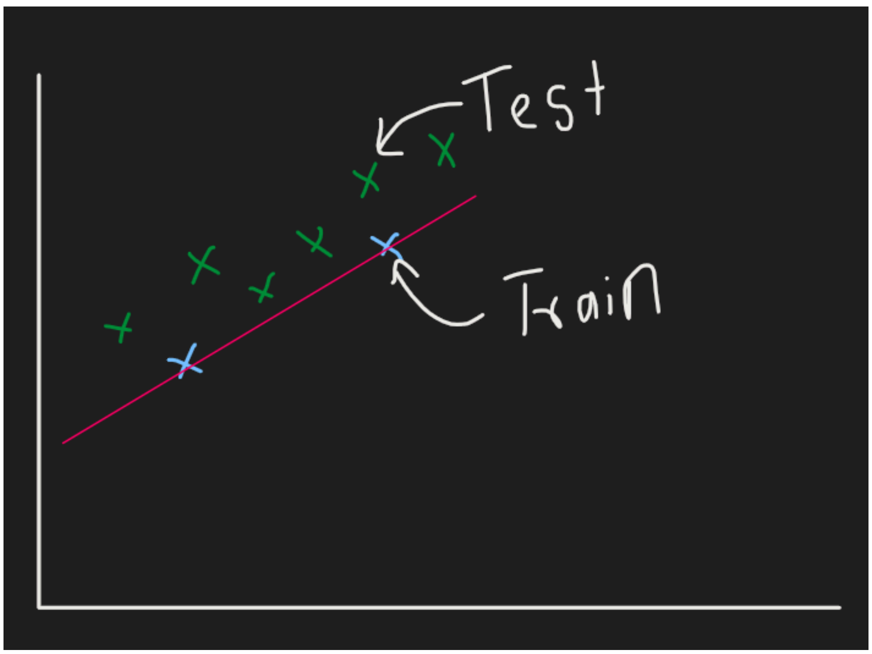
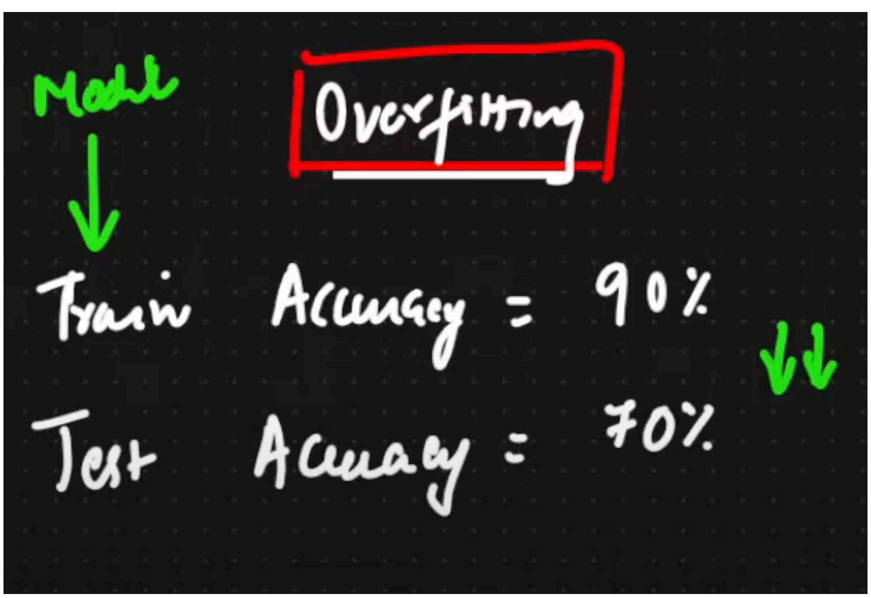
Ridge & Lasso Regression

- Prevents overfitting
- **Regularization** introduces a penalty term to the loss function during model training. This penalty discourages large coefficients and helps produce simpler, more generalizable models.





Overfitting👆

Ridge Regression

- **Ridge (L2 Regularization)**:
  - Adds a penalty term equal to the **sum of squared coefficients**
  - **No Feature Selection**: Shrinks coefficients but rarely sets them to zero.

Objective Function

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

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$: Ordinary Least Squares (OLS) loss (sum of squared residuals).
- $\lambda \sum_{j=1}^{p} \beta_j^2$: L2 penalty term (shrinks coefficients toward zero).

- Known as **L2 Regularization because you multiply by square**

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

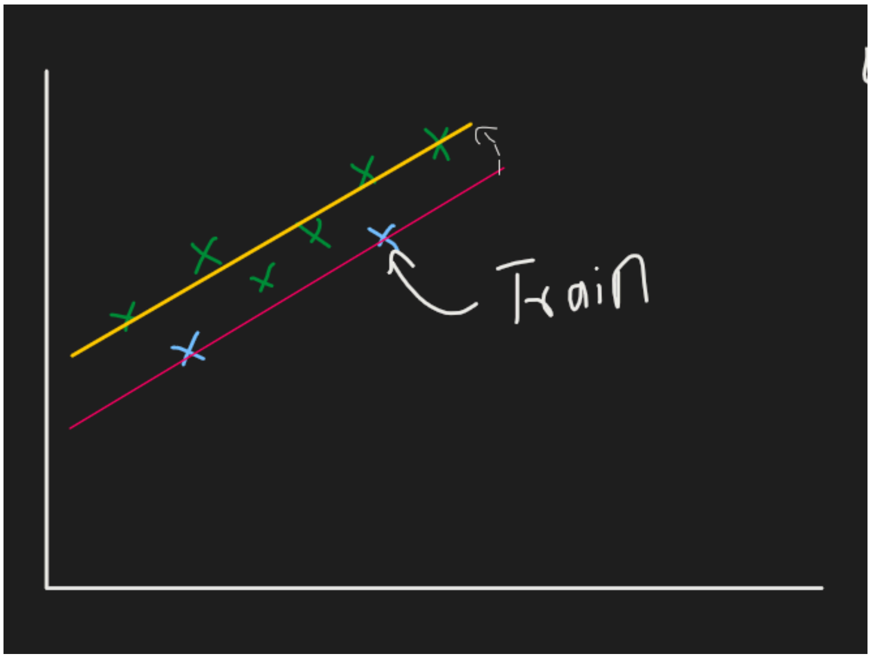
$$+ \lambda m^2$$

$0 - \infty$

$y = mx + b$

- In overfitted models → m is high
- **We have to reduce** m

- To do this, you add λm^2
- This is hyperparameter
- You can tune its value.



Code:

```
from sklearn.datasets import load_diabetes

data=load_diabetes()
```

```
print(data.DESCR)
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline
```

```
X=data.data
y=data.target
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=45)
```

```
from sklearn.linear_model import LinearRegression
L=LinearRegression()

L.fit(X_train,y_train)

print(L.coef_)
print(L.intercept_)

Output:
[  23.45465406 -247.42747406  492.1087518   329.35876431 -970.79723039
  573.54295519  182.42162368  255.92168168  794.21609282   89.32249214]
152.13623331746496
```

```
y_pred=L.predict(X_test)

from sklearn.metrics import r2_score,mean_squared_error

print("R2 score",r2_score(y_test,y_pred))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))

Output:
R2 score 0.5188113124539249
RMSE 48.72713760953253
```

Now do the same with Ridge Regression:

```
from sklearn.linear_model import Ridge
R=Ridge(alpha=0.0001)

R.fit(X_train,y_train)
```

```
print(R.coef_)
print(R.intercept_)

Output:
[  23.51763492 -247.31766656  492.28244914  329.3317593  -957.46324421
  562.90310325  176.71070198  254.47033329  789.10867561   89.41375823]
152.13492030963658
```

```
y_pred1=R.predict(X_test)

print("R2 score",r2_score(y_test,y_pred1))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred1)))

Output:
```

```
R2 score 0.518973263588495
RMSE 48.718937001819555
```

Ridge Regression with Gradient Descent

```
reg = SGDRegressor(penalty='l2',max_iter=500,eta0=0.1,learning_rate='constant',alpha=0.001)
```

`penalty='l2'` → **Ridge (**L1 is Lasso**)**

`eta0=0.1` → Learning rate

`alpha=0.001` → **λ in Ridge Regression**

```
reg.fit(X_train,y_train)

y_pred= reg.predict(X_test)
print("R2 score",r2_score(y_test,y_pred))
print(reg.coef_)
print(reg.intercept_)
```

Output:

R2 score 0.4917350255359758
[40.95027982 -125.19406163 378.55185529 255.30708968 -25.12973185
-69.4432912 -183.4794615   131.29527455 322.24438019 137.46469942]
[145.97997383]

GridSearchCV

`GridSearchCV` is used to **find the optimal hyperparameter** (e.g., `alpha` for Ridge regression) by testing all combinations in the specified parameter grid (`alpha: [1, 2, 5, ..., 90]`) and selecting the best one using cross-validation (CV).

- **`GridSearchCV`** combines CV with **hyperparameter search**:
    1. Tests all `alpha` values.

2. Uses 5-fold CV to evaluate each `alpha`.
3. Selects the `alpha` with the best average validation score.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Step 1: Define model and parameter grid
ridge_regressor = Ridge()
parameters = {'alpha': [1, 2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90]}

# Step 2: GridSearchCV tests all alphas with 5-fold CV
ridgecv = GridSearchCV(
    ridge_regressor,
    parameters,
    scoring='r2',
    cv=5  # 5-fold cross-validation
)
ridgecv.fit(X_train, y_train)

# Step 3: Best alpha and model
print("Best alpha:", ridgecv.best_params_['alpha'])  # e.g., alpha=10
print("Best MSE:", -ridgecv.best_score_)  # Convert back to positive MSE
```

Output:

{'alpha': 20}
best ridge score: 0.6917447889048314

Predict y:

```
ridge_pred=ridgecv.predict(X_test)
```

Test all values from 1 to 50

```
alphas = np.arange(1, 51)

# Initialize the Ridge regressor
ridge_regressor = Ridge()

# Set up the parameter grid
parameters = {'alpha': alphas}
```

```
ridgecv = GridSearchCV(ridge_regressor, parameters, scoring='r2', cv=5)

# Fit the model to the training data
ridgecv.fit(X_train, y_train)

# Retrieve the best alpha value
best_alpha = ridgecv.best_params_['alpha']
print(f"The best alpha value is: {best_alpha}")
```

Output:

The best alpha value is: 20

Lasso Regression (**Least Absolute Shrinkage and Selection Operator**)

- **L1 regularization**
- It adds a **penalty** to the absolute values of coefficients, which can **shrink some coefficients to zero**, effectively selecting important features and removing irrelevant ones.
- Prevents Overfitting

Formula:

$$\min \sum (y_i - \hat{y}_i)^2 + \lambda \sum |\beta_j|$$

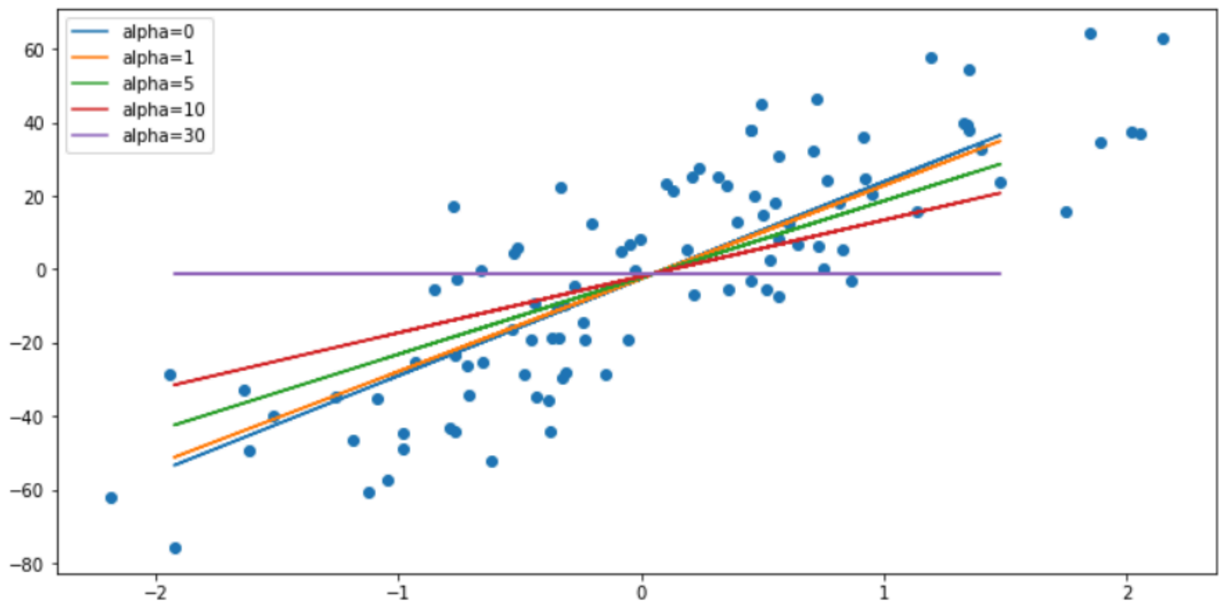
- If **λ = 0**, Lasso acts as normal Linear Regression (no penalty).
- If **λ is high**, many coefficients shrink to zero → feature selection happens.

Why Use Lasso?

- **Feature Selection**: Automatically removes irrelevant features by setting their coefficients to zero.
- **Handles High-Dimensional Data**: Effective when the number of features (*p*) exceeds the number of samples (n).
- **Reduces Overfitting**: Penalizes complex models to improve generalization.

When to Use Lasso?

- Many features, but only a subset are relevant.
- Need a simpler, interpretable model.
- Suspect multicollinearity but want feature selection.



Comparison with Ridge and Elastic Net

Method	Regularization	Feature Selection	Use Case
Lasso	L1 (absolute)	Yes	Sparse models, feature selection.
Ridge	L2 (squared)	No	Stabilize coefficients, multicollinearity.
Elastic Net	L1 + L2	Yes	Correlated features + sparsity.

Python Implementation of Lasso Regression

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

# Generate Sample Data
np.random.seed(42)
X = np.random.rand(100, 5)  # 100 samples, 5 features
```

```
y = 3*X[:,0] + 2*X[:,1] + np.random.randn(100)  # True relationship

# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Apply Lasso Regression
lasso = Lasso(alpha=0.1)  # λ = 0.1
lasso.fit(X_train, y_train)

# Predictions
y_pred = lasso.predict(X_test)

# Evaluate Model
mse = mean_squared_error(y_test, y_pred)
print("Lasso MSE:", mse)

# Display Coefficients
print("Lasso Coefficients:", lasso.coef_)
```

Output:

Lasso MSE: 0.9218118933165386
Lasso Coefficients: [1.36645736 0.58798862 0.   -0.   -0.   ]

Choosing the Best Alpha (Hyperparameter Tuning with Cross-Validation)

```
from sklearn.linear_model import LassoCV

# Automatically finds the best alpha using cross-validation
lasso_cv = LassoCV(alphas=np.logspace(-4, 1, 50), cv=5)
lasso_cv.fit(X_train, y_train)

# Best alpha
print("Best Alpha:", lasso_cv.alpha_)
```

```
Output:
Best Alpha: 0.017575106248547933
```

When to Use Lasso?

- When feature selection is needed.
- When many features are irrelevant (sparse models).
- When avoiding multicollinearity (reduces highly correlated features).

✅ **Use Lasso when interpretability matters** (simplifies models).

❌ **Don't use Lasso when all features are important** (use Ridge instead).

GridSearchCV

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

```
lasso=Lasso()
parameters={'alpha':[1,2,5,10,20,30,40,50,60,70,80,90]}
lassocv=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error',
cv=5)
lassocv.fit(X_train,y_train)
```

```
print(lassocv.best_params_)
print('best score: ',lassocv.best_score_)

Output:
{'alpha': 1}
best score:  -31.153603752119004
```

Now predict the y:

```
lasso_pred=lassocv.predict(X_test)
```

Key Points (Ridge & Lasso) / Intuition:

- As you increase **the value of Lambda (λ)**, the coefficients get close to **zero**.
- As you **decrease the value of Lambda (λ)**, the coefficients get close to zero.
    - Bias will decrease (**Model will overfit**)
    - Variance will increase
- λ=0 → Simple linear regression
- Big coefficients reduce more compared to small ones.

Lasso vs Ridge Regression (L1 vs L2 Regularization)

Feature	**Lasso (L1 Regularization)**	**Ridge (L2 Regularization)**
Feature Selection	**Yes** – Shrinks some coefficients to **exact zero**, removing features	**No** – Shrinks coefficients but **keeps all features**
Best For	When **some features are irrelevant**, Lasso will drop them	When **all features are useful**, Ridge will just reduce their impact
Effect on Multicollinearity	Selects **one feature** among correlated ones, others become zero	Distributes weight across correlated features
Model Complexity	Simpler (fewer features remain)	More complex (all features remain)
Computational Cost	Higher (requires optimization for sparsity)	Lower (simpler gradient descent)