write,

Repeat until convergence

{

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{ij}$$

}

$\theta_j = 0, 1, 2, \cdots n$

{ Repeat until convergence

$$\theta_0 = \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 = \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{i1}$$

$$\theta_2 = \theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{i2}$$

$$\vdots$$

$$\theta_n = \theta_n - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{in}$$

}

Now, update $\theta_1, \theta_2 \cdots$

Steps 1: Taking random values, $\theta_0 = 0, \theta_1 = 1, \theta_2 = 1$.

Steps 2: epoch = 100, $\alpha = 0.1$.

[for 2 rows and 2 column
$m = 2$
$n = 2$]

Algorithm

Below is the algorithm for the Least Square Solution using Coordinate Descent.

Notice, $r + x_j w_j$ is nothing but $(y_i - \sum_{k \neq j}^{p} x_{ik} w_k)$.

- Normalize X
- Initialize `w` with zeros (or randomly)
- LOOP O: Until Convergence
  - $r := \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{p} x_{ij} w_j \right)$
  - LOOP P: For j = 1, 2, ... p
    - $r_{-j} = r + x_j w_j$
    - $w_j = \sum_{i=1}^{N} x_{ij} r_{-j}$
    - $r = r - x_j w_j$