

# Machine Learning

## Video 66:

### Lasso Regression | Intuition and Code Sample | Regularized Linear Models

Lasso regression is a machine learning technique used for linear regression that adds an L1 penalty to the loss function. This regularization helps prevent overfitting by shrinking some coefficients to zero, effectively selecting important features and improving model generalization, especially in high-dimensional datasets.

The image shows handwritten notes on a blackboard. At the top, it says 'L1 Regularization | overfitting' with an arrow pointing to 'L2 reg'. Below this, the loss function is written as  $L = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \|w\|_1^2$ . A note below the second term says 'L2 reg' and 'L2 reg' with an arrow pointing to the term. Below the loss function, it says '→ 0' and 'w<sub>1</sub> → w<sub>n</sub> → coeff'. Below this, it says 'Lasso' with an arrow pointing to the loss function. To the right, it says  $\lambda [ |w_1| + |w_2| + |w_3| + \dots + |w_n| ]$ . At the bottom, it says '→ L = MSE + λ ||w|| → code implementation'.

The choice between Lasso and linear regression depends on the problem:

- **Linear regression** is preferred when you believe all features are important and there's no need for regularization.
- **Lasso regression** is better when you have many features and suspect that some are irrelevant, as it performs feature selection by shrinking coefficients to zero.

Lasso helps avoid overfitting, especially in high-dimensional datasets, making it useful when feature reduction is needed.

Example:

Code link:

<https://colab.research.google.com/drive/1nKH9UHqSBclvED5dLPWRb62qn73xMLyR?usp=sharing>

Here are the key points regarding **Lasso Regression**:

1. **How are coefficients affected?**
  - a. Lasso regression applies L1 regularization, which penalizes the absolute values of the coefficients. This results in some coefficients being shrunk to zero, effectively performing **feature selection**.
2. **Higher coefficients are affected more:**

- a. Larger coefficients tend to be penalized more heavily because Lasso uses the absolute value of coefficients. However, Lasso can shrink them to zero, completely eliminating some features from the model.
- 3. **Impact on bias and variance due to lambda:**
  - a. **Bias:** As lambda (the regularization parameter) increases, Lasso shrinks more coefficients to zero, which can increase bias by simplifying the model.
  - b. **Variance:** A larger lambda reduces the model's complexity, decreasing variance and overfitting, but too large a lambda can lead to underfitting.
- 4. **Effect of regularization on the loss function:**
  - a. The loss function in Lasso regression consists of the **ordinary least squares error** (residual sum of squares) plus the **L1 penalty** (lambda times the sum of absolute values of coefficients). This regularization term discourages large coefficients, balancing fit and complexity. The loss function is:

$$\text{Loss} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

## Video 67:

### Why Lasso Regression creates sparsity?

Lasso regression creates sparsity because of its use of **L1 regularization** (the absolute value penalty on coefficients). The L1 penalty encourages some of the regression coefficients to shrink to exactly **zero** during the model fitting process.

Here's why it leads to sparsity:

1. **L1 Regularization:** The Lasso cost function includes the sum of absolute values of the coefficients, scaled by the regularization parameter  $\lambda$  (lambda). This penalty forces the model to minimize not just the residual sum of squares (error) but also the magnitude of the coefficients.
2. **Shrinking Coefficients to Zero:** Unlike Ridge regression (which uses L2 regularization and shrinks coefficients towards zero but never to exactly zero), Lasso can force coefficients to become **exactly zero**. This is because the L1 penalty creates a sharp "corner" at zero, leading some coefficients to be eliminated entirely, thereby selecting a subset of features.
3. **Feature Selection:** As a result, Lasso performs **feature selection** by setting some feature coefficients to zero, essentially ignoring less relevant features. This creates a **sparse model** where only the most important features are kept.

In summary, Lasso regression creates sparsity because the L1 regularization encourages coefficient shrinkage, and some coefficients are driven to zero, making the model more interpretable and reducing overfitting.

**Lasso**

for  $m > 0$

$$m = \frac{\sum (y_i - \bar{y})(x_i - \bar{x}) - \lambda}{\sum (x_i - \bar{x})^2}$$

for  $m = 0$

$$m = \frac{\sum (y_i - \bar{y})(x_i - \bar{x})}{\sum (x_i - \bar{x})^2}$$

for  $m < 0$

$$m = \frac{\sum (y_i - \bar{y})(x_i - \bar{x}) + \lambda}{\sum (x_i - \bar{x})^2}$$

Example calculations:

$\lambda = 100$

$$m = \frac{100 - 100}{50} = 0$$

$\lambda = 10$

$$m = \frac{100 - 10}{50} = \frac{90}{50} = 1.8$$

$\lambda = 50$

$$m = \frac{100 - 50}{50} = 1$$

$\lambda = 150$

$$m = \frac{100 + 150}{50} = \frac{250}{50} = 5$$

Ridge regression doesn't create sparsity because it uses **L2 regularization** (the square of the coefficients in the penalty term), which shrinks the coefficients towards zero but **never** forces them to exactly zero. Here's why:

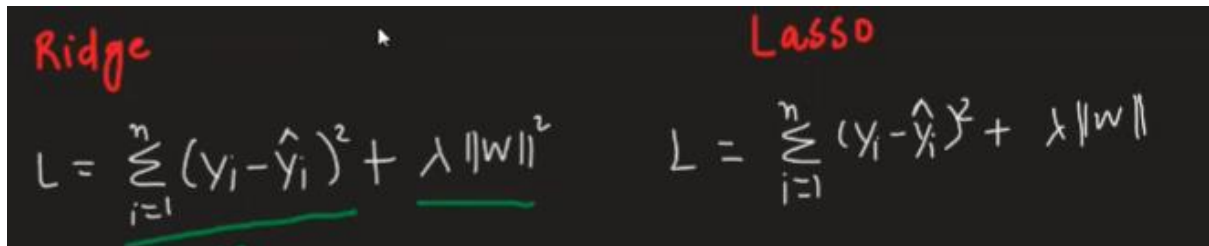
1. **L2 Regularization:** The penalty term in Ridge regression is the sum of the squares of the coefficients, scaled by a regularization parameter  $\lambda$ . The L2 penalty encourages small coefficients but does not create sharp corners or discontinuities at zero.
2. **Smooth Shrinkage:** L2 regularization applies a **continuous, smooth shrinkage** to the coefficients, meaning it reduces their magnitude but doesn't set them exactly to zero. This means all features remain in the model, but with smaller weights.
3. **No Feature Selection:** Because Ridge doesn't force coefficients to zero, all features are included in the model, regardless of their relevance. It only reduces their contribution by making their coefficients smaller, but doesn't discard any of them. This leads to **no sparsity** in the model.

In contrast to **Lasso**, which actively performs **feature selection** by setting coefficients to zero, **Ridge** reduces the impact of less relevant features but keeps them in the model, leading to a less interpretable model with no sparsity.

## Video 68:

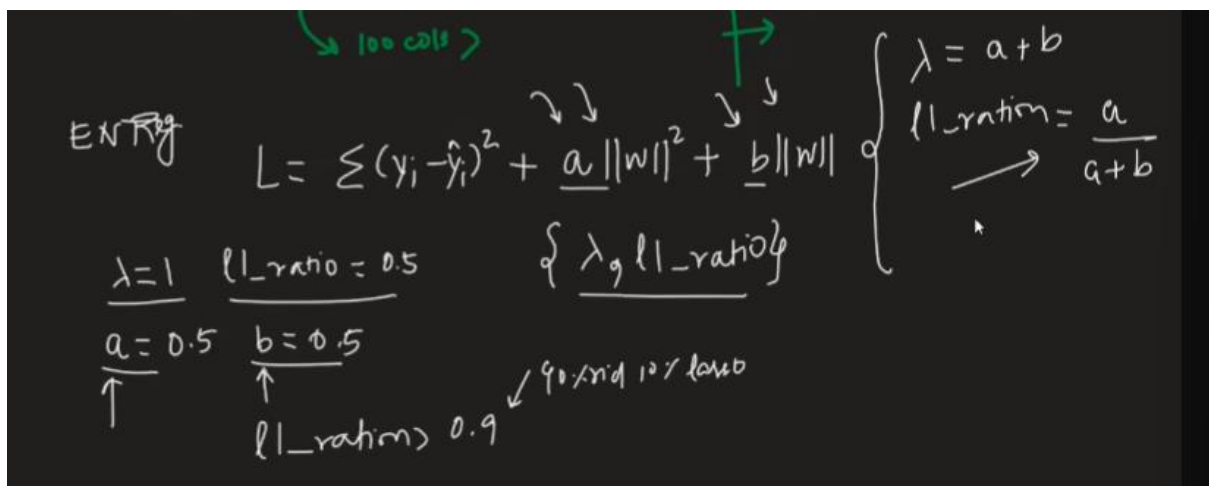
### ElasticNet Regression | Intuition and Code Example | Regularized Linear Models

We know, Ridge is used where all columns are important, while Lasso is used where subset of columns are important.



The image shows two handwritten equations on a black background. On the left, under the word 'Ridge' in red, is the formula  $L = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \|w\|^2$ . On the right, under the word 'Lasso' in red, is the formula  $L = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \|w\|$ .

Now, Elastic Net regression is a combination of Lasso and Ridge regression, using both L1 and L2 regularization. It leverages the strengths of both methods: Lasso's feature selection and Ridge's handling of correlated features. Elastic Net is particularly useful when there are many correlated features or when Lasso performs poorly.



The image shows a handwritten Elastic Net regression formula and its parameter breakdown. The formula is  $L = \sum (y_i - \hat{y}_i)^2 + a \|w\|^2 + b \|w\|$ . To the right of the formula is a bracketed section containing  $\lambda = a + b$  and  $l1\_ratio = \frac{a}{a+b}$ . Below the formula, there are two rows of parameters:  $\lambda = 1$  and  $l1\_ratio = 0.5$ , and  $a = 0.5$  and  $b = 0.5$ . Arrows point from these parameters to the formula. A note at the bottom right says '90% and 10% ratio' with an arrow pointing to the  $l1\_ratio$  parameter.

Example:

Code link:

<https://colab.research.google.com/drive/1nKH9UHqSBclvED5dLPWRb62qn73xMlyR?usp=sharing>

## Video 69:

### Logistic Regression Part 1 | Perceptron Trick

#### **Logistic**

#### **Aspect:**

Logistic regression uses the logistic (sigmoid) function to model the relationship between input features and the binary output. It maps predicted values into the range of 0 to 1, ensuring the outcome represents probabilities rather than raw scores. This is ideal for classification tasks in machine learning.

#### **Probability**

#### **Aspect:**

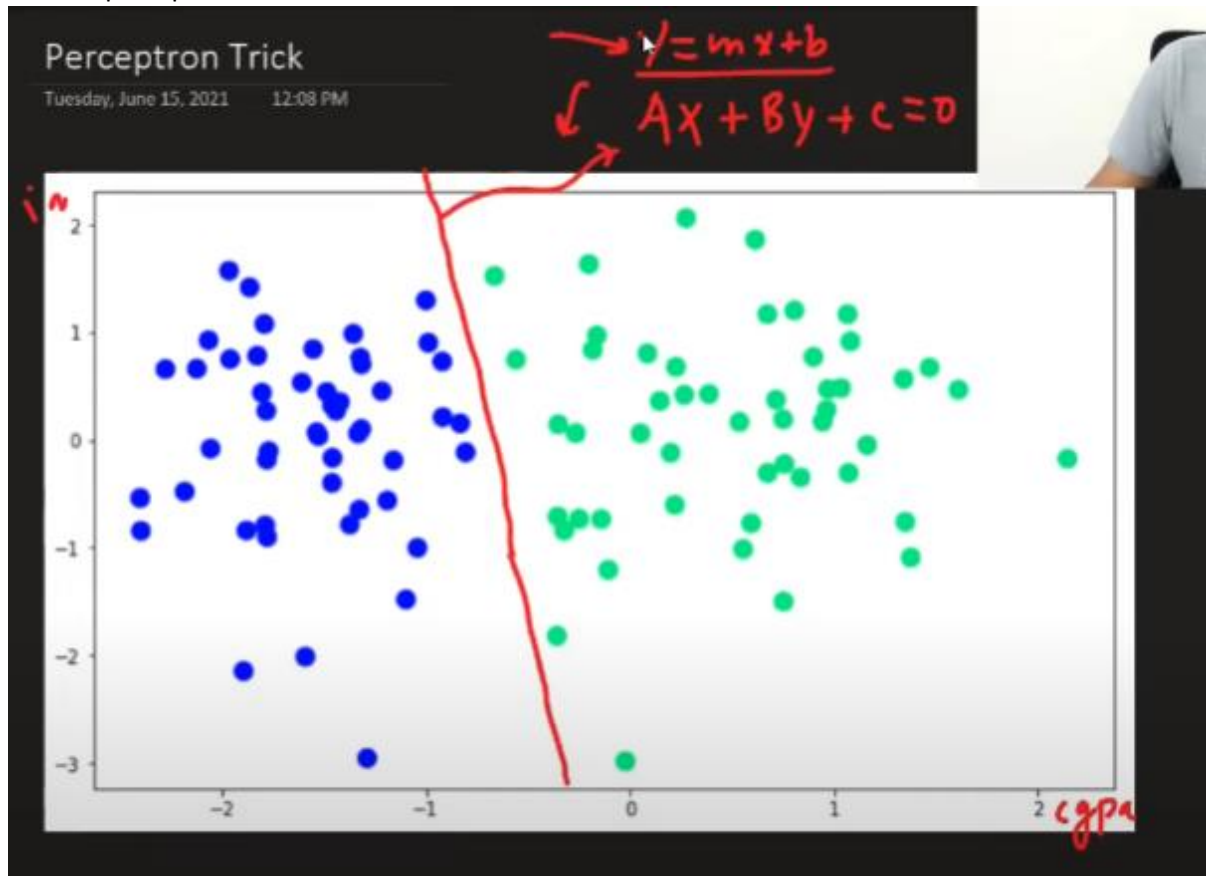
In logistic regression, the output is interpreted as a probability of belonging to a certain class. It estimates the likelihood that a given input belongs to the positive class by applying the sigmoid function, converting linear predictions into a probability value between 0 and 1.

The prerequisites of data for applying logistic regression include:

1. **Binary or Categorical Outcome Variable:** Logistic regression is typically used for binary classification (e.g., 0 or 1, Yes or No). For multiclass problems, multinomial logistic regression is used.
2. **Numerical or Categorical Predictor Variables:** Input features can be continuous (numerical) or categorical. Categorical variables should be encoded (e.g., one-hot encoding) to be usable in the model.
3. **Independence of Observations:** The observations (data points) should be independent of each other, as logistic regression assumes no correlation between them.
4. **No Multicollinearity:** The predictor variables should not be highly correlated with each other. High correlation (multicollinearity) can make the model's estimates unreliable.
5. **Sufficient Sample Size:** Logistic regression requires a sufficient number of observations to avoid overfitting and to provide reliable estimates. Too few data points can lead to unstable models.
6. **Linearity in the Logit:** There should be a linear relationship between the log-odds of the dependent variable and the independent variables. This doesn't mean the variables themselves need to be linear but the log-transformed outcomes should be linear in relation to the predictors.
7. **Absence of Outliers:** While logistic regression can handle some outliers, extreme values in predictor variables may distort the model and should be treated before applying the model.

These data requirements ensure that logistic regression works optimally and produces meaningful results.

What is perceptron trick?



The **Perceptron trick** is a technique used to improve the performance of the Perceptron algorithm, particularly when dealing with the learning of linearly separable data. It is a simple yet effective method to adjust the weights of the Perceptron during training and is based on a clever use of the inner product of the input vector and the weights.

In the context of the Perceptron algorithm, the **Perceptron trick** involves:

1. **Updating Weights:** When a misclassified data point is encountered, the algorithm updates the weights by adding or subtracting the input vector to the weight vector. This is done based on the sign of the misclassification.
2. **Learning Rate:** The learning rate (step size) is typically set to a fixed value, and it governs how large the weight updates are. The Perceptron trick often doesn't involve complex hyperparameter tuning, as it is a simple update rule based on the mistakes made by the model.

#### Basic Perceptron Update Rule (The Trick):

- If the prediction for an input  $x$  is incorrect, update the weights  $w$  as follows:

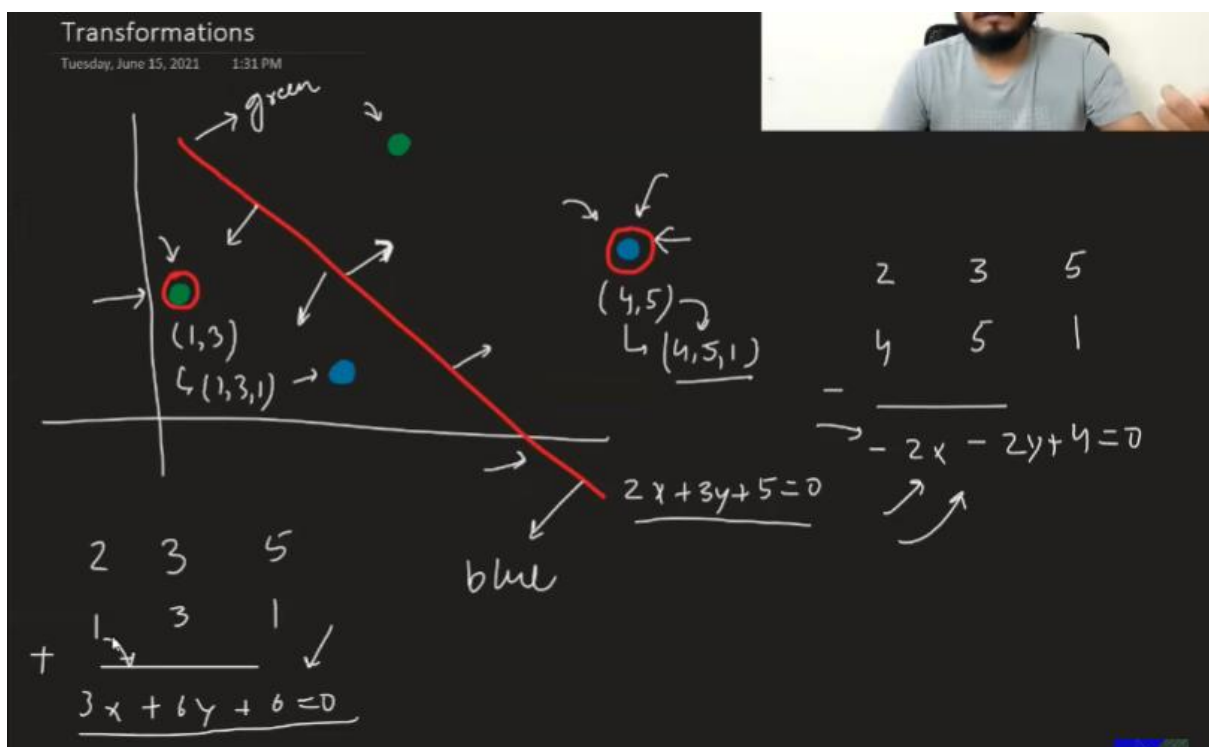
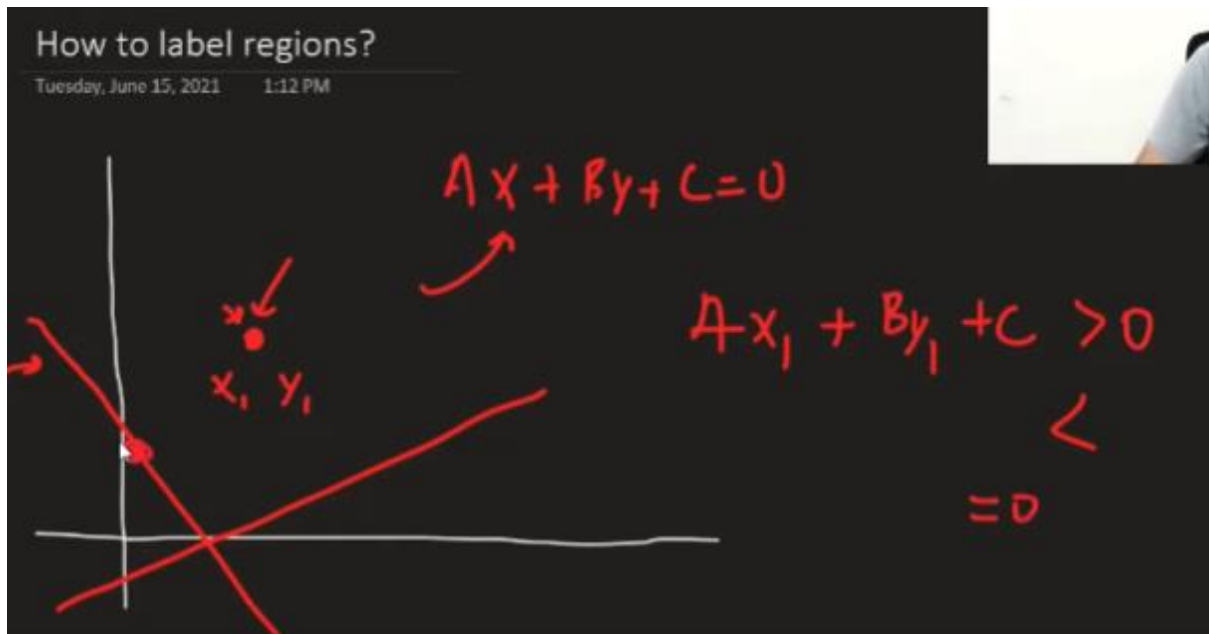
$$w = w + y_i \cdot x_i$$

- 
- Where  $y_i$  is the true label (either +1 or -1), and  $x_i$  is the input vector.

The trick allows the model to "learn" the correct decision boundary by iteratively adjusting the weights in response to classification errors. Over time, the Perceptron converges (if the data is linearly separable), and it can classify new inputs correctly based on the learned weights.

### Significance:

- The Perceptron trick makes the learning process efficient by focusing updates only on misclassified points.
- It's fundamental in understanding more advanced algorithms like Support Vector Machines (SVMs), which use a similar concept with more sophisticated optimizations.



Algorithm:

epoch  $\rightarrow 1000$  ,  $\eta = 0.01$

for  $i$  in range (epochs):

randomly select a student

if  $x_i \in N$  and  $\sum_{i=0}^2 w_i x_i \geq 0$  { +ve



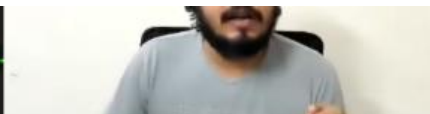
$w_{new} = w_{old} - \eta x_i$

if  $x_i \in P$  and  $\sum_{i=1}^2 w_i x_i < 0$

$w_{new} = w_{old} + \eta x_i$

$[w_0, w_1, w_2]$

$\left\{ \begin{array}{l} x_i \in N \quad w_i x_i < 0 \\ x_i \in P \quad w_i x_i \geq 0 \end{array} \right.$

### Simplified Algo

Tuesday, June 15, 2021 2:44 PM

$x_i \in N$  and  $\sum w_i x_i \geq 0$

$w_n = w_0 - \eta x_i$

$x_i \in P$  and  $\sum w_i x_i < 0$

$w_n = w_0 + \eta x_i$



## **Video 70:**

### **Logistic Regression Part 2 | Perceptron Trick**

Perceptron trick code:

Code link:

<https://colab.research.google.com/drive/1LYbsNLXhJW7sTX3FGvxRn6IDWE9zHody?usp=sharing>