Linear Regression

Linear Regression

Linear regression is also a type of **supervised machine-learning algorithm** that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets. It computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation with observed data. It predicts the continuous output variables based on the independent input variable.

Goal of Linear Regression

The main goal of linear regression is to find the values of **m (slope)** and **c (y-intercept)** that define **the best-fit line**. Once we have these values, we can:

- Understand the relationship between x and y.
- Make predictions about y for any given value of x.

For example:

- If **m>0**, it means there's a positive relationship (as x increases, y also increases).
- If **m<0**, it means there's a negative relationship (as x increases, y decreases).

Linear regression assumes that the relationship between x and y is linear. It means the trend can be represented by a straight line.

Working of Linear Regression Prediction

A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the bias term (also called the intercept term).

Equation:

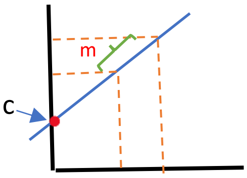
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Where,

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the i^{th} feature value.
- θ_j is the j^{th} model parameter (including the bias term θ and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

It is similar to the linear equation y=mx+c.

Where m = θ_1 ie. Slope, c= θ_0 ie. y-intercept.



This can be written much more concisely using a vectorized form,

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

Where,

- θ is the model's parameter vector, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- x is the instance's feature vector, containing x_0 to x_n , with x_0 always equal to 1.
- $\theta \cdot x$ is the dot product of the vectors θ and x, which is of course equal to $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$.
- h is the hypothesis function, using the model parameters θ.

NOTE: In Machine Learning, vectors are often represented as column vectors, which are 2D arrays with a single column. If θ and x are column vectors, then the prediction is $\hat{y} = \theta^T x$, where θ^T is the transpose of θ (a row vector instead of a column vector) and $\theta^T x$ is the matrix multiplication of θ^T and x. It is of course the same prediction, except that it is now represented as a single-cell matrix rather than a scalar value.

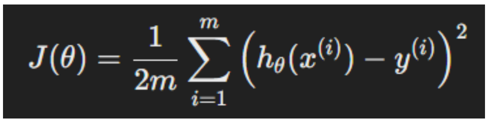
Best Fit Line

Among all possible lines you could draw through the data, linear regression finds the one that **minimizes the errors** (the gaps between the line and the points). This is called the line of best fit. By changing the Value of θ_0 & θ_1, we will get best fit line.

Cost Function

Error= y− ŷ i.e predictions minus true values is known as **loss function** (Refers to the error associated with a single training example. It measures how well the model predicts for that specific instance). Now, a **cost function** aggregates loss function and gives a single number representing the **model's performance.**

Equation:



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

The cost function in linear regression is a crucial metric that measures how well the model's predictions match the actual data. It quantifies the error between the predicted values and the actual values, presenting this error as a single real number.

Objective: The goal of linear regression is to find the parameters (values of θ_1, θ_2, ⋯, θ_n) that minimize the cost function J(θ).

Optimization Techniques

Optimization techniques for Linear Regression are methods used to minimize the cost function by finding the optimal values of model parameters (weights and bias) that result in the best fit for the data. Some Optimization Techniques are

- Gradient Descent
- OLS (Ordinary Least Square)
- Regularization Techniques

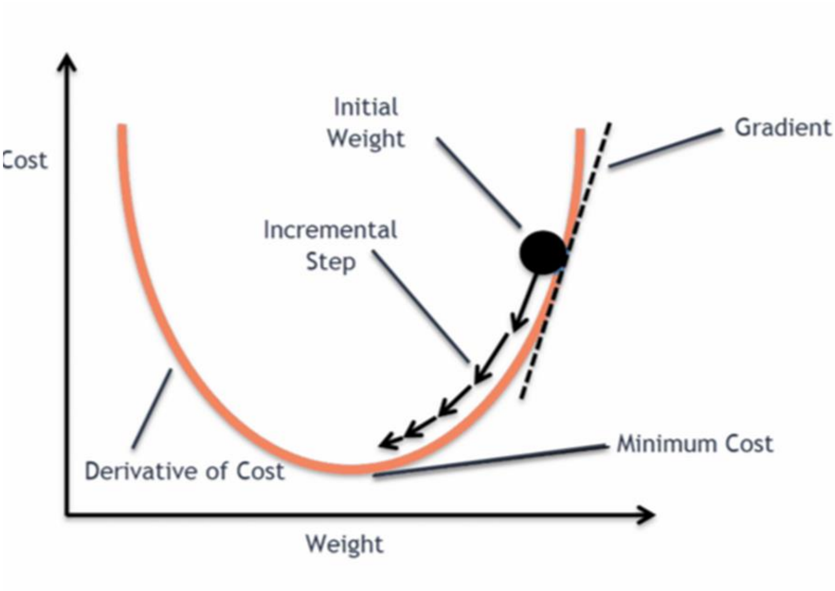
Gradient Descent

Gradient descent is the backbone of the learning process for various algorithms, including linear regression, logistic regression, support vector machines, and neural networks which serves as a fundamental optimization technique to minimize the cost function of a model by iteratively adjusting the model parameters to reduce the difference between predicted and actual values, improving the model's performance.

Gradient Descent in Linear Regression

Gradient descent minimizes the Mean Squared Error (MSE) which serves as the loss function to find the best-fit line. Gradient Descent is used to iteratively update the weights (coefficients) and bias by computing the gradient of the MSE with respect to these parameters.

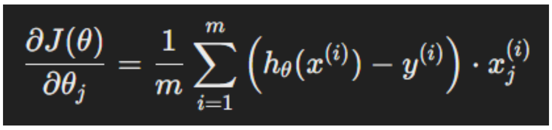
Since MSE is a convex function **gradient descent guarantees convergence to the global minimum if the learning rate is appropriately chosen.**



How Gradient Descent Works:

1. **Initialize Parameters:**
    - Start with some initial values for $\theta_0, \theta_1, \cdots, \theta_n$ usually small random values or zeros.
2. **Compute the Gradient:**
    - The **gradient** of the cost function is a vector of partial derivatives of $J(\theta)$ with respect to each parameter θ_j .
    - The gradient indicates the direction and steepness of the slope of the cost function. It helps us understand how to adjust the parameters to reduce the cost.

The gradient of the cost function with respect to θj is:

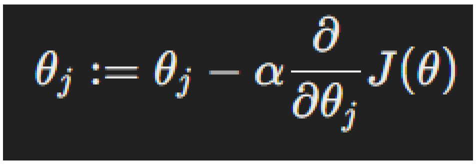


$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)}$$

where $x_j^{(i)}$ is the value of the j-th feature in the i-th training example.

3. **Update Parameters:**
   - Use the gradient to update the parameters θ$_1$, θ$_2$, ⋯, θ$_n$ in the direction that reduces the cost function.
     1. **Negative Gradient**: When the gradient is negative, it means the slope is pointing downward. To reduce the cost function (error), the parameters need to be increased (moved to the right) to reach the minimum.
     2. **Positive Gradient**: When the gradient is positive, the slope is upward. To reduce the cost, the parameters should be decreased (moved to the left) to approach the minimum.
   - The update rule is given by:

Equation:



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

where:

- α is the **learning rate**, a small positive number that controls the size of the steps taken toward minimizing the cost function.
- θj is updated by moving in the direction opposite to the gradient, which reduces the cost.

4. **Repeat:**
   - Repeat the process (compute the gradient and update parameters) for multiple iterations until the parameters converge to values that minimize the cost function. This is done until the gradient becomes very small, indicating that we have reached or are close to the minimum of the cost function.

Key Points About Gradient Descent:

- **Learning Rate**:
  - The learning rate α controls how large or small the steps are in each iteration.

  - If α is too small, convergence will be slow, and it may take many iterations to find the minimum.
  - If α is too large, gradient descent can overshoot the minimum and may not converge at all.

- **Global vs. Local Minimum**:

  - For **linear regression**, the cost function J(θ) is convex, which means it has only one global minimum (no local minima). Therefore, gradient descent will always converge to the optimal solution, provided the learning rate is appropriate.

Visualizing Gradient Descent

Imagine the cost function J(θ) as a hilly terrain, where you are standing on a hilltop. Your goal is to reach the bottom (global minimum). Gradient Descent works like this:

- The gradient tells you the direction of the steepest descent (the direction that decreases the cost the most).
- The learning rate controls how big a step you take in that direction.
- With each step, you move closer to the bottom, and the slope gets less steep as you approach the minimum. You continue this until the slope is almost flat, meaning you've reached or are close to the minimum.

How Gradient Descent Minimizes Cost Function in Linear Regression:

1. Initially, the model's parameters θ will predict values far from the actual values, leading to a large cost.
2. As Gradient Descent iterates, the parameters are updated to reduce the error.
3. After each iteration, the cost function's value decreases, and the line that fits the data (in the case of linear regression) becomes closer to the actual data points.
4. Eventually, the parameters reach the values that minimize the cost function, and the line fits the data with minimal error.

Types of Gradient Descent

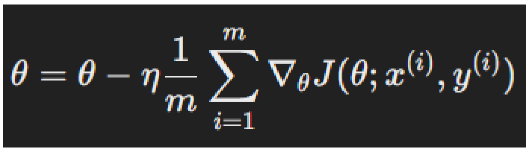
1. Batch Gradient Descent (BGD)

- **Description**: In batch gradient descent, the model computes the gradient of the cost function with respect to the parameters using the entire training dataset before updating the parameters. It ensures the direction of the update is accurate since it's calculated based on all data points.

- **Working**:

  - For each iteration, you calculate the gradient by taking the average of all the individual gradients (i.e., using all training examples) and update the parameters.
  - This method is computationally expensive for large datasets because it requires passing through the entire dataset before each update.

- **Advantages**:
  - Guaranteed to converge towards the global minimum (for convex cost functions) because it's calculated using all data points.
- **Disadvantages**:
  - Slower as it computes gradients over the entire dataset in every iteration, making it less suitable for large datasets.
- **Update Rule**:



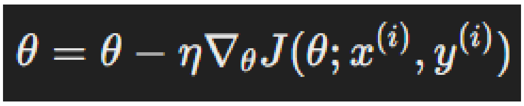
$$\theta = \theta - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

Where,

- θ is the parameter vector
- m is the total number of training examples
- η is the learning rate
- J(θ) is the cost function

2. Stochastic Gradient Descent (SGD)

- **Description**: Stochastic gradient descent updates the parameters for every training example one at a time rather than the entire dataset.
- **Working**:
  - Instead of averaging the gradient over all examples, SGD approximates it using one example at a time, randomly selected from the training set.
  - This randomness introduces noise in the gradient calculation, making it possible to escape from local minima.
- **Advantages**:
  - Faster and can handle larger datasets since it performs updates more frequently (one example at a time).
  - Can potentially escape local minima due to its noisier updates.
- **Disadvantages**:
  - Can be very noisy and might not converge as smoothly as batch gradient descent.
  - Often oscillates around the minimum, which may require more fine-tuning of the learning rate or decay schedules.
- **Update Rule**:



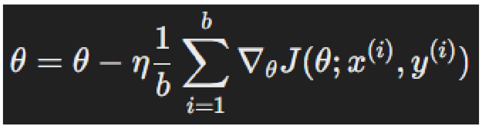
$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

Where:

- θ is updated for every single training example iii.

3. Mini-Batch Gradient Descent

- **Description**: Mini-batch gradient descent is a combination of batch and stochastic gradient descent. It splits the training dataset into small batches (mini-batches) and performs updates based on the average gradient of each mini-batch.
- **Working**:
  - Instead of using the entire dataset or just one example, mini-batch gradient descent uses small random subsets of the data (mini-batches) to update parameters.
  - Common mini-batch sizes are between 32 and 256 examples, depending on the dataset size and model complexity.
- **Advantages**:
  - Efficient: Reduces variance in the gradient estimates, leading to more stable convergence compared to SGD.
  - Faster than batch gradient descent because it updates more frequently.
  - Can take advantage of parallelism (use of vectorized computations) on modern hardware (e.g., GPUs).
- **Disadvantages**:
  - Still requires tuning the batch size and learning rate.
- **Update Rule**:



$$\theta = \theta - \eta \frac{1}{b} \sum_{i=1}^{b} \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

Where, b is the mini-batch size.

Ordinary Least Squares (OLS)

Ordinary Least Squares (OLS) is one of the most widely used methods for estimating the parameters of a linear regression model. It minimizes the sum of the squared differences between the observed (actual) values and the predicted values generated by the model.

The core idea behind OLS is to find the best-fit line (or hyperplane in the case of multiple variables) that minimizes the overall error across all data points. This method is commonly used in linear regression due to its simplicity and effectiveness.

The **OLS cost function** is given by:

$$J(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Where:

- **y_i** is the actual value for the i-th data point.
- **ŷ_i** is the predicted value for the i-th data point (using the regression model).
- The goal of OLS is to minimize this cost function.

By minimizing this function, OLS finds the **optimal parameters** (intercept and slope) that generate the best-fitting line for the given data.

Steps in OLS

1. **Minimize the Sum of Squared Residuals**: The objective of OLS is to minimize the sum of the squared residuals (errors). This is done by differentiating the cost function with respect to each parameter and setting the derivatives to zero. Solving these equations gives us the optimal parameters (intercept and slopes).

For Derivation visit: Understanding Ordinary Least Squares (OLS) Regression | Built In

Assumptions of OLS

For OLS to produce unbiased and efficient estimates, several key assumptions must be satisfied:

1. **Linearity**: The relationship between the independent variables and the dependent variable is linear.

2. **Independence**: Observations are independent of each other, meaning no two residuals are correlated.

3. **Homoscedasticity**: The variance of the residuals is constant across all levels of the independent variables. In other words, the spread of the residuals is consistent across the range of data points.

4. **Normality**: The residuals are normally distributed. This assumption is important for hypothesis testing and confidence intervals but less crucial for parameter estimation.

5. **No Multicollinearity**: The independent variables should not be too highly correlated with each other. High correlation between predictors (multicollinearity) can inflate the variance of the coefficient estimates, making them unstable and unreliable.

Interpreting OLS Results

Once the OLS method is used to estimate the parameters, it is essential to interpret the results:

1. **Intercept (θ_0)**: The intercept represents the predicted value of the dependent variable when all the independent variables are equal to zero. This value may or may not have practical significance, depending on the context of the data.

2. **Coefficients (θ_1, θ_2, …, θ_n)**: The coefficients represent the change in the dependent variable for a one-unit change in the corresponding independent variable, holding all other variables constant. The sign (+/-) of the coefficient indicates the direction of the relationship.

3. **R-squared (R^2)**: R^2 measures how well the independent variables explain the variation in the dependent variable. An R^2 value close to 1 indicates that the model explains most of the variation in the target variable, while a low R^2 value suggests that the model has limited explanatory power.

4. **p-values**: The p-values for the coefficients tell us whether each independent variable significantly affects the dependent variable. A p-value less than a chosen significance level (e.g., 0.05) indicates that the variable is statistically significant.

Advantages:

1. **Simplicity**: OLS is easy to understand and interpret, making it one of the most popular methods for regression analysis.

2. **Efficient Estimators**: Under the right conditions, OLS provides efficient and unbiased estimates of the regression coefficients.

3. **Closed-Form Solution**: OLS provides a closed-form solution for the parameter estimates, making it computationally efficient for small datasets.

Disadvantages:

1. **Sensitive to Outliers**: OLS is sensitive to outliers, as it squares the errors. Large errors (outliers) can disproportionately affect the overall model fit.

2. **Assumption of Linearity**: OLS assumes that the relationship between the dependent and independent variables is linear, which may not always be the case.

3. **Multicollinearity**: High correlation between independent variables can make the parameter estimates unreliable.

1. Overfitting

- Overfitting occurs when a model learns the training data **too well**, including noise and irrelevant patterns. As a result, the model performs very well on the training data but poorly on new, unseen data.
- In overfitting, the model becomes too complex, with too many parameters or degrees of freedom, which leads to poor generalization.

Signs of overfitting

  - Very low error on training data, but high error on validation/test data.
  - Model is excessively tuned to the specifics of the training data.

Causes of Overfitting

- Model complexity: Too many features or model parameters.
- Insufficient training data: The model is forced to capture even the noise in a small dataset.
- Lack of regularization: Regularization techniques like Ridge, Lasso, or Elastic Net are not used to penalize large coefficients.

Example: Imagine fitting a very high-degree polynomial to a small set of data points. The curve will pass through or very close to all points (training data) but will not generalize to new points, leading to high error on test data.

Solution to Overfitting

- Use a simpler model (reduce the number of features or parameters).
- Apply regularization (Ridge, Lasso, Elastic Net).
- Gather more training data.
- Use cross-validation to check model performance.

2. Underfitting

- Underfitting occurs when a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both the training data and new, unseen data.
- In underfitting, the model fails to learn the relationships between the input and output variables.

Signs of Underfitting

- High error on both training and test data.
- The model is too basic to capture the patterns in the data.

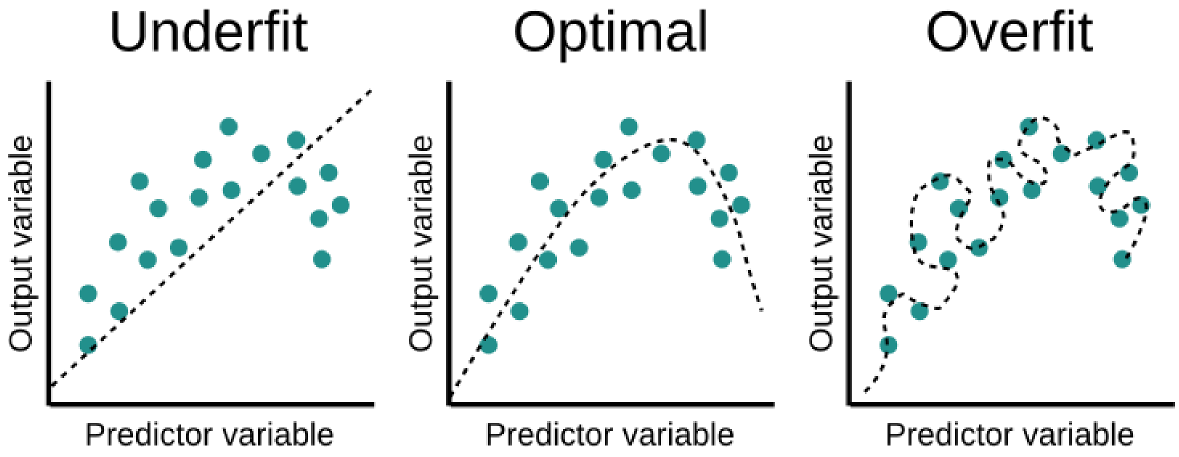
Causes of Underfitting

- Model is too simple: For example, using a linear model for non-linear data.
- Not enough features: The input data lacks sufficient information to make accurate predictions.
- Too much regularization: Excessive penalization of coefficients can lead to a very simple model.

Example: Fitting a straight line (linear regression) to a dataset with non-linear patterns would result in underfitting, as the line cannot adequately capture the data points.

Solution to Underfitting

- Use a more complex model (e.g., a higher-degree polynomial, decision tree).
- Add more relevant features to the dataset.
- Reduce regularization strength if it's too high.



Regularization

While regularization is not strictly an optimization technique, it is used in conjunction with optimization methods (like gradient descent) to improve the model’s performance and prevent **overfitting**. **Regularization** methods are used to prevent overfitting by adding a penalty to the model for having too many or overly large coefficients. Regularization methods address this issue by constraining or shrinking the model's coefficients, making the model simpler and more generalizable to new data.

The two most common regularization techniques for linear regression are **Ridge Regression** and **Lasso Regression**, and a hybrid of these two called **Elastic Net**.

1. Ridge Regression (L2 Regularization)

Overview:

- Ridge regression adds a penalty equal to the sum of the squared values of the coefficients to the linear regression cost function. This forces the algorithm to reduce the size of the coefficients, preventing them from becoming too large.
- Ridge regression shrinks the coefficients toward zero, but never exactly zero. Therefore, it maintains all features in the model but reduces their impact.

Ridge Regression Cost Function:

In Ridge regression, we add an L2 penalty term to linear regression cost function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

Where:

- λ is the regularization parameter that controls the strength of the penalty. A higher λ leads to more regularization (shrinkage).
- θj represents the model parameters (coefficients).
- $\sum_{j=1}^{n} \theta_j^2$ is the sum of the squares of all coefficients (except the intercept).

Interpretation:

- **When λ=0**: Ridge regression becomes ordinary least squares (OLS), as there is no regularization.
- **When λ increases**: The coefficients are shrunk, reducing the variance of the model. However, too high a λ may result in underfitting.

Use Case:

Ridge regression is useful when all the features are important but some need to be shrunk to avoid overfitting. It is particularly effective when there is multicollinearity (high correlation between features), as it can help reduce variance caused by correlated variables.

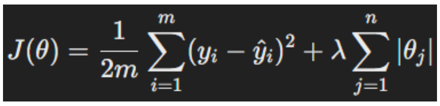
2. Lasso Regression (L1 Regularization)

Overview:

- Lasso (Least Absolute Shrinkage and Selection Operator) regression adds a penalty equal to the absolute value of the coefficients to the linear regression cost function. It can shrink some coefficients to exactly zero, thus performing **feature selection** by effectively removing unimportant features from the model.

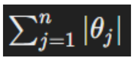
Lasso Regression Cost Function:

In Lasso regression, the cost function becomes:



$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{n}|\theta_j|$$

Where:

- λ is the regularization parameter controlling the strength of the penalty.
-  $\sum_{j=1}^{n}|\theta_j|$ is the sum of the absolute values of the coefficients (L1 norm).

Interpretation:

- **When λ=0**: Lasso regression becomes OLS with no regularization.
- **When λ increases**: Lasso starts shrinking coefficients, and some of them may become exactly zero, effectively removing those features from the model.

Use Case:

Lasso regression is used when you expect that only a small number of features are relevant to the model. It performs both regularization and feature selection, which is useful when you want a sparse model or when dealing with high-dimensional datasets where some features are irrelevant.

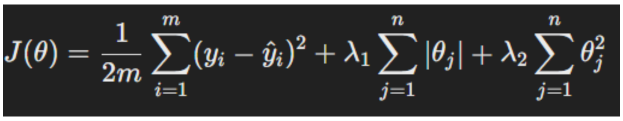
3. Elastic Net Regression (Combination of L1 and L2 Regularization)

Overview:

- Elastic Net is a hybrid of Ridge and Lasso regression that combines both L1 and L2 penalties. It helps address the limitations of Ridge (which doesn't perform feature selection) and Lasso (which may select too few features in the case of highly correlated predictors).

Elastic Net Cost Function:

The cost function for Elastic Net is:



$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \lambda_1\sum_{j=1}^{n}|\theta_j| + \lambda_2\sum_{j=1}^{n}\theta_j^2$$

Where:

- λ1 controls the strength of the L1 regularization (Lasso).
- λ2controls the strength of the L2 regularization (Ridge).

Interpretation:

- Elastic Net performs feature selection like Lasso but also includes Ridge's stability when predictors are correlated.
- It allows tuning of both penalties to balance the trade-off between feature selection and coefficient shrinkage.

Use Case:

Elastic Net is used when you want both regularization and feature selection, especially in cases where the predictors are highly correlated, which Lasso alone might struggle with. Elastic Net is more flexible than Ridge or Lasso alone since you can adjust both λ1 and λ2 to optimize model performance.

Types of Cost Functions in Linear Regression

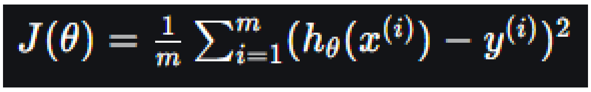
In linear regression, various cost functions can be used to measure the performance of the model. Some common cost functions are,

- Mean Absolute Error (MAE)
- Mean Square Error (MSE)
- Root Mean Square Error (RMSE)
- Huber Loss

Mean Square Error (MSE)

A common cost function in linear regression is Mean Squared Error (MSE), which penalizes large errors more than small ones.

Equation:



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

where:

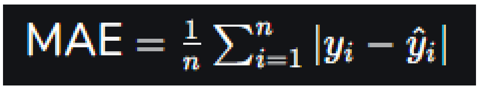
- J(θ) is the cost function,
- m is the number of data points,
- $h_\theta(x_i)$ is the predicted value from the linear regression model,
- y_i is the actual value for the i-th data point.

- **Advantages**:
  - MSE gives more weight to larger errors due to squaring, making it useful for models where large deviations are more important to correct.
  - Commonly used because it is mathematically convenient (its derivative is simple to compute) and works well with gradient-based optimization methods like gradient descent.
- **Disadvantages**:
  - Since it squares the errors, MSE is very sensitive to outliers. A few large errors can disproportionately affect the result.
- **When to Use**:
  - MSE is often preferred in general regression tasks due to its mathematical properties and the fact that it penalizes larger errors more heavily.

Mean Absolute Error (MAE)

Mean Absolute Error (MAE) calculates the average of the absolute differences between actual and predicted values offering a straightforward measure of prediction accuracy without considering whether the predictions were too high or too low.

Equation:



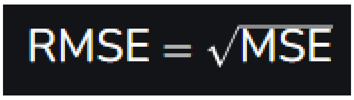
$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

- ❖ **Advantages**:
  - ○ MAE is simple and easy to interpret.
  - ○ Less sensitive to outliers compared to other cost functions like Mean Squared Error (MSE).
- ❖ **Disadvantages**:
  - ○ Since MAE doesn't penalize larger errors as strongly as squared-error functions, it may not be suitable when large errors are more concerning.
- ❖ **When to Use**:
  - ○ MAE is used when you want a cost function that measures the average magnitude of errors, and outliers aren't a significant concern.

Root Mean Square Error (RMSE)

Root Mean Squared Error (RMSE) is the square root of the MSE, providing a measure of error that is also in the same units as the target variable. RMSE provides a measure of error in the same units as the target variable. This makes it easier to interpret than MSE, as it tells you the average error in your predictions.

Equation:



$$\text{RMSE} = \sqrt{\text{MSE}}$$

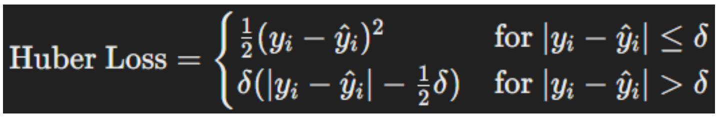
RMSE is beneficial when you need an interpretable measure of error that maintains sensitivity to larger mistakes, making it suitable for many regression tasks.

- ❖ **Advantages**:
  - ○ RMSE is more interpretable compared to MSE because the result is in the same unit as the original data.
  - ○ Like MSE, it gives more weight to larger errors.
- ❖ **Disadvantages**:
  - ○ Sensitive to outliers due to squaring of errors (just like MSE).
  - ○ The square root adds an additional computation, but it has little practical impact.
- ❖ **When to Use**:
  - ○ Use RMSE when you want the error metric to be in the same unit as the original data, and you want a more interpretable result than MSE.

Huber Loss

Huber Loss combines the advantages of both MSE and MAE. It is less sensitive to outliers than MSE by combining the squared error for small errors and absolute error for large errors. It introduces a threshold δ that determines when the function switches from squared error to absolute error.

Equation:



$$\text{Huber Loss} = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{for } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{for } |y_i - \hat{y}_i| > \delta \end{cases}$$

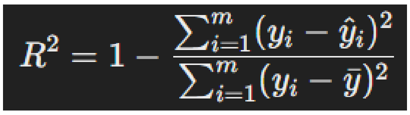
- **Advantages**:
  - It is less sensitive to outliers compared to MSE but penalizes them more than MAE.
  - Provides a smoother transition between MSE (for small errors) and MAE (for large errors).
- **Disadvantages**:
  - Requires choosing a threshold δ, which may require experimentation.
  - More complex to implement compared to MSE or MAE.
- **When to Use**:
  - Huber Loss is useful in situations where the data contains outliers, and you want to penalize small errors using squared errors while avoiding excessive penalties for large outliers.

Performance metrics

MAE, MSE, RMSE are act as Cost function as well as Performance metrics. Other than these, some performance metrics are

R-squared (R²) – Coefficient of Determination

- **Definition**: R-squared measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It essentially tells how well the model explains the variability of the target data.
- **Formula**:



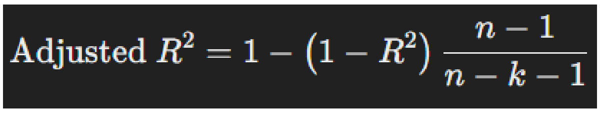
$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}$$

Where, y is the mean of the actual values.

- **Interpretation**:
  - R^2 ranges from 0 to 1, with 1 indicating that the model perfectly predicts the dependent variable.
  - An R^2 value of 0 means that the model does not explain any of the variance, while an R2 of 1 means the model explains all of the variance in the target variable.
- **When to Use**:
  - Use R^2 when you want to assess how well your model captures the variability in the data.
- **Limitations**:
  - R^2 can be misleading when comparing models with different numbers of features, as it does not account for the number of predictors in the model. A higher R^2 does not necessarily mean a better model.
  - In case of adding irrelevant features, the R^2 may increase even if the model performance doesn't improve.

Adjusted R-squared

- **Definition**: Adjusted R-squared modifies the R^2 metric to account for the number of predictors in the model. It penalizes models for having too many predictors, thereby preventing overfitting.
- **Formula**:



$$\text{Adjusted } R^2 = 1 - \left(1 - R^2\right)\frac{n-1}{n-k-1}$$

Where, n is the number of data points, k is the number of independent variables (predictors).

- **Interpretation**:
  - Adjusted R^2 increases only if the new predictor improves the model more than would be expected by chance. A higher adjusted R^2 indicates a better model, particularly when dealing with multiple predictors.
- **When to Use**:
  - Use Adjusted R^2 when comparing models with different numbers of predictors, as it adjusts for model complexity.