**Regularization in Machine Learning:**

**Regularization** is an important technique in machine learning that helps to improve model accuracy by preventing overfitting which happens when a model learns the training data too well including noise and outliers and perform poor on new data. By adding a penalty for complexity it helps simpler models to perform better on new data. In this article, we will see main types of regularization i.e Lasso, Ridge and Elastic Net and see how they help to build more reliable models.

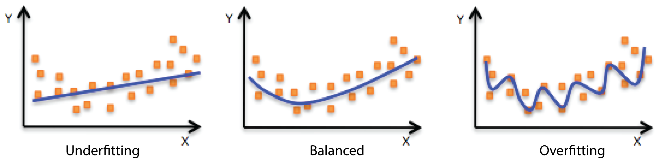
**Least Square Method finds the *Best* and *Unbiased* Coefficients**

You may know that least square method finds the coefficients that best fit the data. One more condition to be added is that it also finds the unbiased coefficients. Here unbiased means that **OLS doesn’t consider which independent variable is more important than others**. It simply finds the coefficients for a given data set. In short, there is only one set of betas to be found, resulting in the lowest ‘Residual Sum of Squares (RSS)’. The question then becomes *"Is a model with the lowest RSS truly the best model?"*.

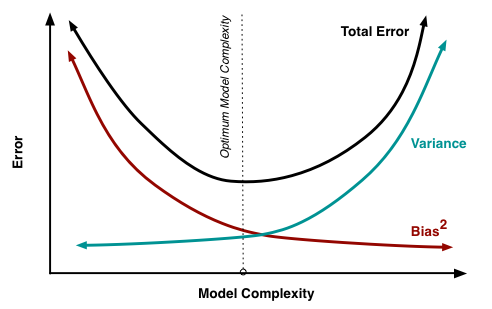
**Bias vs. Variance**

The answer for the question above is *"Not really"*. As hinted in the word ‘Unbiased’, we need to consider ‘Bias’ too. Bias means how equally a model cares about its predictors. Let’s say there are two models to predict an apple price with two predictor ‘sweetness’ and ‘shine’; one model is unbiased and the other is biased.

First, the unbiased model tries to find the relationship between the two features and the prices, just as the OLS method does. This model will fit the observations as perfectly as possible to minimize the RSS. However, this could easily lead to [overfitting](https://en.wikipedia.org/wiki/Overfitting) issues. In other words, the model will not perform as well with new data because it is built for the given data so specifically that it may not fit new data.



The biased model accepts its variables unequally to treat each predictor differently. Going back to the example, we would want to only care about ‘sweetness’ to build a model and this should perform better with new data. The reason will be explained after understanding **Bias vs. Variance**. If you’re not familiar with the bias vs. variance topic, I strongly recommend you to watch [this video](https://www.youtube.com/watch?v=EuBBz3bI-aA) that will give you insight.



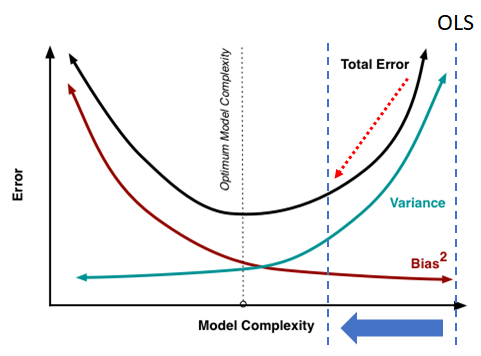
It can be said that **bias is related with a model failing to fit the training set and variance is related with a model failing to fit the testing set**. Bias and variance are in a trade-off relationship over model complexity, which means that a simple model would have high-bias and low-variance, and vice versa. In our apple example, a model only considering ‘sweetness’ would not fit the training data as much as the other model considering both ‘sweetness’ and ‘shine’, but the simpler model will be better at predicting new data.

This is because ‘sweetness’ is a determinant of a price while ‘shine’ should not by common sense. We all know this as a human but mathematical models do not think like us and just calculate what’s given until it finds some relationship between all the predictors and the independent variable to fit training data.

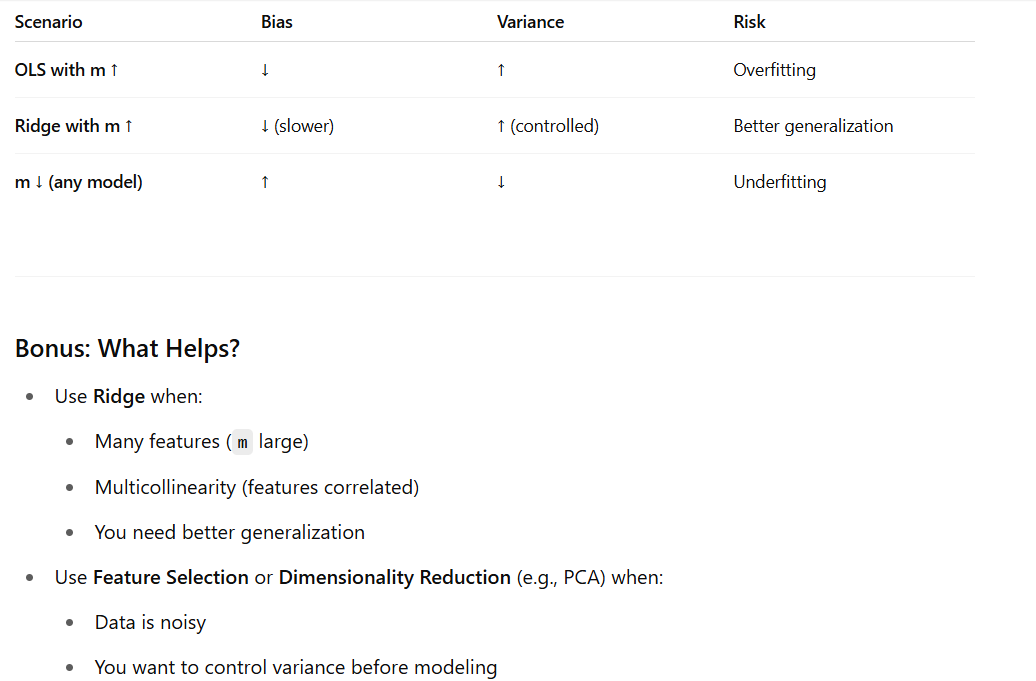
\**Note*: We assume that ‘sweetness’ and ‘shine’ are not correlated

**Where Ridge Regression Comes Into Play**

Looking at *Bias vs. Variance* figure, the Y-axis is ‘Error’ which is the ‘Sum of Bias and Variance’. Since both of them are basically related with failing, we would like to minimize those. Now taking a second look at the figure closely, you will find that the spot the total error is lowest is somewhere in the middle. This is often times called ‘Sweet Spot’.

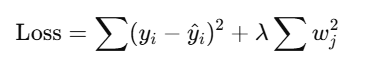


Let’s recall that OLS treats all the variables equally (unbiased). Therefore, an OLS model becomes more complex as new variables are added. It can be said that an OLS model is always on the rightest of the figure, having the lowest bias and the highest variance. It is fixed there, never moves, but we want to move it to the sweet spot. This is when ridge regression would shine, also referred to as *Regularization*. I**n ridge regression, you can tune the lambda parameter so that model coefficients change**. This can be best understood with a programming demo that will be introduced at the end.



**Q1. What is Ridge Regression?**

**A:** Ridge Regression is a regularized version of linear regression that adds an L2 penalty to the loss function. It discourages large weights by minimizing:



**Q2. What problem does Ridge Regression solve?**

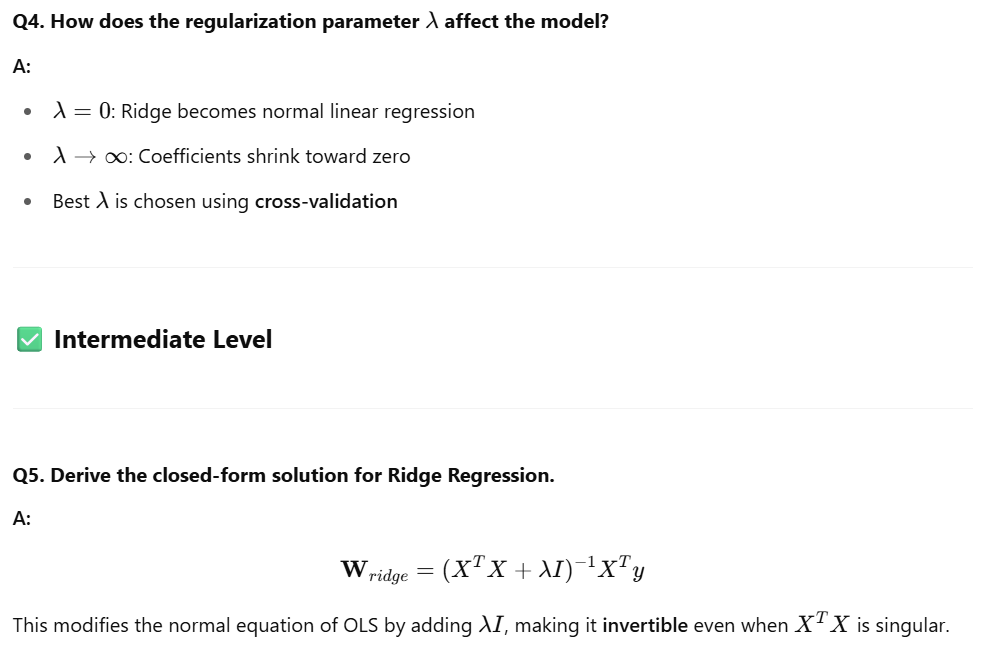
**A:** It solves **overfitting** in linear regression by:

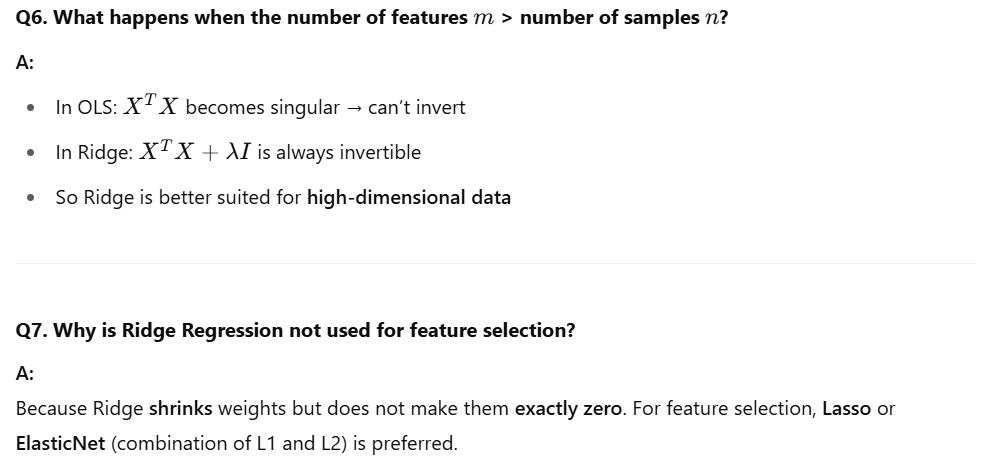
* Reducing **model variance**
* Handling **multicollinearity** (highly correlated features)
* Making the solution more **robust** and **generalizable**

**Q3. What's the difference between Ridge and Lasso?**

**A:**

| **Ridge** | **Lasso** |
| --- | --- |
| L2 regularization | L1 regularization |
| Penalizes square of weights | Penalizes absolute weights |
| Shrinks weights, but keeps all features | Can set some weights to **zero** (feature selection) |





**Q8. How does Ridge Regression handle multicollinearity?**

**A:**

* It **stabilizes the inversion** of X^T \* X
* Spreads the weight across correlated features
* Prevents one feature from dominating due to redundancy

**Q9. Explain the bias-variance trade-off in Ridge Regression.**

**A:**

* Ridge **increases bias** slightly (adds constraint)
* But **reduces variance** significantly (prevents overfitting)
* Overall error may **decrease**, improving generalization

**Q10. How do you select the best value of λ?**

**A:**

* Use **K-fold cross-validation**
* Test different λ values (e.g., using a log scale)
* Choose the one minimizing **validation error**

**✅ Bonus Coding Question**

**Q11. Implement Ridge Regression from scratch using NumPy.**

**Types of Regularization:**

* 1. **Ridge Regression**
  2. **Lasso Regression**
  3. **Elastic Net Regression**
  4. **Ridge regression:**

[**https://www.geeksforgeeks.org/what-is-ridge-regression/**](https://www.geeksforgeeks.org/what-is-ridge-regression/)

[**https://www.geeksforgeeks.org/implementation-of-ridge-regression-from-scratch-using-python/**](https://www.geeksforgeeks.org/implementation-of-ridge-regression-from-scratch-using-python/)

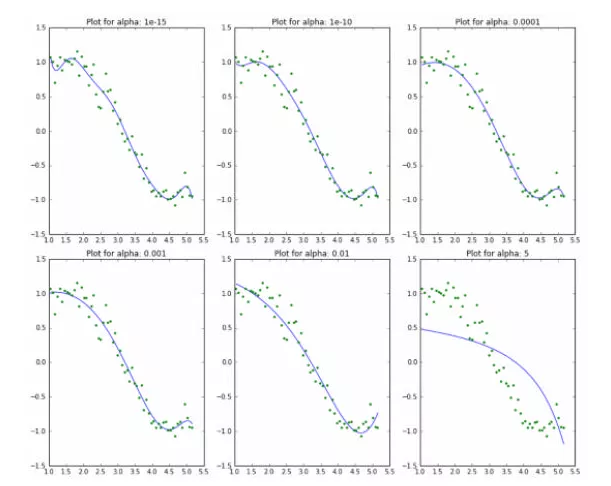
[**https://www.geeksforgeeks.org/ml-ridge-regressor-using-sklearn/**](https://www.geeksforgeeks.org/ml-ridge-regressor-using-sklearn/)

[**https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db/**](https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db/)

**1. Ridge Regression:**

A regression model that uses the **L2 regularization** technique is called [**Ridge regression**](https://www.geeksforgeeks.org/what-is-ridge-regression/). It adds the **squared magnitude** of the coefficient as a penalty term to the loss function(L).

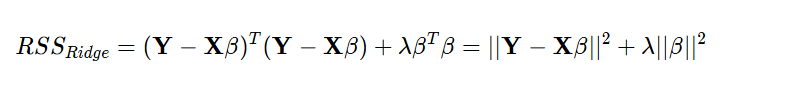
**Ridge Regression** (also called **L2 regularization**) is a variant of **Linear Regression** that **adds a penalty on the size of coefficients** to **reduce overfitting** and **improve generalization**.



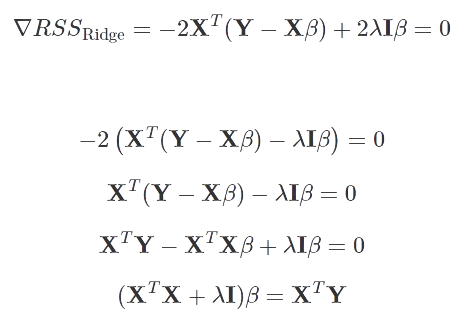
Here we can clearly observe that **as the value of alpha increases, the model complexity reduces**. Though higher values of alpha reduce overfitting, significantly high values can cause underfitting as well (e.g., alpha = 5). Thus alpha should be chosen wisely. A widely accepted technique is cross-validation, i.e., the value of alpha is iterated over a range of values, and the one giving a higher cross-validation score is chosen.

**How to estimate coefficients in ridge regression?**

Just as in the case of regression, where we minimized the RSS, for ridge regression, we minimize the expression we mentioned earlier, but this time let’s express it in matrix form:

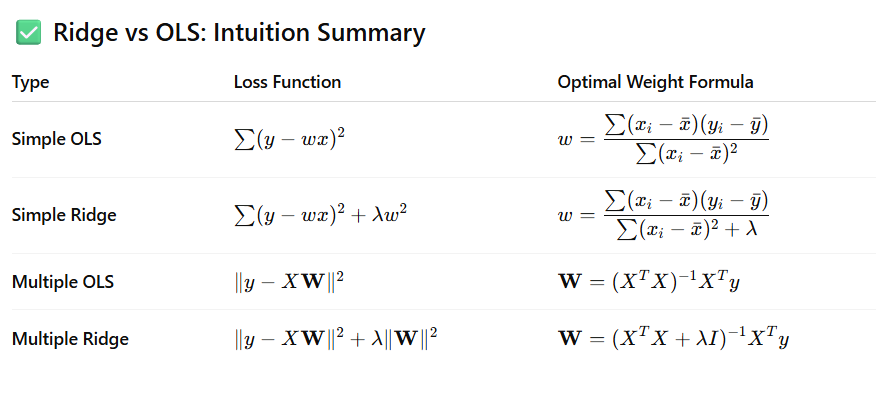


To minimize the *𝑅𝑆𝑆\_Ridge* expression, we will set its derivative (with respect to 𝛽) equal to zero.



The matrix 𝑋^𝑇𝑋+𝜆𝐼 has full rank and it is invertible. As a consequence:

Bias–variance tradeoff of the ridge estimator



When we talk about regression, we often end up discussing [**Linear and Logistic Regression**](https://www.analyticsvidhya.com/blog/2020/12/beginners-take-how-logistic-regression-is-related-to-linear-regression/), as they are the most popular of the 7 types of regressions. In this article, we’ll focus on Ridge and Lasso regression, which are powerful techniques generally used for creating parsimonious models in the presence of a ‘large’ number of features. Here ‘large’ can typically mean either of two things:

* Large enough to enhance the tendency of a model to overfit (as low as 10 variables might cause overfitting)
* Large enough to cause computational challenges. With modern systems, this situation might arise in the case of millions or billions of features.

Though Ridge and Lasso might appear to work towards a common goal, the inherent properties and practical use cases differ substantially. If you’ve heard of them before, you must know that they work by penalizing the magnitude of coefficients of features and minimizing the error between predicted and actual observations. These are called ‘regularization’ techniques.

Here’s a table outlining the key differences between ridge vs lasso regression:

| **Feature** | **Ridge Regression** | **Lasso Regression** |
| --- | --- | --- |
| Description | Ridge regression, also known as Tikhonov regularization, is a technique that introduces a penalty term to the linear regression model to shrink the coefficient values. | Lasso regression, or Least Absolute Shrinkage and Selection Operator, is a regularization method that also includes a penalty term but can set some coefficients exactly to zero, effectively selecting relevant features. |
| Penalty Type | Ridge regression utilizes an L2 penalty, which adds the sum of the squared coefficient values multiplied by a tuning parameter (lambda). | Lasso regression employs an L1 penalty, which sums the absolute values of the coefficients multiplied by lambda. |
| Coefficient Impact | The L2 penalty in ridge regression discourages large coefficient values, pushing them towards zero but never exactly reaching zero. This shrinks the less important features’ impact. | The L1 penalty in lasso regression can drive some coefficients to exactly zero when the lambda value is large enough, performing feature selection and resulting in a sparse model. |
| Feature Selection | Ridge regression retains all features in the model, reducing the impact of less important features by shrinking their coefficients. | Lasso regression can set some coefficients to zero, effectively selecting the most relevant features and improving model interpretability. |
| Use Case | Ridge regression is useful when the goal is to minimize the impact of less important features while keeping all variables in the model. | Lasso regression is preferred when the goal is feature selection, resulting in a simpler and more interpretable model with fewer variables. |
| Model Complexity | Ridge regression tends to favor a model with a higher number of parameters, as it shrinks less important coefficients but keeps them in the model. | Lasso regression can lead to a less complex model by setting some coefficients to zero, reducing the number of effective parameters. |
| Interpretability | The results of ridge regression may be less interpretable due to the inclusion of all features, each with a reduced but non-zero coefficient. | Lasso regression can improve interpretability by selecting only the most relevant features, making the model’s predictions more explainable. |
| Sparsity | Ridge regression does not yield sparse models since all coefficients remain non-zero. | Lasso regression can produce sparse models by setting some coefficients to exactly zero. |
| Sensitivity | More robust and less sensitive to outliers compared to lasso regression. | More sensitive to outliers due to the absolute value in the penalty term. |

Regularization Techniques

**Full Ridge Regression Example in Python (with Analysis)**

**🎯 Goal: Compare Linear Regression and Ridge Regression on polynomial data**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, Ridge

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Generate synthetic data (non-linear)

np.random.seed(42)

X = 2 \* np.random.rand(100, 1) - 1 # X in [-1, 1]

y = 3 \* X\*\*2 + 2 \* X + 1 + np.random.randn(100, 1) \* 0.3 # Quadratic + noise

print("Sample data (first 5 rows):")

print(np.hstack((X[:5], y[:5])))

# Step 2: Polynomial features (degree=2)

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_poly = poly.fit\_transform(X) # Creates [x, x^2]

# Step 3: Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_poly, y, test\_size=0.3, random\_state=42)

# Step 4: Linear Regression (baseline)

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train, y\_train)

y\_pred\_lin = lin\_reg.predict(X\_test)

# Step 5: Ridge Regression

ridge\_reg = Ridge(alpha=1.0) # α = regularization strength

ridge\_reg.fit(X\_train, y\_train)

y\_pred\_ridge = ridge\_reg.predict(X\_test)

# Step 6: Evaluation

print("\n🔍 Linear Regression:")

print("Coefficients:", lin\_reg.coef\_)

print("Intercept:", lin\_reg.intercept\_)

print("MSE:", mean\_squared\_error(y\_test, y\_pred\_lin))

print("R² Score:", r2\_score(y\_test, y\_pred\_lin))

print("\n🔍 Ridge Regression:")

print("Coefficients:", ridge\_reg.coef\_)

print("Intercept:", ridge\_reg.intercept\_)

print("MSE:", mean\_squared\_error(y\_test, y\_pred\_ridge))

print("R² Score:", r2\_score(y\_test, y\_pred\_ridge))

# Step 7: Visualization

plt.figure(figsize=(10, 6))

plt.scatter(X, y, color='gray', alpha=0.5, label='Data')

X\_plot = np.linspace(-1, 1, 100).reshape(-1, 1)

X\_plot\_poly = poly.transform(X\_plot)

plt.plot(X\_plot, lin\_reg.predict(X\_plot\_poly), color='blue', label='Linear Regression')

plt.plot(X\_plot, ridge\_reg.predict(X\_plot\_poly), color='green', linestyle='--', label='Ridge Regression (α=1.0)')

plt.title("Linear vs Ridge Regression")

plt.xlabel("X")

plt.ylabel("y")

plt.grid(True)

plt.legend()

plt.show()

* 1. **Lasso regression:**

[**https://www.geeksforgeeks.org/what-is-lasso-regression/**](https://www.geeksforgeeks.org/what-is-lasso-regression/)

[**https://www.geeksforgeeks.org/implementation-of-lasso-regression-from-scratch-using-python/**](https://www.geeksforgeeks.org/implementation-of-lasso-regression-from-scratch-using-python/)

* 1. **Ridge Regression vs Lasso Regression:**

[**https://www.geeksforgeeks.org/ridge-regression-vs-lasso-regression/**](https://www.geeksforgeeks.org/ridge-regression-vs-lasso-regression/)

**1. Lasso Regression**

**Introduction to LASSO Regression**

LASSO regression, also known as L1 regularization, is a popular technique used in statistical modeling and machine learning to estimate the relationships between variables and make predictions. LASSO stands for Least Absolute Shrinkage and Selection Operator.

The primary goal of LASSO regression is to find a balance between model simplicity and accuracy. It achieves this by adding a penalty term to the traditional[linear regression](https://www.mygreatlearning.com/blog/linear-regression-in-machine-learning/)model, which encourages sparse solutions 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.

The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). 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.

Lasso Regression uses L1 regularization technique (will be discussed later in this article). It is used when we have more features because it automatically performs feature selection.

A regression model which uses the L1 Regularization technique is called [LASSO (Least Absolute Shrinkage and Selection Operator)](https://www.geeksforgeeks.org/what-is-lasso-regression/) regression. It adds the absolute value of magnitude of the coefficient as a penalty term to the loss function(L). This penalty can shrink some coefficients to zero which helps in selecting only the important features and ignoring the less important ones.

**Difference Between Ridge Regression and Lasso Regression**

| **Ridge Regression** | **Lasso Regression** |
| --- | --- |
| The penalty term is the sum of the squares of the coefficients (L2 regularization). | The penalty term is the sum of the absolute values of the coefficients (L1 regularization). |
| Shrinks the coefficients but doesn’t set any coefficient to zero. | Can shrink some coefficients to zero, effectively performing feature selection. |
| Helps to reduce overfitting by shrinking large coefficients. | Helps to reduce overfitting by shrinking and selecting features with less importance. |
| Works well when there are a large number of features. | Works well when there are a small number of features. |
| Performs “soft thresholding” of coefficients. | Performs “hard thresholding” of coefficients. |

In short, Ridge is a shrinkage model, and Lasso is a feature selection model. Ridge tries to balance the bias-variance trade-off by shrinking the coefficients, but it does not select any feature and keeps all of them. Lasso tries to balance the bias-variance trade-off by shrinking some coefficients to zero.  
  
In this way, Lasso can be seen as an optimizer for feature selection.

**Regularization**

When it comes to training models, there are two major problems one can encounter: [**overfitting**](https://www.datacamp.com/tutorial/towards-preventing-overfitting-regularization) and underfitting.

* Overfitting happens when the model performs well on the training set but not so well on unseen (test) data.
* Underfitting happens when it neither performs well on the train set nor on the test set.

Particularly, regularization is implemented to avoid overfitting of the data, especially when there is a large variance between train and test set performances. With regularization, the number of features used in training is kept constant, yet the magnitude of the coefficients (w) as seen in equation 1.1, is reduced.

Consider the image of coefficients below to predict house prices. While there are quite a number of predictors, RM and RAD have the largest coefficients. The implication of this will be that housing prices will be driven more significantly by these two features leading to overfitting, where generalizable patterns have not been learned.

There are different ways of reducing model complexity and preventing overfitting in linear models. This includes ridge and lasso regression models.

**Introduction to Lasso Regression**

This is a regularization technique used in feature selection using a Shrinkage method also referred to as the **penalized regression method**. Lasso is short for **L**east **A**bsolute **S**hrinkage and **S**election **O**perator, which is used both for regularization and model selection. If a model uses the **L1 regularization** technique, then it is called lasso regression.

**Lasso Regression for Regularization**

In this shrinkage technique, the coefficients determined in the linear model from equation 1.1. above are shrunk towards the central point as the mean by introducing a penalization factor called the alpha α (or sometimes lamda) values.

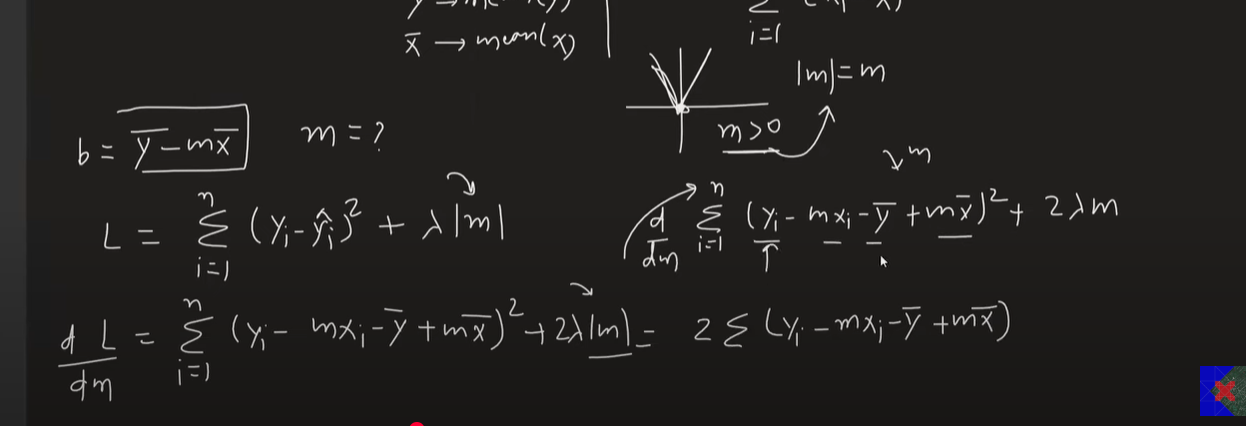


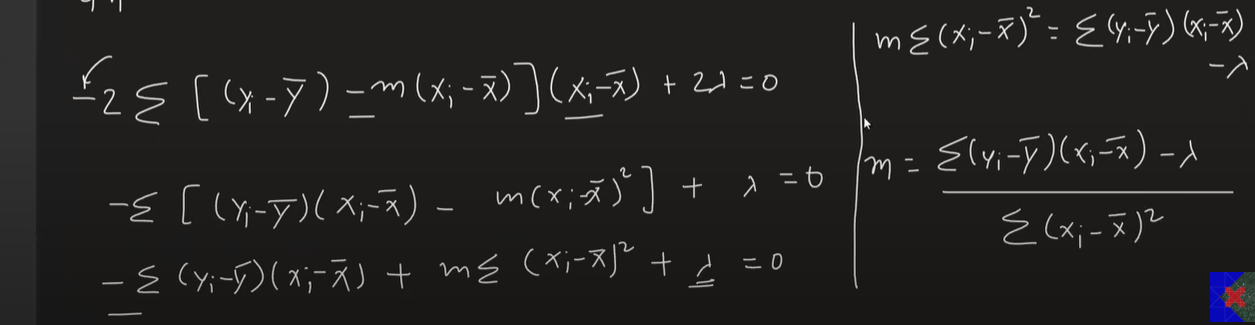
Alpha (α) is the penalty term that denotes the amount of shrinkage (or constraint) that will be implemented in the equation. With alpha set to zero, you will find that this is the equivalent of the linear regression model from equation 1.2, and a larger value penalizes the optimization function. Therefore, lasso regression shrinks the coefficients and helps to reduce the model complexity and multi-collinearity.

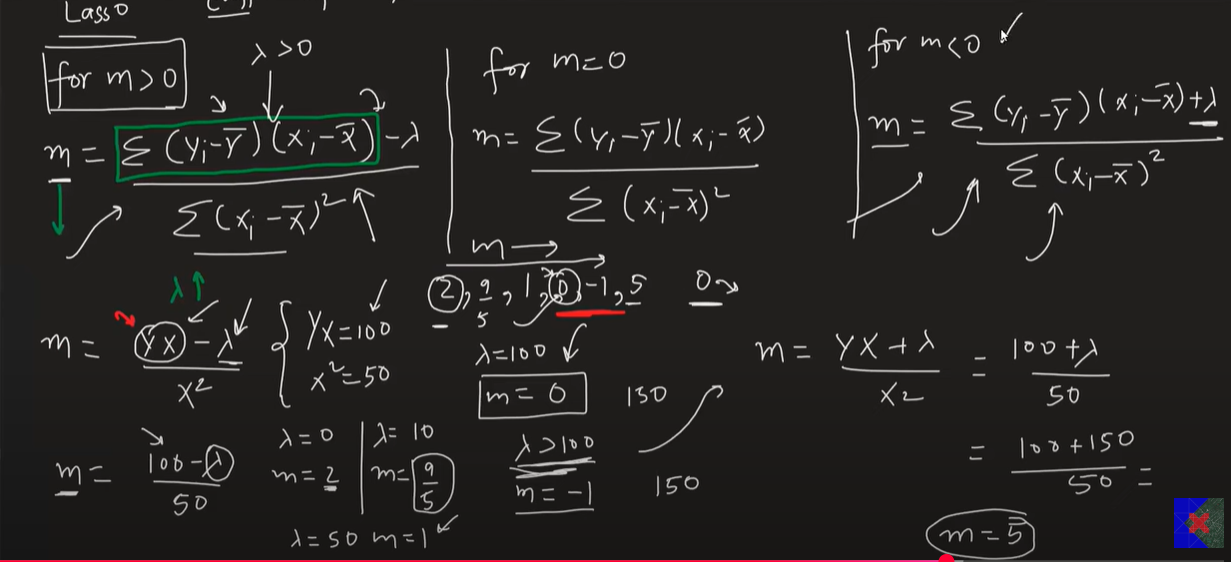
Alpha (α) can be any real-valued number between zero and infinity; the larger the value, the more aggressive the penalization is.

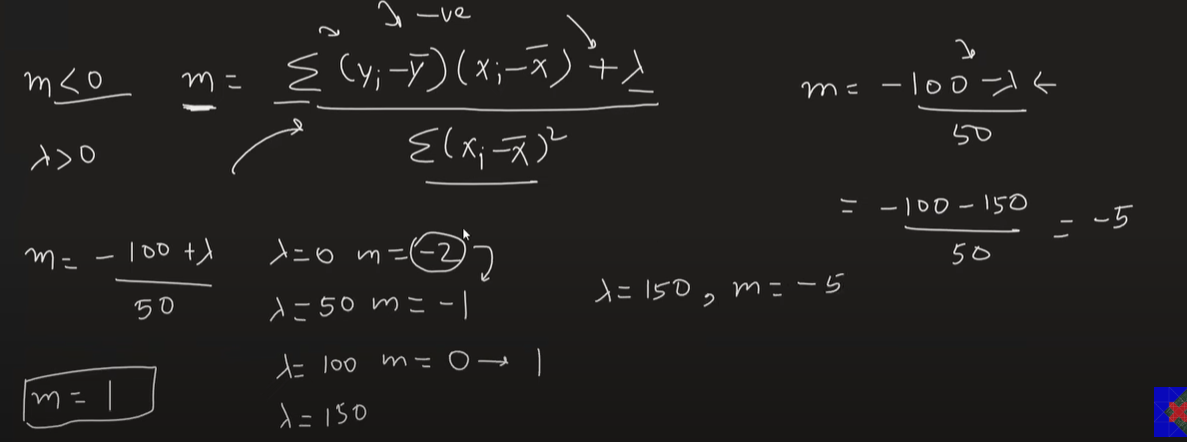
**Lasso Regression for Model Selection**

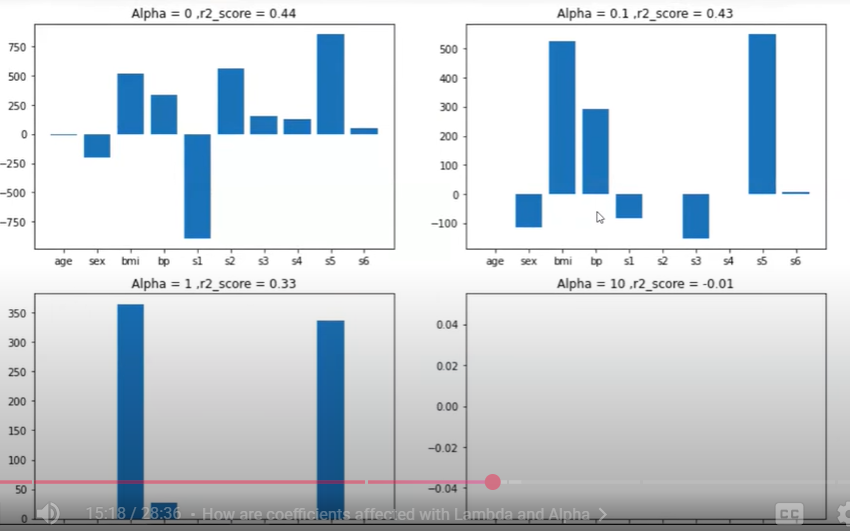
Due to the fact that coefficients will be shrunk towards a mean of zero, less important features in a dataset are eliminated when penalized. The shrinkage of these coefficients based on the alpha value provided leads to some form of automatic feature selection, as input variables are removed in an effective approach.



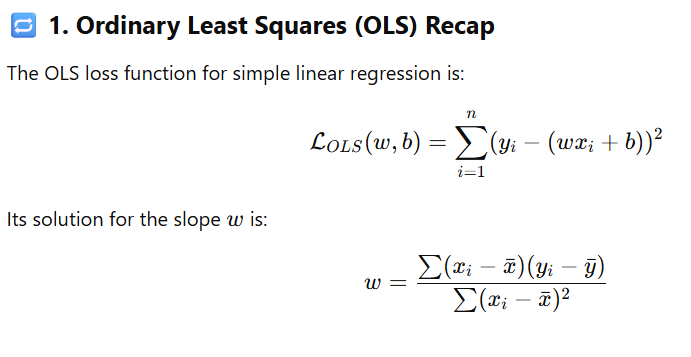
****

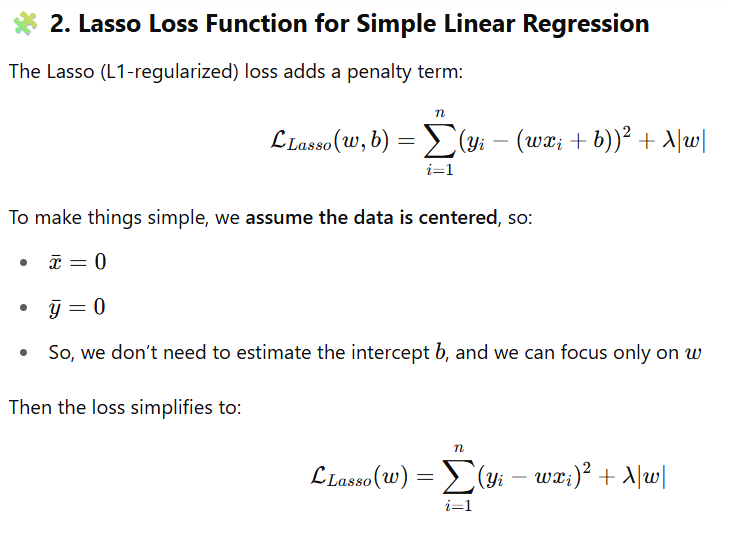


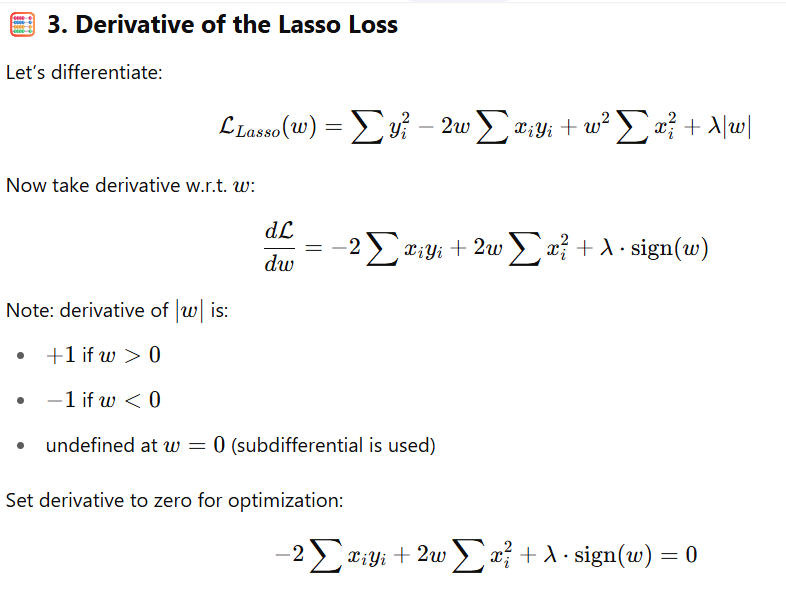


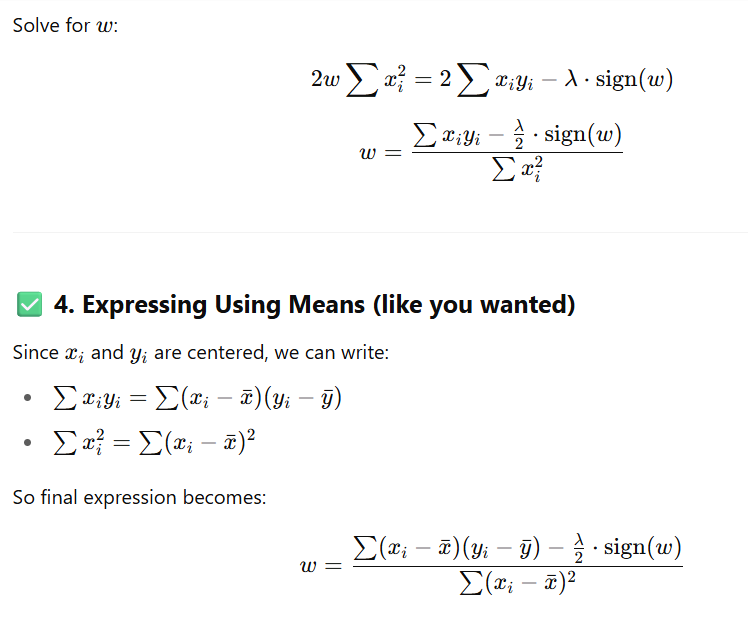


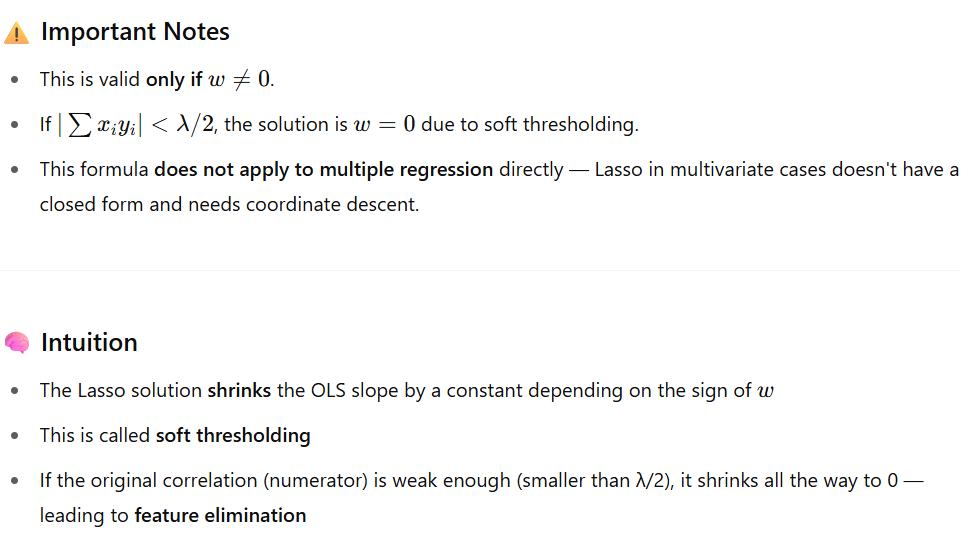
**Full Derivation formula:**



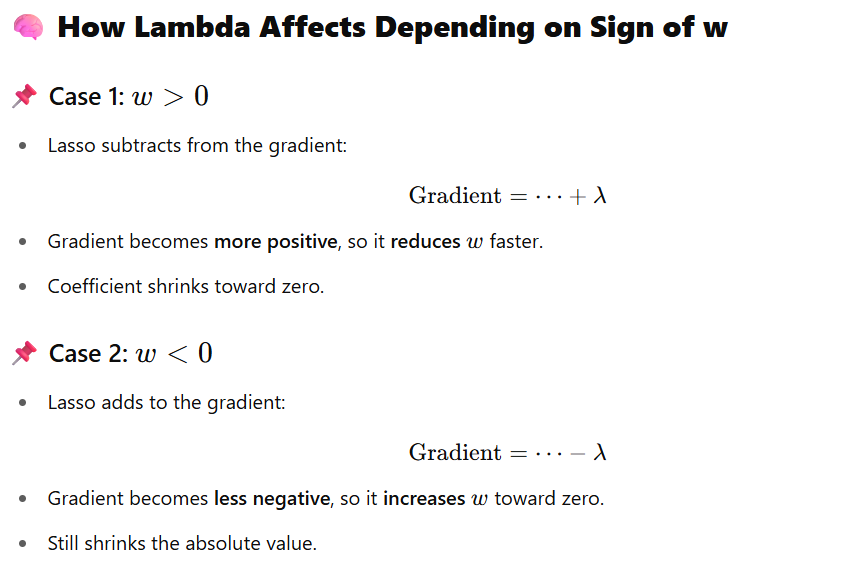










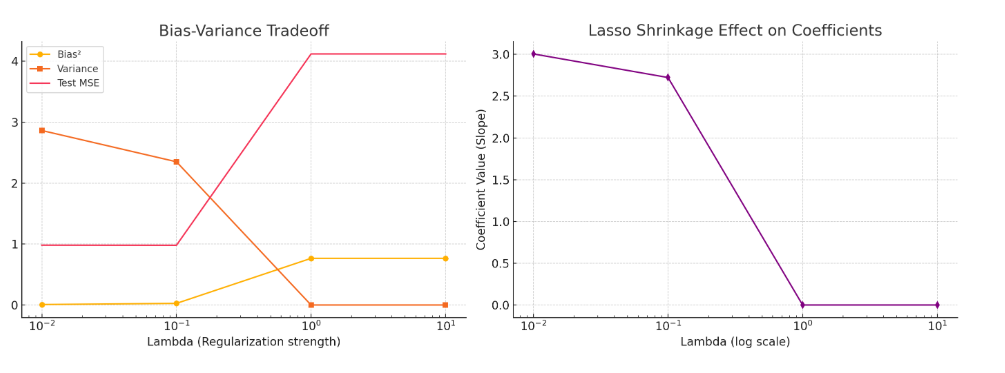


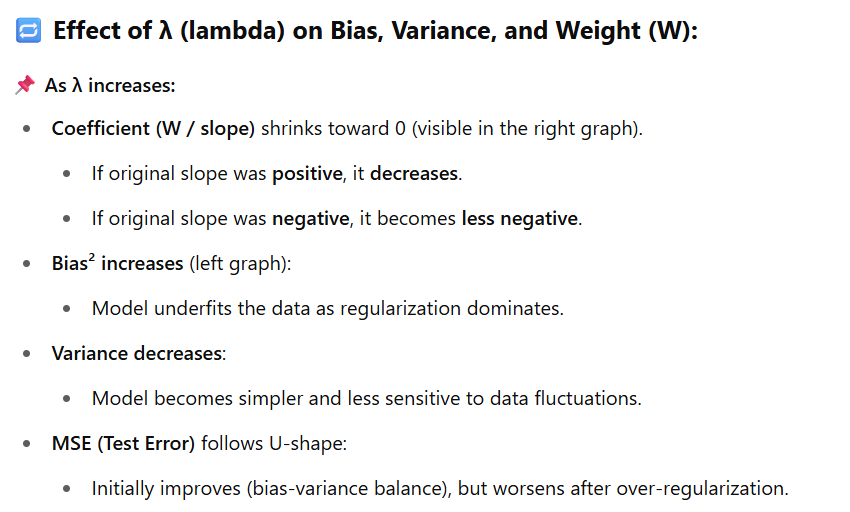
**Geometrical Intuition (L1 vs L2)**

* L1 (Lasso): Diamond-shaped constraint → encourages sparse solutions (some exact zero).
* L2 (Ridge): Circular constraint → shrinks all weights but rarely to exact zero.

**✅ Final Intuition Summary**

* **λ increases ⇒ more regularization ⇒ more bias, less variance**
* **Lasso favors sparse models** – especially useful when many features are irrelevant.
* **For positive or negative w**, lambda always **shrinks toward zero**.





**4. Why does Lasso shrink some weights to zero?**

**Answer:**  
Because the **L1 penalty** forms a **diamond-shaped constraint**, when combined with the OLS loss (a paraboloid), the minimum often occurs at a vertex where some coefficients are exactly zero.

This geometric property leads to **sparsity** in the solution.

**5. What happens as λ increases in Lasso Regression?**

**Answer:**

* **Small λ** → behaves like normal linear regression.
* **Moderate λ** → reduces some weights, selects important features.
* **Large λ** → shrinks most or all weights to zero (underfitting).
* **Bias increases**, **variance decreases**.

**6. Can Lasso handle multicollinearity?**

**Answer:**  
Yes, Lasso can handle multicollinearity to some extent by **selecting one variable from a group of correlated features** and shrinking the others to zero. However, it may be unstable in how it chooses among them.

**7. What are the limitations of Lasso Regression?**

**Answer:**

* **Instability with correlated features**.
* **Only one variable is chosen from a group of correlated predictors**, which may not be ideal in all scenarios.
* **Not suitable when number of predictors > number of observations (p > n)** unless λ is well-tuned.

**8. How is λ selected in Lasso Regression?**

**Answer:**

* Typically chosen using **cross-validation** (e.g., k-fold CV).
* Tools like LassoCV in scikit-learn can automatically search for the best λ.

**9. How do you interpret the output of Lasso Regression?**

**Answer:**

* **Non-zero coefficients** → important predictors.
* **Zero coefficients** → irrelevant predictors (eliminated).
* The magnitude of non-zero coefficients indicates strength of influence.

**10. Which optimization method is used to solve Lasso Regression?**

**Answer:**

* **Coordinate Descent** is commonly used.
* Other methods: **LARS**, **Subgradient methods**, and **Proximal Gradient Descent**.

**11. Can Lasso be used for classification tasks?**

**Answer:**  
Yes. The concept of L1 regularization is used in **Lasso Logistic Regression** for binary classification. It still enforces sparsity by shrinking unimportant feature weights to zero.

**12. When should you prefer Lasso over Ridge?**

**Answer:**

* When you **expect many features to be irrelevant**.
* When you **want feature selection and model interpretability**.
* In **high-dimensional** problems (e.g., text data, genomics).

**13. How does Lasso relate to feature selection methods like Recursive Feature Elimination (RFE)?**

**Answer:**

* Lasso performs **automatic feature selection** as part of model training.
* RFE is a **wrapper method** that requires retraining multiple models.
* Lasso is typically faster and better suited for high-dimensional data.

**14. What does the regularization path mean in Lasso?**

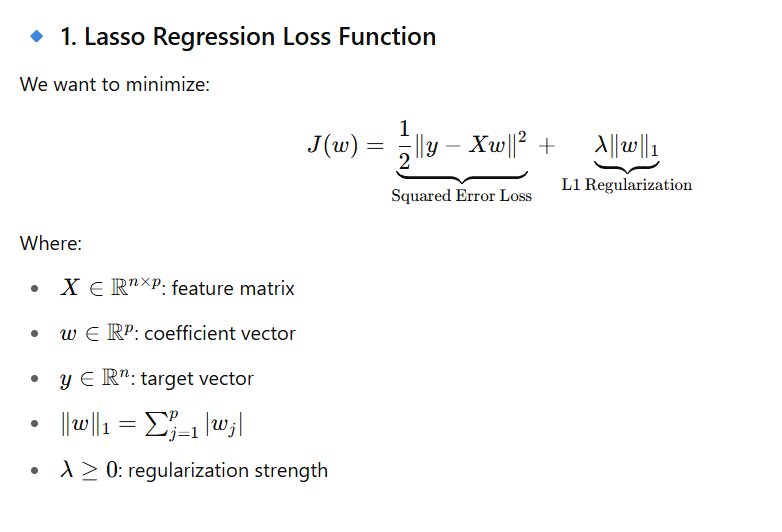
**Answer:**  
It shows how the coefficients of each feature change as λ varies. As λ increases, more coefficients shrink to zero. This is useful for **model selection and feature analysis**.

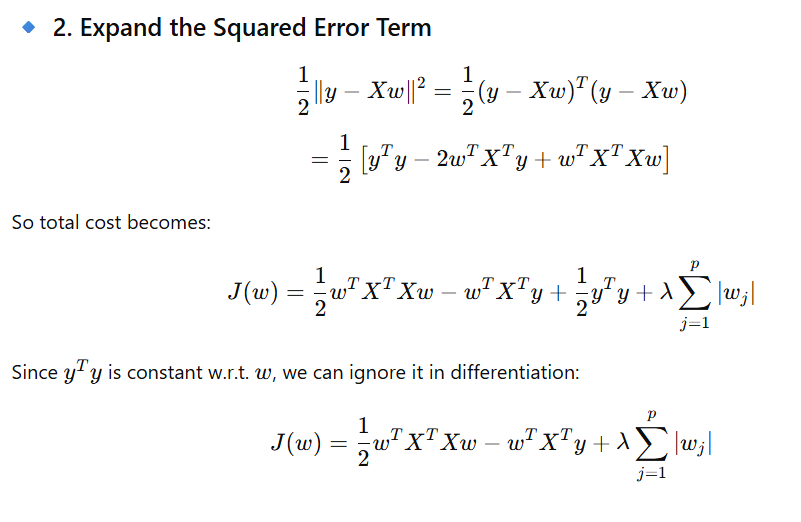
**15. What are some alternatives to Lasso?**

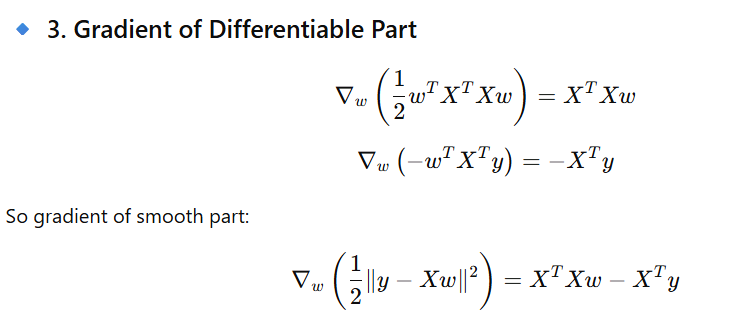
**Answer:**

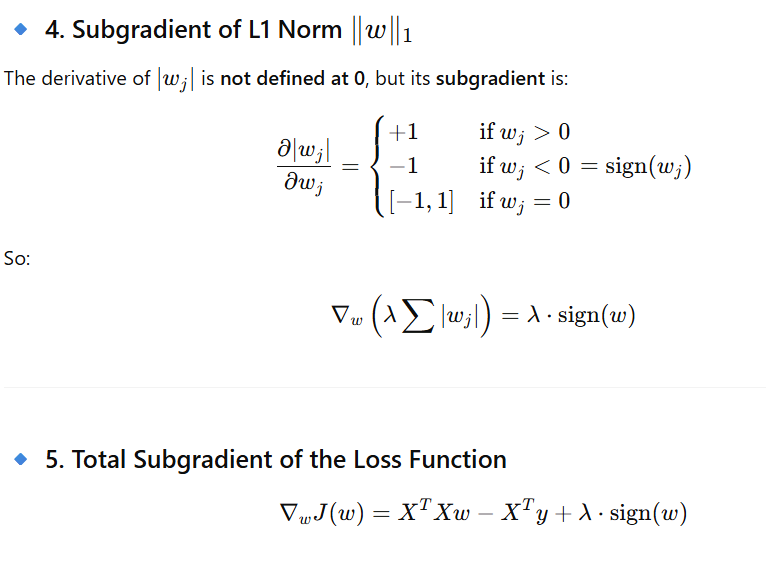
* **Ridge Regression** – no feature selection, good for multicollinearity.
* **Elastic Net** – hybrid of Lasso and Ridge, handles correlated features better.
* **Stepwise Selection** – traditional but computationally expensive.

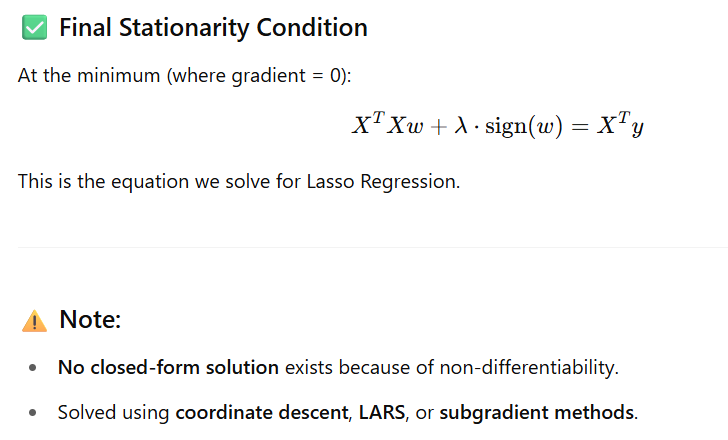
**Lasso Multiple regression :-**











**Full Code with Explanations**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, Lasso

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import PolynomialFeatures

# Step 1: Generate synthetic data (non-linear)

np.random.seed(42) # Ensures reproducibility

X = 2 \* np.random.rand(100, 1) - 1 # 100 points between [-1, 1]

y = 3 \* X\*\*2 + 2 \* X + 1 + np.random.randn(100, 1) \* 0.3 # y = 3x² + 2x + 1 + noise

# Print data before split

print("Sample Data (X, y):")

print(np.hstack((X[:5], y[:5])))

# Step 2: Polynomial features transformation (degree=2)

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_poly = poly.fit\_transform(X)

# Parameters:

# degree=2 → Create x and x² features

# include\_bias=False → Don't add column of 1s (for intercept)

# Step 3: Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_poly, y, test\_size=0.3, random\_state=42)

# Parameters:

# test\_size=0.3 → 30% data for testing

# random\_state=42 → Ensures consistent split across runs

# Step 4: Linear Regression model (baseline)

lin\_model = LinearRegression()

lin\_model.fit(X\_train, y\_train)

# Step 5: Lasso Regression model

lasso\_model = Lasso(alpha=0.1, max\_iter=10000)

lasso\_model.fit(X\_train, y\_train)

# Parameters:

# alpha=0.1 → Regularization strength (higher = more shrinkage)

# max\_iter=10000 → To ensure convergence

# Step 6: Predictions

y\_pred\_lin = lin\_model.predict(X\_test)

y\_pred\_lasso = lasso\_model.predict(X\_test)

# Step 7: Evaluation

mse\_lin = mean\_squared\_error(y\_test, y\_pred\_lin)

r2\_lin = r2\_score(y\_test, y\_pred\_lin)

mse\_lasso = mean\_squared\_error(y\_test, y\_pred\_lasso)

r2\_lasso = r2\_score(y\_test, y\_pred\_lasso)

# Print results

print("\n🔍 Linear Regression:")

print("Coefficients:", lin\_model.coef\_)

print("Intercept:", lin\_model.intercept\_)

print("MSE:", mse\_lin)

print("R²:", r2\_lin)

print("\n🔍 Lasso Regression:")

print("Coefficients:", lasso\_model.coef\_)

print("Intercept:", lasso\_model.intercept\_)

print("MSE:", mse\_lasso)

print("R²:", r2\_lasso)

# Step 8: Plotting

plt.figure(figsize=(10, 6))

plt.scatter(X, y, color='gray', alpha=0.5, label='Data')

# Sort X for plotting smooth curve

X\_plot = np.linspace(-1, 1, 100).reshape(-1, 1)

X\_plot\_poly = poly.transform(X\_plot)

plt.plot(X\_plot, lin\_model.predict(X\_plot\_poly), color='blue', label='Linear Regression')

plt.plot(X\_plot, lasso\_model.predict(X\_plot\_poly), color='red', linestyle='--', label='Lasso Regression (α=0.1)')

plt.title("Linear vs Lasso Regression")

plt.xlabel("X")

plt.ylabel("y")

plt.legend()

plt.grid(True)

plt.show()

**Result Analysis**

**Linear Regression:**

* Fits the data closely (no penalty).
* May **overfit** slightly when the noise is high or the model is too complex.

**Lasso Regression:**

* Applies L1 penalty to shrink coefficients.
* If alpha is high, it can **zero out coefficients** → Feature selection.
* Helps in **preventing overfitting**.

**Elastic Net Regression**

The main purpose of ElasticNet Regression is to find the coefficients that minimize the sum of error squares by applying a penalty to these coefficients. ElasticNet combines L1 and L2 (Lasso and Ridge) approaches. As a result, it performs a more efficient smoothing process. In another source, it is defined as follows:

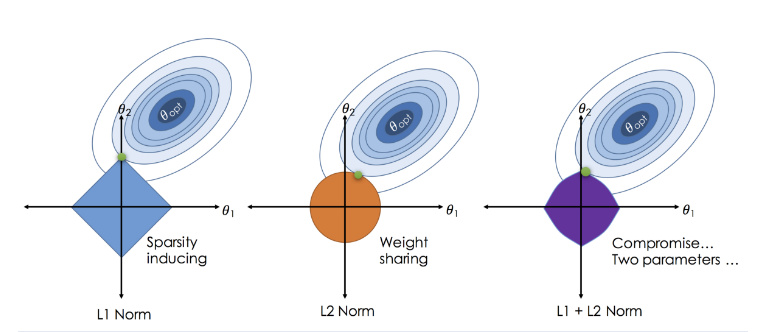
*Elastic Net first emerged as a result of critique on Lasso, whose variable selection can be too dependent on data and thus unstable. The solution is to combine the penalties of Ridge regression and Lasso to get the best of both worlds.*

**Features of ElasticNet Regression**

* It combines the L1 and L2 approaches.
* It performs a more efficient regularization process.
* It has two parameters to be set, *λ* and*α.*

*The elastic net method improves on lasso’s limitations, i.e., where lasso takes a few samples for high dimensional data, the elastic net procedure provides the inclusion of “n” number of variables until saturation. In a case where the variables are highly correlated groups, lasso tends to choose one variable from such groups and ignore the rest entirely.*

Zoom image will be displayed



Differences between L1, L2, and L1+L2 Norm

**Elastic Net Regression**

Elastic Net Regression is a mix of Ridge and Lasso Regression that combines their penalty terms. The name "Elastic Net" comes from physics: just like an elastic net can stretch and still keep its shape, this method adapts to data while maintaining structure.

The model balances three goals: minimizing prediction errors, keeping the size of coefficients small (like Lasso), and preventing any coefficient from becoming too large (like Ridge). To use the model, you input your data’s feature values into the linear equation, just like in standard Linear Regression.

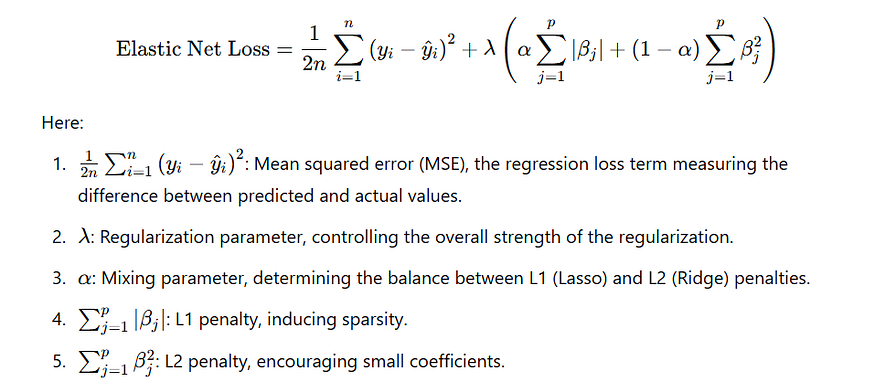
The main advantage of Elastic Net is that when features are related, it tends to keep or remove them as a group instead of randomly picking one feature from the group.

**What is Elastic Net Regression?**

Elastic Net Regression is an extension of linear regression that incorporates both L1 (Lasso) and L2 (Ridge) regularization penalties into the loss function. This blending allows Elastic Net to handle situations where there are a large number of features, and some of them are highly correlated.

**Formula**

Zoom image will be displayed



βj​ represents the **weights or coefficients** of the regression model.

**What is λ?**

* λ determines the strength of the regularization term, which penalizes large values of the coefficients (βj​).
* It balances the trade-off between:
* **Minimizing the prediction error** (measured by MSE).
* **Controlling the complexity of the model** (by shrinking the coefficients).

When λ=0:

* No regularization is applied. The model minimizes only the prediction error (ordinary linear regression).
* Coefficients (βj​) can grow large, leading to overfitting if the model is too complex.

When λ is large:

* The penalty term dominates, shrinking the coefficients (βj) significantly toward zero.

**Example to Illustrate λ**

Suppose we have a dataset for predicting house prices based on features like size, number of rooms, and location. The coefficients (βj​) represent the influence of each feature on the house price.

**No Regularization (λ=0):**

* The model fits the data perfectly, including noise.
* Coefficients (βj​) might be large:
* Example: β1=500,β2=200,β3=800.

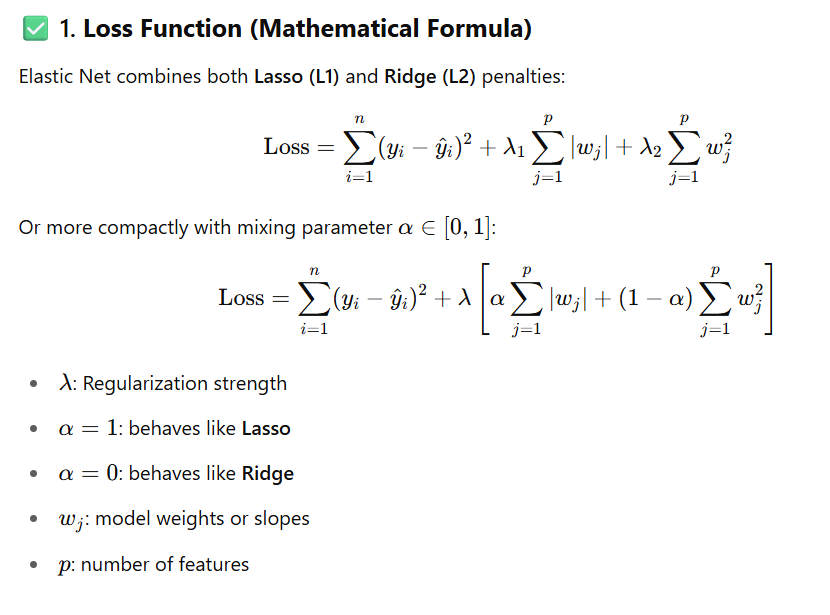
**Small λ (Mild Regularization):**

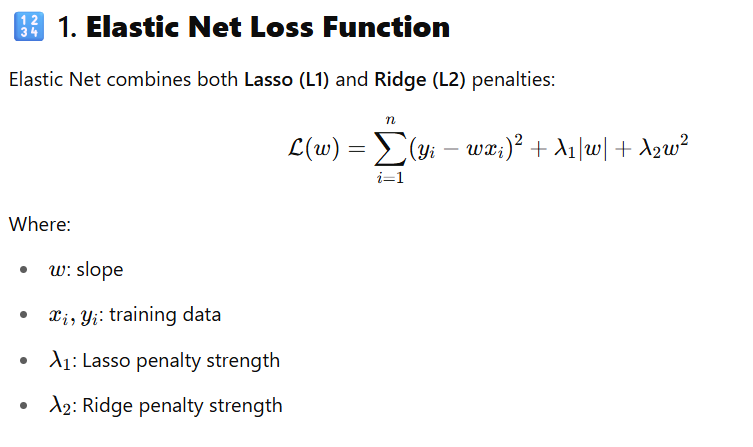
* Adds a penalty, slightly shrinking the coefficients:
* Example: β1=450,β2=180,β3=700.
* Reduces the effect of noise while retaining the predictive power.

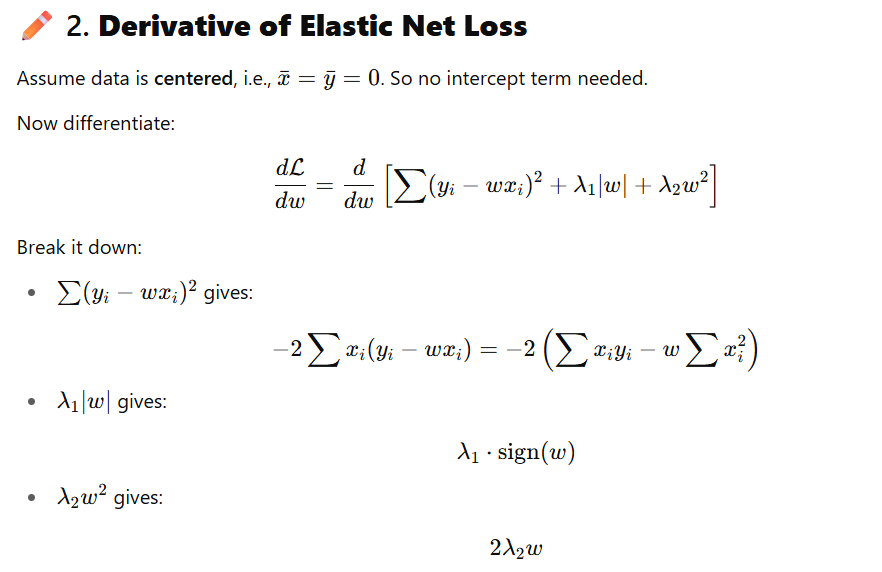
**Large λ (Strong Regularization):**

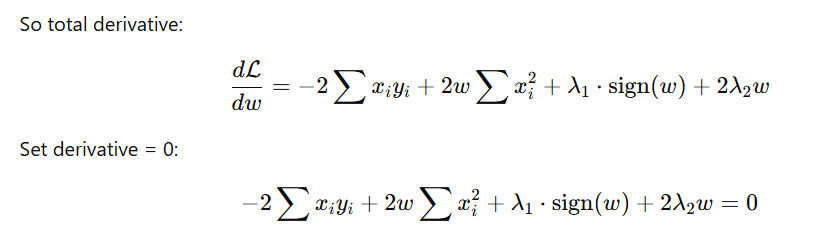
* Coefficients are heavily penalized and shrink close to zero:
* Example: β1=50,β2=10,β3=20.
* This prevents overfitting but may oversimplify the model, losing valuable information.

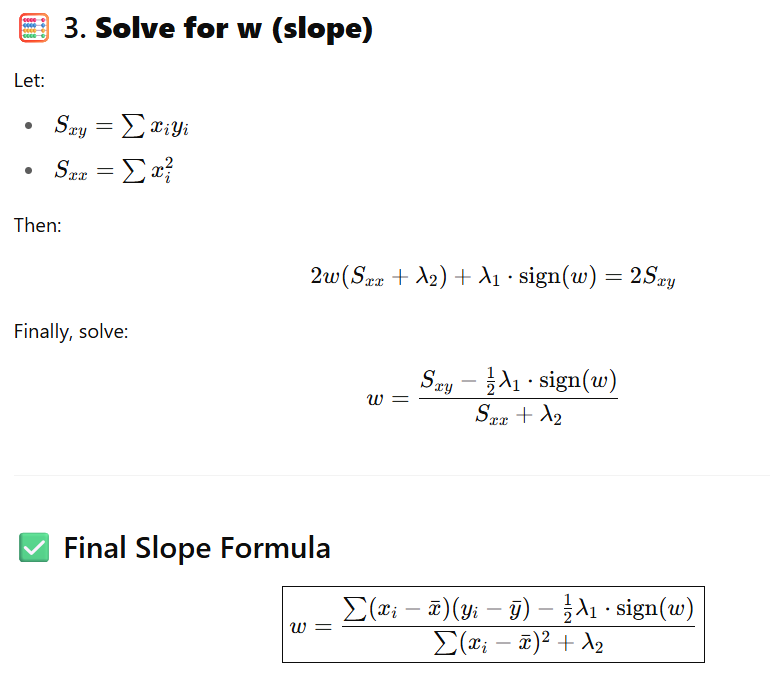
Mathematical intuition:



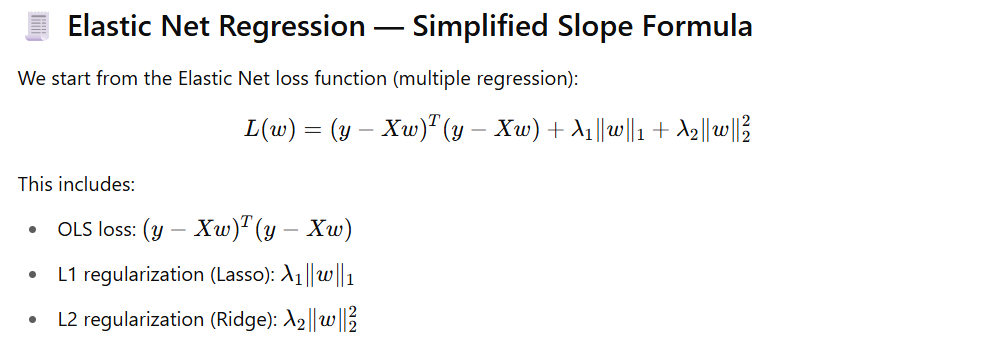


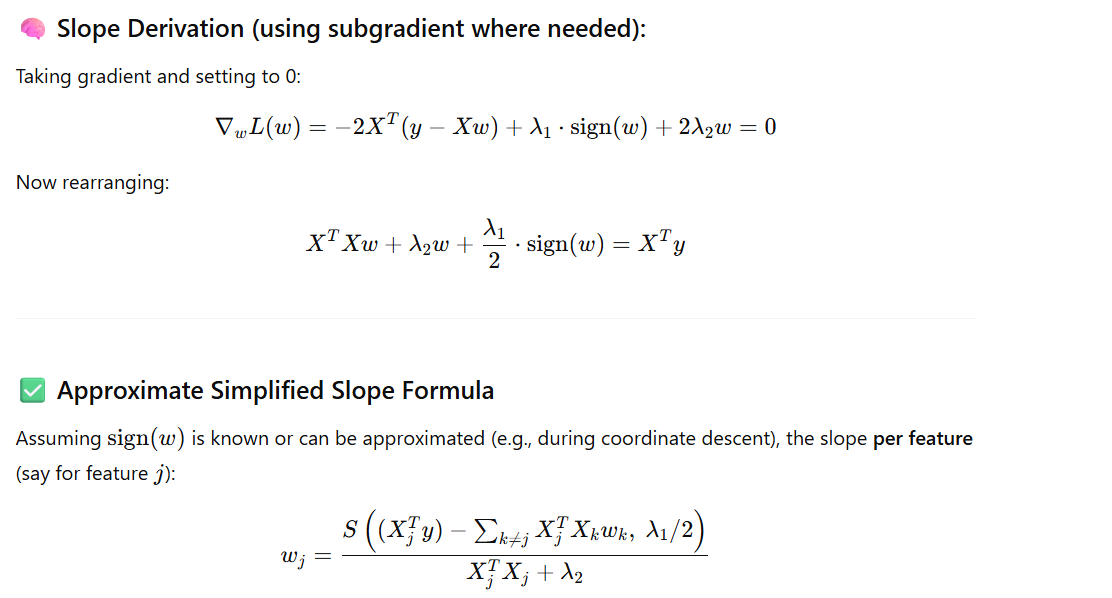


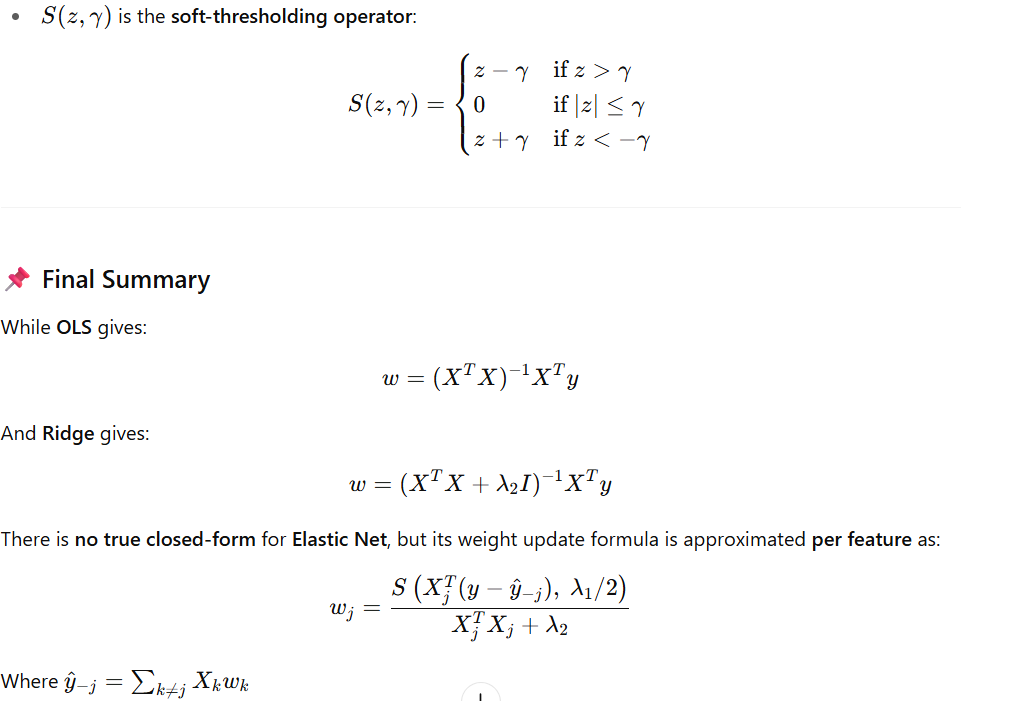


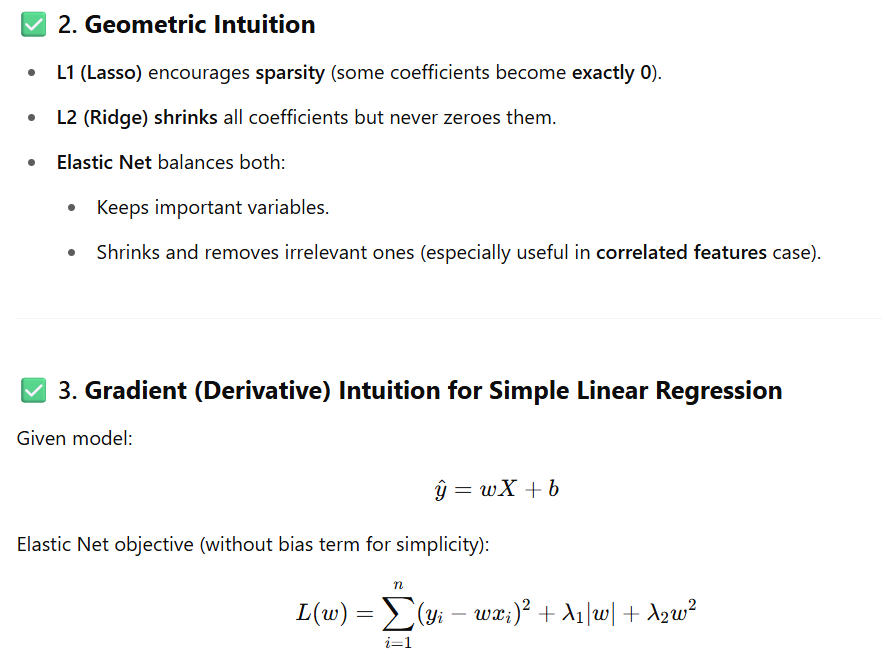


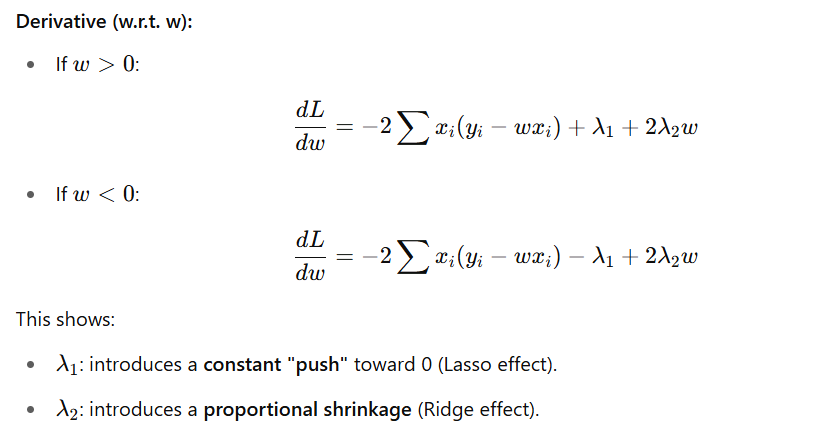
**Multiple linear regression:**

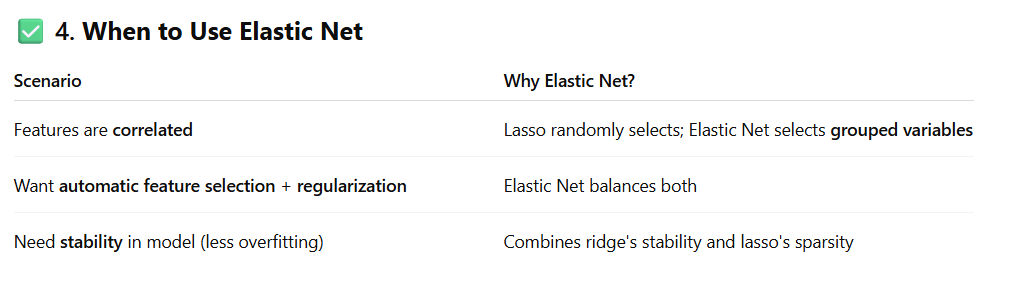












**✅ 5. Bias-Variance Trade-off**

* Increasing **λ**:
  + **Bias increases**
  + **Variance decreases**
  + Similar to ridge/lasso
* Tuning **α**:
  + α ≈ 0: lower bias, retains all variables (Ridge-like)
  + α ≈ 1: higher sparsity, more bias (Lasso-like)

**✅ 6. Elastic Net in Interview: Sample Q&A**

**Q1: What is Elastic Net Regression?**  
**A:** It's a regularized regression combining both L1 (lasso) and L2 (ridge) penalties to balance sparsity and coefficient shrinkage.

**Q2: Why not just use Ridge or Lasso?**  
**A:** When features are correlated or numerous, Lasso may randomly pick one; Ridge keeps all. Elastic Net selects correlated groups and shrinks them together.

**Q3: What role does α play in Elastic Net?**  
**A:** It controls the mix of L1 and L2:

* α = 1: pure Lasso
* α = 0: pure Ridge
* α ∈ (0, 1): balance of both

**Q4: Can Elastic Net set coefficients to zero?**  
**A:** Yes, due to the L1 part. It retains sparsity like Lasso.

**1. What is Elastic Net Regression?**

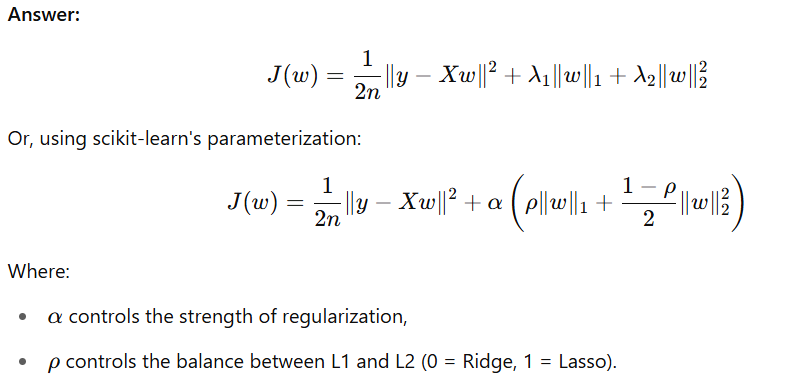
**Answer:**  
Elastic Net is a regularized regression technique that combines **L1 (Lasso)** and **L2 (Ridge)** penalties. It helps in both **feature selection** and **coefficient shrinkage**, especially when the features are highly correlated.

**2. Why do we use Elastic Net instead of just Lasso or Ridge?**

**Answer:**

* **Lasso (L1)** can zero out coefficients, but struggles when features are highly correlated.
* **Ridge (L2)** handles multicollinearity well but doesn’t perform feature selection.
* **Elastic Net** combines both, so it benefits from **L1 sparsity** and **L2 stability**, making it ideal for high-dimensional or correlated datasets.

**3. What is the cost function of Elastic Net?**



**4. What are hyperparameters in Elastic Net?**

**Answer:**

* **alpha (λ)**: Controls overall strength of regularization.
* **l1\_ratio (ρ)**: Controls the mix between L1 and L2.
  + ρ = 1 → Lasso
  + ρ = 0 → Ridge
  + 0 < ρ < 1 → Elastic Net

**5. When should you prefer Elastic Net over Lasso or Ridge?**

**Answer:**  
Use Elastic Net when:

* You have **many features** and **some are correlated**.
* You need **automatic feature selection** (like Lasso).
* Lasso zeroes out too many features or performs poorly due to multicollinearity.

**🔶 Intermediate Level**

**6. Explain the intuition behind combining L1 and L2 penalties.**

**Answer:**

* **L1 penalty** adds sparsity (zeroes out coefficients), but can behave unpredictably with correlated features.
* **L2 penalty** shrinks coefficients smoothly and keeps all features.
* **Combining** both allows for **sparse but stable** models that:
  + Eliminate some coefficients (L1),
  + Distribute weight among correlated variables (L2).

**7. How does Elastic Net help with multicollinearity?**

**Answer:**  
Elastic Net distributes coefficient weights across correlated features (like Ridge), avoiding the instability Lasso may introduce, while still allowing zeroing out (via L1) for unimportant features.

**8. What is the geometric interpretation of Elastic Net?**

**Answer:**

* Lasso’s constraint is a **diamond (L1 ball)** → corners encourage sparsity.
* Ridge’s constraint is a **circle (L2 ball)** → smooth shrinkage.
* Elastic Net’s constraint is a **mixture** (a rounded diamond), enabling sparsity **and** shrinkage simultaneously.

**9. Does Elastic Net always select a subset of features?**

**Answer:**  
Not necessarily. The **L1 component** allows zeroing out, but the **L2 component** can reduce sparsity. If ρ\rhoρ is closer to 0 (more L2), the model tends to retain more features.

**10. What if you set l1\_ratio=0 or l1\_ratio=1?**

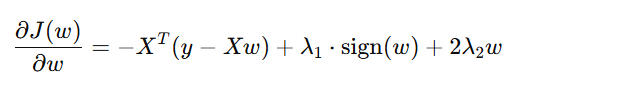
**Answer:**

* l1\_ratio = 0: Model becomes pure Ridge regression.
* l1\_ratio = 1: Model becomes pure Lasso regression.
* Values between 0 and 1 yield Elastic Net behavior.

**🔴 Advanced Level**

**11. What is the derivative of the Elastic Net loss function?**

**Answer:**  
For a simplified case with convexity and differentiability (assuming sign(w)\text{sign}(w)sign(w) is defined):



Gradient descent or coordinate descent is often used for optimization.

**12. How do you optimize Elastic Net?**

**Answer:**

* **Coordinate Descent** is the most popular algorithm.
  + It updates each coefficient iteratively while keeping others fixed.
* Due to the **non-differentiability** of the L1 term at zero, special handling (like soft-thresholding) is required.

**13. What are some challenges in tuning Elastic Net?**

**Answer:**

* Finding the **optimal combination of alpha and l1\_ratio** can be computationally expensive.
* Grid search with cross-validation is commonly used.

**14. How does Elastic Net perform feature selection?**

**Answer:**  
The **L1 component** in Elastic Net drives coefficients to zero, effectively performing feature selection. However, the **L2 component** stabilizes coefficients of correlated features, often preventing over-selection or under-selection.

**15. In what scenarios might Elastic Net underperform?**

**Answer:**

* If most features are uncorrelated and few are informative, **Lasso** may work better.
* If all features are useful and multicollinearity is low, **Ridge** may suffice.
* **Elastic Net** adds complexity due to dual regularization terms.
* When you want **benefits of both L1 and L2 regularization**.

**📌 Key Points**

| **Method** | **Can remove features?** | **Handles multicollinearity?** | **Shrinks coefficients** |
| --- | --- | --- | --- |
| Lasso (L1) | ✅ Yes | ❌ Poor | ✅ Yes |
| Ridge (L2) | ❌ No | ✅ Good | ✅ Yes |
| Elastic Net | ✅ Yes | ✅ Good | ✅ Yes |

**🧪 Code Example: Elastic Net Regression with Explanation**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import ElasticNet, LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

# 1. Generate synthetic data

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X[:, 0] + np.random.randn(100)

# Add multicollinearity (second feature is correlated with first)

X = np.c\_[X, X[:, 0] + 0.01 \* np.random.randn(100)]

# 2. Split train/test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 3. Fit Linear Regression

lr = LinearRegression()

lr.fit(X\_train, y\_train)

y\_pred\_lr = lr.predict(X\_test)

# 4. Fit Elastic Net

en = ElasticNet(alpha=0.1, l1\_ratio=0.5, random\_state=42) # 50% L1, 50% L2

en.fit(X\_train, y\_train)

y\_pred\_en = en.predict(X\_test)

# 5. Evaluate

print("🔷 Linear Regression:")

print(" Coefficients:", lr.coef\_)

print(" Intercept:", lr.intercept\_)

print(" MSE:", mean\_squared\_error(y\_test, y\_pred\_lr))

print(" R² Score:", r2\_score(y\_test, y\_pred\_lr))

print("\n🔷 Elastic Net Regression:")

print(" Coefficients:", en.coef\_)

print(" Intercept:", en.intercept\_)

print(" MSE:", mean\_squared\_error(y\_test, y\_pred\_en))

print(" R² Score:", r2\_score(y\_test, y\_pred\_en))

# 6. Plot

plt.figure(figsize=(10, 6))

plt.scatter(X\_test[:, 0], y\_test, color='blue', label='True Data')

plt.scatter(X\_test[:, 0], y\_pred\_lr, color='green', label='Linear Predicted')

plt.scatter(X\_test[:, 0], y\_pred\_en, color='red', label='Elastic Net Predicted', marker='x')

plt.xlabel("Feature X")

plt.ylabel("Target y")

plt.title("Linear vs Elastic Net Regression")

plt.legend()

plt.grid(True)

plt.show()

**Analysis of Results**

**Linear Regression**

* May overfit if features are highly correlated.
* Coefficients can be unstable in presence of multicollinearity.

**Elastic Net**

* Coefficients are **shrunk**, improving generalization.
* Can **zero out** some coefficients (if l1\_ratio is high).
* Helps handle **collinearity** better than Lasso alone.

**🔧 Parameter Explanation**

| **Parameter** | **Description** |
| --- | --- |
| alpha | Regularization strength (higher = more penalty) |
| l1\_ratio | Mix between Lasso (1.0) and Ridge (0.0) |
| random\_state | For reproducibility |
| fit\_intercept | Whether to estimate intercept (default True) |

**✅ Summary**

**Elastic Net** is the best of both worlds:

* Lasso's ability to remove irrelevant features
* Ridge's robustness to correlated predictors

It’s often used when:

* You have **many features**
* You want to **reduce overfitting**
* You want **some feature selection** but with **stability**

**What are Correlated Features?**

**Correlation** between features means that **one feature can be linearly predicted from another** with some degree of accuracy.

* If Feature A increases and Feature B also increases, they are **positively correlated**.
* If Feature A increases and Feature B decreases, they are **negatively correlated**.
* If there is **no linear relationship**, they are **uncorrelated**.

📌 Correlation is measured by **Pearson's correlation coefficient (r)**, which ranges from -1 to +1.

**🔷 What is Multicollinearity?**

**Multicollinearity** is a condition where **two or more independent variables (features) in a regression model are highly correlated**.

This is a problem because:

* It becomes hard to determine the effect of each feature independently.
* Coefficients become unstable (large or flip signs).
* It can lead to **overfitting** and **poor generalization**.

✅ Ridge and ElasticNet handle multicollinearity well by **shrinking coefficients**.

**🧪 Sample Data to Illustrate Correlation & Multicollinearity**

**🔹 Step 1: Generate Example**

python

CopyEdit

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

# Seed for reproducibility

np.random.seed(0)

# Create a feature X1

X1 = np.random.rand(100)

# Create another feature X2 highly correlated with X1

X2 = X1 + np.random.normal(0, 0.01, 100) # Add small noise

# Create a target variable y based on X1

y = 3 \* X1 + np.random.normal(0, 0.1, 100)

# Combine into DataFrame

df = pd.DataFrame({'X1': X1, 'X2': X2, 'y': y})

print(df.head())

**🔹 Step 2: Visualize Correlation Matrix**

python

CopyEdit

# Compute and plot correlation matrix

corr = df.corr()

sns.heatmap(corr, annot=True, cmap='coolwarm')

plt.title("Correlation Matrix")

plt.show()

**💡 Expected Output:**

markdown

CopyEdit

X1 X2 y

0 0.548814 0.549401 1.753551

1 0.715189 0.716988 2.189402

2 0.602763 0.603061 1.832431

3 0.544883 0.546506 1.566114

4 0.423655 0.425522 1.325846

**🔸 Correlation Matrix Output (example):**

markdown

CopyEdit

X1 X2 y

X1 1.00 0.999 0.97

X2 0.999 1.00 0.97

y 0.97 0.97 1.00

🔥 Here, **X1 and X2 have a correlation of 0.999**, meaning they're almost **linearly dependent**.

**🔍 Why Is This a Problem?**

Imagine using both X1 and X2 in linear regression:

python

CopyEdit

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(df[['X1', 'X2']], df['y'])

print("Coefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

You might get weird or unstable coefficients like:

makefile

CopyEdit

Coefficients: [ 8.3 -5.2 ]

This happens because the model cannot distinguish the separate effects of X1 and X2 since they are almost the same.

**🛠 How to Detect Multicollinearity?**

* **Correlation matrix** (as we did above)
* **Variance Inflation Factor (VIF)**: VIF > 5 or 10 is a red flag

python

CopyEdit

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# Calculate VIF for each feature

X = df[['X1', 'X2']]

vif\_data = pd.DataFrame()

vif\_data["Feature"] = X.columns

vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

print(vif\_data)

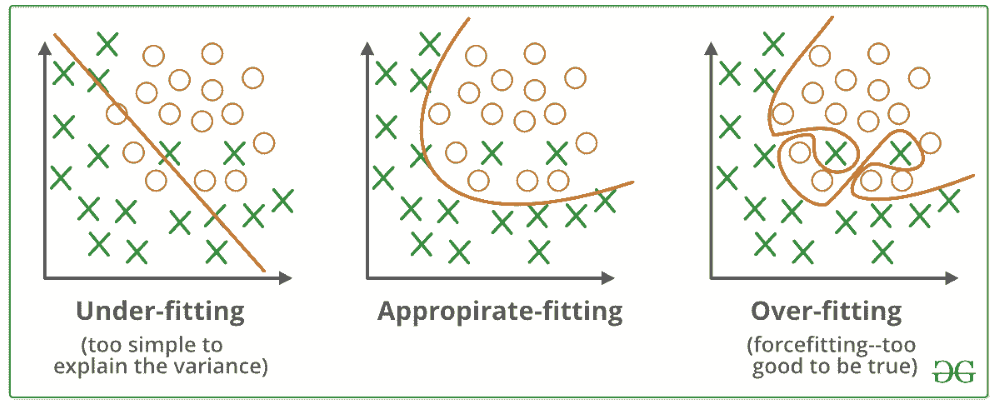
This might show very high VIF values (~100+), confirming **multicollinearity**.

**✅ Summary**

| **Term** | **Meaning** |
| --- | --- |
| Correlated Feature | Two or more features that move together (have a high correlation) |
| Multicollinearity | Multiple features are correlated, affecting model stability |
| Problem? | Yes. Makes coefficients unreliable in linear models like OLS regression |
| Solution | Use **Ridge** or **Elastic Net** to handle it |

**What are Overfitting and Underfitting?**

**Overfitting** and **underfitting**are terms used to describe the performance of machine learning models in relation to their ability to generalize from the training data to unseen data.



[**Overfitting**](https://www.geeksforgeeks.org/how-to-handle-overfitting-in-tensorflow-models/) happens when a machine learning model learns the training data too well including the noise and random details. This makes the model to perform poorly on new, unseen data because it memorizes the training data instead of understanding the general patterns.

For example, if we only study last week’s weather to predict tomorrow’s i.e our model might focus on one-time events like a sudden rainstorm which won’t help for future predictions.

[**Underfitting**](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/) is the opposite problem which happens when the model is too simple to learn even the basic patterns in the data. An underfitted model performs poorly on both training and new data. To fix this we need to make the model more complex or add more features.

For example if we use only the average temperature of the year to predict tomorrow’s weather hence the model misses important details like seasonal changes which results in bad predictions.

**What are Bias and Variance?**

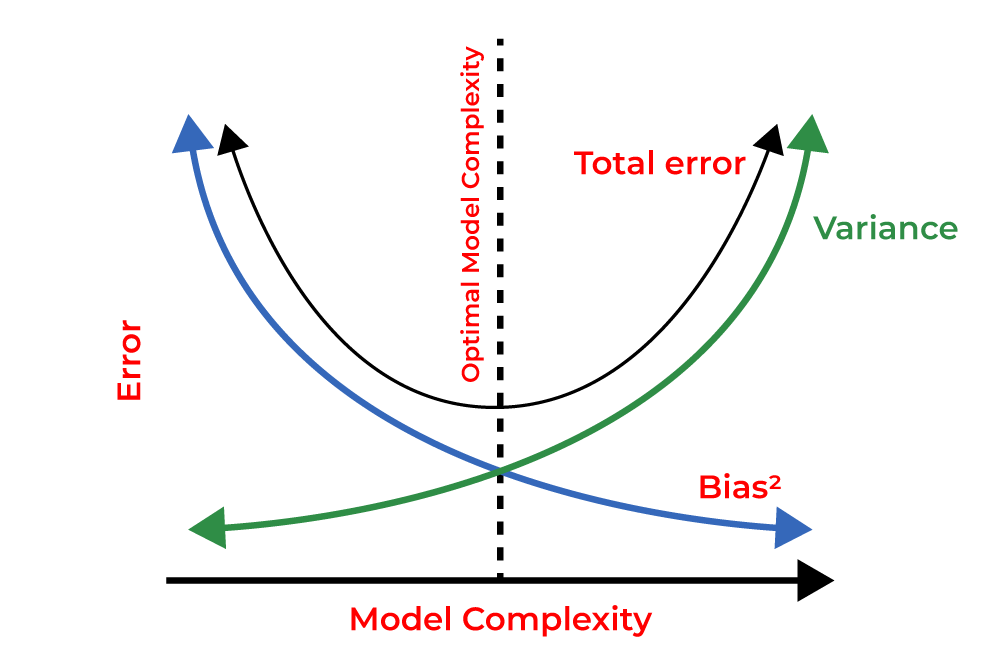
* **Bias** refers to the errors which occur when we try to fit a statistical model on real-world data which does not fit perfectly well on some mathematical model. If we use a way too simplistic a model to fit the data then we are more probably face the situation of **High Bias** (underfitting) refers to the case when the model is unable to learn the patterns in the data at hand and perform poorly.
* **Variance** shows the error value that occurs when we try to make predictions by using data that is not previously seen by the model. There is a situation known as **high variance** (overfitting) that occurs when the model learns noise that is present in the data.

Finding a proper balance between the two is also known as the **Bias-Variance Tradeoff** which helps us to design an accurate model.

**Bias Variance tradeoff**

The [**Bias-Variance Tradeoff**](https://www.geeksforgeeks.org/ml-bias-variance-trade-off/)refers to the balance between bias and variance which affect predictive model performance. Finding the right tradeoff is important for creating models that generalize well to new data.

* The **bias-variance tradeoff**shows the inverse relationship between bias and variance. When one decreases, the other tends to increase and vice versa.
* Finding the right balance is important. An overly simple model with high bias won't capture the underlying patterns while an overly complex model with high variance will fit the noise in the data.



**Benefits of Regularization**

Now, let’s see various benefits of regularization which are as follows:

1. **Prevents Overfitting:** Regularization helps models focus on underlying patterns instead of memorizing noise in the training data.
2. **Improves Interpretability:** L1 (Lasso) regularization simplifies models by reducing less important feature coefficients to zero.
3. **Enhances Performance:** Prevents excessive weighting of outliers or irrelevant features helps in improving overall model accuracy.
4. **Stabilizes Models:** Reduces sensitivity to minor data changes which ensures consistency across different data subsets.
5. **Prevents Complexity:** Keeps model from becoming too complex which is important for limited or noisy data.
6. **Handles Multicollinearity:** Reduces the magnitudes of correlated coefficients helps in improving model stability.
7. **Allows Fine-Tuning:** Hyperparameters like alpha and lambda control regularization strength helps in balancing bias and variance.
8. **Promotes Consistency:** Ensures reliable performance across different datasets which reduces the risk of large performance shifts.