Regularization

What is Regularization?

Regularization is a strategy that prevents overfitting by providing new knowledge to the machine learning algorithm. It is a type of regression. But here, the coefficient values are reduced to zero.

In layman's terms, "the Regularization approach reduces the size of the independent factors while maintaining the same number of variables." It keeps the model 's efficiency as well as its applicability.

When to use Regularization? :

1. Preventing Overfitting: Regularization is most commonly used as a tool to prevent overfitting. If your model performs well on the training data but poorly on the validation or test data, it might be overfitting, and regularization could help.
2. High Dimensionality: Regularization is particularly useful when you have a high number of features compared to the number of data points. In such scenarios, models tend to overfit easily, and regularization can help by effectively reducing the complexity of the model.
3. Multicollinearity: When features are highly correlated (multicollinearity), it can destabilize your model and make the model's estimates sensitive to minor changes in the model. L2 regularization (Ridge regression) can help in such cases by distributing the coefficient estimates among correlated features.
4. Feature Selection: If you have a lot of features and you believe many of them might be irrelevant, L1 regularization (Lasso) can help. It tends to produce sparse solutions, driving the coefficients of irrelevant features to zero, thus performing feature selection.
5. Interpretability: If model interpretability is important and you want a simpler model, regularization can help achieve this by constraining the model's complexity.
6. Model Performance: Even if you're not particularly worried about overfitting, regularization might still improve your model's out-of-sample prediction performance.

Three types :

1. Ridge Regression (R2)
2. Lasso Regression (R1)
3. Elastic Net Regression

Ridge Regularization

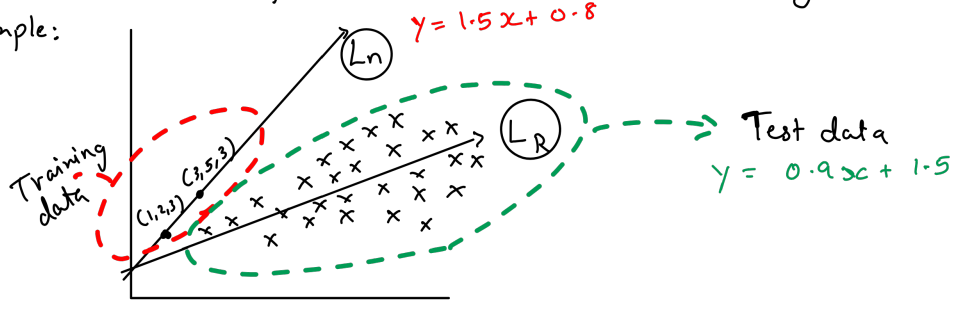
Ridge regularization, also known as L2 regularization, is a technique used in linear regression and other machine learning models to prevent overfitting by adding a penalty to the loss function based on the sum of squares of the model's coefficient.

Here, to reduce the overall loss function, we add one more regularization term along with the loss in the loss function

$$\text{Loss} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 + \underbrace{\lambda(m^2)}_{\text{extra term (This is a hyperparameter)}}$$

The overall loss function would be decreased by this above eq.

example:



Suppose Ln is the predicted line based on the training data and L_R is the actual line where it is supposed to be.

We can clearly see that there is overfitting, as the model is not performing well on a new data.

Calculating the Loss :

Loss Ln

$\lambda = 1$

$0 + (1.5)^2$

$= \boxed{2.25}$

Loss L_R

$\lambda = 1$

$(2.3 - 0.9 - 1.5)^2$

$+ (5.3 - 2.7 - 1.5)^2$

$+ (0.9)^2$

$= (0.1)^2 + (1.1)^2$

$+ (0.9)^2$

$= \boxed{2.03}$

$\therefore$ The loss function has been reduced.

$y = mx + b$

↓

The m is the most important value since, the more m value, the more loss

Deriving Mathematically for 2D

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda m^2$$

$$L = \sum_{i=1}^{n} (y_i - mx_i - b)^2 + \lambda m^2$$

$$L = \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})^2 + \lambda m^2$$

$b = \bar{y} - m\bar{x}$

where
- $\bar{y}$ → y mean
- $\bar{x}$ → x mean
- m → slope

Derivation

$$\frac{\partial L}{\partial m} = 2 \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})(-x_i + \bar{x}) + 2\lambda m = 0$$

$$= -2 \sum_{i=1}^{n} (y_i - \bar{y} - mx_i + m\bar{x})(x_i - \bar{x}) + 2\lambda m = 0$$

$$= \lambda m - \sum_{i=1}^{n} [(y_i - \bar{y}) - m(x_i - \bar{x})](x_i - \bar{x}) = 0$$ { Expanding

$$= \lambda m - \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) - m(x_i - \bar{x})^2 = 0$$

$$= \lambda m - \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) + m \sum_{i=1}^{n} (x_i - \bar{x})^2 = 0$$

$$= \lambda m + m \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})$$

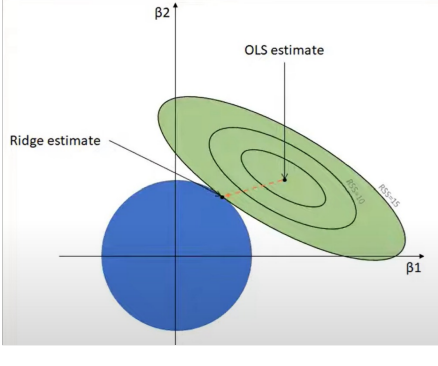
$$m = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2 + \lambda}$$

This is what really is ridge regression.

The value of m is same in LR except the λ in the denominator. The more value of λ, the less value of m.

Few Important takeaways. (Also refer to the coding files)

1. How are coefficients affected?
   All the coefficient will keep on reducing towards zero (never 0)
2. Higher value are impacted more
   → Meaning, if the value of coefficient are: 1000, 100, 10, then the one with 1000 will have more affect and will be reduced near zero faster than as compared to other ones.
3. Bias Variance Trade-off
   Bias and Variance depends on the value of λ (α (alpha))
   If the value of λ is very small (close to 0), then the Bias will be reduced as your model is overfitting and the variance will be increased and vice versa.
4. Impact on the Loss function (Review code)
   $$L = \sum_{i=0}^{n} (y_i - \hat{y}_i)^2 + \lambda \|w\|^2$$
   the value of m changes as we increase the value of lambda.
5. Why called Ridge?



Complex topic :—

Always make sure, when you have 2 or more the 2 input cols, then only apply ridge regression.

Lasso Regression (L1)

Lasso stands for Least Absolute shrinkage and Selection Operator.

We u a technique known as L1 Regularization.

Its ary purpose is to prevent overfitting, same as Ridge regression. Here we add an extra term as $\lambda|w|$

Deriving Mathematically for 2D

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda|m|$$

$$L = \sum_{i=1}^{n} (y_i - mx_i - b)^2 + \lambda|m|$$

$b = \bar{y} - m\bar{x}$

where

$\bar{y}$ → y mean

$\bar{x}$ → x mean

m → slope

$$L = \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})^2 + 2\lambda|m|$$

Adding 2 just for mathematically convenience

Since we have $|m|$, we cannot differentiate modulus. Therefore we will break it into 3 cases.

Case (i) $m > 0$ $(\because |m| \to m)$

$$L = \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})^2 + 2\lambda m$$

$$\frac{dL}{dm} = 2 \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})(-x_i + \bar{x}) + 2\lambda = 0$$

[rearranging

$$= -2 \sum_{i=1}^{n} [(y_i - \bar{y}) - m(x_i - \bar{x})](x_i - \bar{x}) + 2\lambda = 0$$

$$= -\sum_{i=1}^{n} [(y_i - \bar{y})(x_i - \bar{x}) - m(x_i - \bar{x})^2] + \lambda = 0$$

$$= -\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) + m \sum_{i=1}^{n} (x_i - \bar{x})^2 + \lambda = 0$$

$$m \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) - \lambda$$

$$m = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) - \lambda}{\sum (x_i - \bar{x})^2}$$

All 3 cases

Case ① $m > 0$

$$m = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x}) - \lambda}{\sum (x_i-\bar{x})^2}$$

Case ② $m = 0$ (Simple Linear Reg.)

$$m = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x})}{\sum (x_i-\bar{x})^2}$$

Case ③ $m < 0$

$$m = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x}) + \lambda}{\sum (x_i-\bar{x})^2}$$

Imp Question:

Why does Lasso Regression has sparsity ? (coefficients value can be 0)

$$m = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x}) - \lambda}{\sum (x_i-\bar{x})^2}$$

Here, let's simplify the terms. $(y_i-\bar{y})(x_i-\bar{x}) = yx$ and $(x_i-\bar{x})^2 = x^2$

$\therefore m = \frac{yx - \lambda}{x^2}$, Now, assume value of $yx = 100$ and $x^2 = 50$

We will measure the behaviour of m as we increase the value of λ.

$m = \frac{100-0}{50} = 2$ $\quad \lambda = 0$

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

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

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

$m = \frac{100-150}{50} = -1$ $\quad \lambda = 150$

We can notice, as we keep on increasing λ, the slope decreases and eventually 0.

But note that if you take $\lambda > 100$, the value of m will be < 0, but this is not possible as when $m < 0$, we use the ③ case formula. and vice versa.

↳ When $m < 0$, we will use ③

$$m = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x}) + \lambda}{\sum (x_i-\bar{x})^2}$$

In Ridge regression,

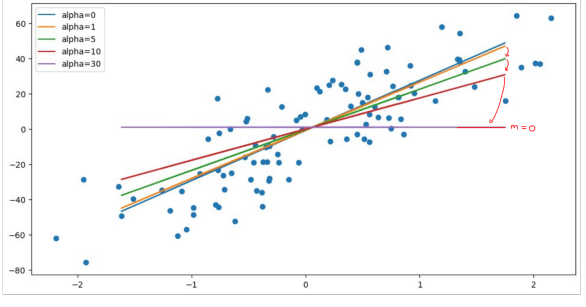
$$m = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x})}{\sum (x_i-\bar{x})^2 + \lambda}$$

because λ is in the denominator, It is impossible for value of m to be 0. Therefore, m in ridge never goes to 0, So, there's no sparsity

The main difference between Ridge and Lasso is that in Ridge, even if you max the alpha, coefficient never goes 0. Therefore, the value of coefficients will always be something (>0). But in lasso, the value of when increased, after certain threshold, the value of coefficients(m) will be zero. So it means that the slope has no effect on the overall model.

$$y = \underbrace{m}_{m=0} x + b$$

This will indirectly work as feature selection.



We can notice, on increasing the value of alpha, the slope decreases and after certain alpha, the slope is flat, resulting in underfitting.

Elastic Net

ElasticNet is just a combination Ridge + Lasso Regression

You apply elastic net regression when you have alot of input cols and are not aware of details for all the input cols.

If you are confused on either to apply Ridge or lasso, you apply this.

$$L = \Sigma(y_i - \bar{y}_i)^2 + a\|w\|^2 + b\|w\|$$

What is ElasticNet?

ElasticNet is a regularized regression method that combines Ridge (L2) and Lasso (L1) penalties. It addresses limitations of both methods by providing variable selection while handling correlated features effectively.

$$\text{Loss} = \text{MSE} + \alpha_1||\beta||_1 + \alpha_2||\beta||_2^2$$

Key Parameters:

- **α (alpha):** Overall regularization strength
- **l1_ratio:** Balance between L1 and L2 (0 to 1)
- l1_ratio = 0 → Ridge regression
- l1_ratio = 1 → Lasso regression

Mathematical Foundation

$$\hat{\beta} = \text{argmin}\{||y - X\beta||^2 + \alpha[(1-\rho)||\beta||_2^2 + \rho||\beta||_1]\}$$

Where:

- ρ = l1_ratio (mixing parameter)
- α = regularization parameter
- $||\beta||_1 = \Sigma|\beta_i|$ (L1 norm - Lasso penalty)
- $||\beta||_2^2 = \Sigma\beta_i^2$ (L2 norm - Ridge penalty)

Python Implementation

```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import
GridSearchCV # Basic usage elastic =
ElasticNet(alpha=1.0, l1_ratio=0.5)
elastic.fit(X_train, y_train) predictions =
elastic.predict(X_test) # Hyperparameter
tuning param_grid = { 'alpha': [0.1, 1.0,
10.0], 'l1_ratio': [0.1, 0.5, 0.7, 0.9] }
grid_search = GridSearchCV(ElasticNet(),
param_grid, cv=5) grid_search.fit(X_train,
y_train)
```

When to Use ElasticNet

Ideal Scenarios:

- High-dimensional data (p > n)
- Correlated features present
- Need both feature selection and regularization
- Sparse solutions desired
- Ridge and Lasso individually insufficient

Applications:

- Genomics and bioinformatics
- Text mining and NLP
- Finance and economics
- Image processing

Comparison with Other Methods

Method	Penalty	Feature Selection	Handles Correlation	Sparsity
Linear Regression	None	No	Poor	No
Ridge	L2	No	Good	No
Lasso	L1	Yes	Poor	Yes
ElasticNet	**L1 + L2**	**Yes**	**Good**	**Yes**

✓ Advantages

- Combines benefits of Ridge and Lasso
- Handles multicollinearity effectively
- Performs automatic feature selection
- Works well with grouped variables
- Stable selection of correlated features
- Prevents overfitting in high dimensions
- Computationally efficient

✗ Disadvantages

- Two hyperparameters to tune
- More complex than Ridge/Lasso alone
- Still assumes linear relationships
- May not capture complex interactions
- Sensitive to feature scaling
- Interpretability reduced vs. Lasso

Hyperparameter Tuning Tips

Alpha (α) Selection:

- Start with range [0.01, 0.1, 1, 10, 100]
- Higher α = more regularization
- Use cross-validation for optimal value

L1_ratio Selection:

- 0.1-0.3: More Ridge-like (grouped selection)
- 0.5: Balanced L1/L2
- 0.7-0.9: More Lasso-like (sparse solutions)

Best Practices

- **Preprocessing:** Always standardize features
- **Cross-validation:** Use 5-10 fold CV
- **Grid Search:** Start coarse, then refine
- **Evaluation:** Use appropriate metrics (RMSE, MAE)
- **Feature Engineering:** Create polynomial/interaction terms
- **Validation:** Hold-out test set for final evaluation

```
# Complete pipeline example from
sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
pipeline = Pipeline([ ('scaler',
StandardScaler()), ('elastic', ElasticNet())
])
```

Key Takeaway: ElasticNet is the go-to choice when you need both regularization and feature selection, especially with correlated predictors. It's particularly powerful in high-dimensional settings where traditional methods fail.