**SUPERVISED MACHINE LEARNING: REGRESSION**

**MODULE 3:**

**CROSS VALIDATION**

**TABLE OF CONTENTS**

[**Cross-Validation Concepts 2**](#_ocnkq1ars1vv)

[K-Fold Cross-Validation 2](#_962tisuuqovs)

[Types of K-Fold Cross-Validation 2](#_jlckkk4uydo)

[Standard K-Fold 2](#_q1gkzkzk1zw)

[Stratified K-Fold 3](#_lw9su58fg6bd)

[Group K-Fold 3](#_stfdwspavusm)

[Model Complexity and Choice of K 4](#_n94gzpryu10k)

[**Regularized Linear Models 4**](#_4xf8ge4uro5v)

[Ridge Regression (L2 Regularization) 5](#_18os23dovpb1)

[Lasso Regression (L1 Regularization) 5](#_12pin8o98wrd)

[Elastic Net (Combination of L1 and L2) 6](#_4rwssoudfozf)

[Selecting the Best Alpha with Cross-Validation 6](#_xmkp738mr695)

[Bias–Variance Trade-Off 8](#_n0pmojrxnw2z)

[**Key Takeaways 8**](#_l1xybcs5szsi)

# 

# 

# Cross-Validation Concepts

* Cross-validation is a method to **assess how well a model generalizes** to unseen data.
* Instead of one train-test split, the dataset is divided into multiple parts (*folds*).
* The model is trained on some folds and validated on the remaining ones, and this process is repeated to obtain an **average performance metric** that reduces random bias.

## K-Fold Cross-Validation

* In **K-Fold Cross-Validation**, the dataset is split into *k* folds:
  + Train the model on *k – 1* folds and validate it on the remaining fold.
  + Repeat *k* times so each fold serves once as validation.
  + The final score is the **average across all folds**.

## Types of K-Fold Cross-Validation

### **Standard K-Fold**

* Randomly splits data into *k* folds of equal size.
* Works well for balanced, independent datasets.

| from sklearn.model\_selection import KFold, cross\_val\_score  from sklearn.linear\_model import Ridge  # Define model  ridge = Ridge(alpha=1.0)  # Standard 10-fold CV  kf = KFold(n\_splits=10, shuffle=True, random\_state=1)  scores = cross\_val\_score(ridge, X, y, cv=kf, scoring='neg\_mean\_squared\_error')  print("MSE for each fold:", -scores)  print("Average MSE:", -np.mean(scores)) |
| --- |

### Stratified K-Fold

* Maintains the same class distribution in each fold.
* Suitable for **imbalanced classification problems**.

| from sklearn.model\_selection import StratifiedKFold, cross\_val\_score  from sklearn.linear\_model import LogisticRegression  from sklearn.datasets import load\_breast\_cancer  # Load data  data = load\_breast\_cancer()  X, y = data.data, data.target  # Define model  model = LogisticRegression(max\_iter=5000)  # Stratified 5-fold CV  skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)  scores = cross\_val\_score(model, X, y, cv=skf, scoring='accuracy')  print("Accuracy per fold:", scores)  print("Average accuracy:", scores.mean()) |
| --- |

# 

### Group K-Fold

* Keeps **related samples** (e.g., from the same group or person) in the same fold.
* Prevents **data leakage** between training and validation.

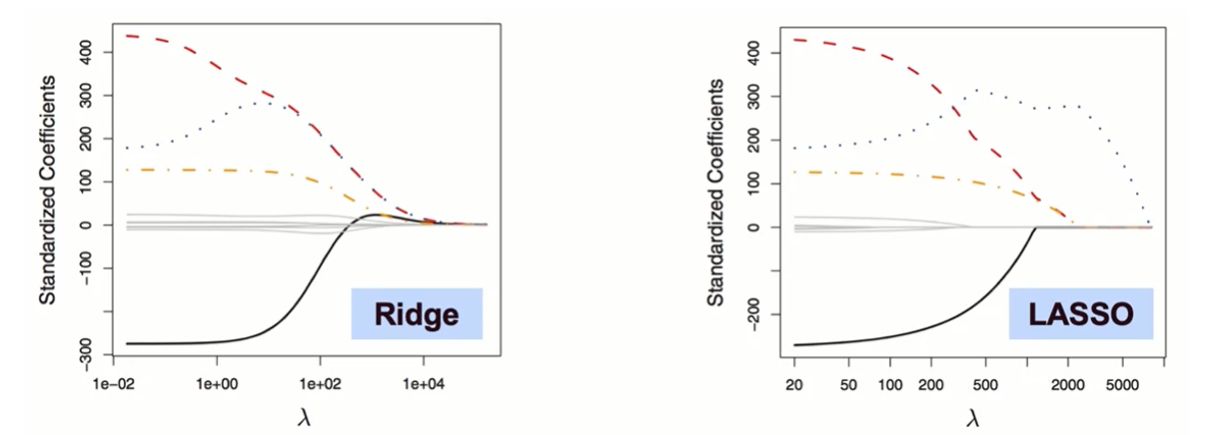
| from sklearn.model\_selection import GroupKFold  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error  import numpy as np  # Create dummy groups  groups=np.array([i//10 for i in range(100)])# 10 samples per group  # Define model and Group K-Fold  model = LinearRegression()  gkf = GroupKFold(n\_splits=5)  for fold, (train\_idx, val\_idx) in enumerate(gkf.split(X, y, groups)):  model.fit(X[train\_idx], y[train\_idx]) # Train on groups  preds = model.predict(X[val\_idx]) # Validate on unseen groups  print(f"Fold {fold+1} MSE:", mean\_squared\_error(y[val\_idx], preds)) |
| --- |

## Model Complexity and Choice of K

| **Model Type** | **Description** | **Common Issue** |
| --- | --- | --- |
| **Low Complexity** | Simple, few parameters | Underfitting |
| **High Complexity** | Many features, very flexible | Overfitting |

* **Smaller k (e.g., 2–3):** higher variance, less stable results.
* **Larger k (e.g., 10–20):** lower variance, more stable but computationally expensive.

# Regularized Linear Models

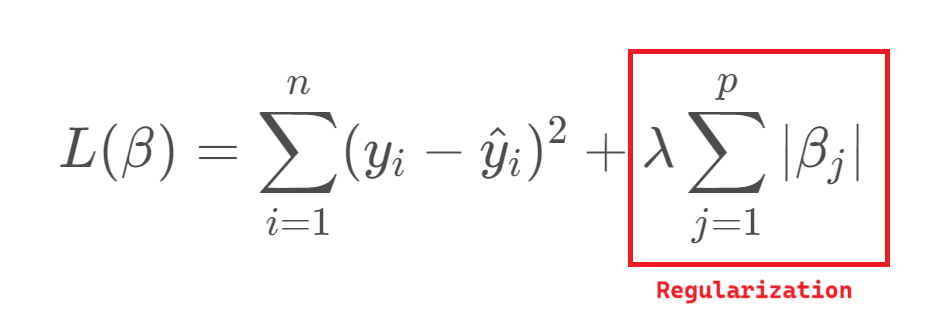


Regularization reduces overfitting by adding a **penalty term** to the cost function, keeping coefficients small and improving generalization.

| ***Minimize (Loss+Penalty)*** |
| --- |

## Ridge Regression (L2 Regularization)

**Adds a squared penalty:**

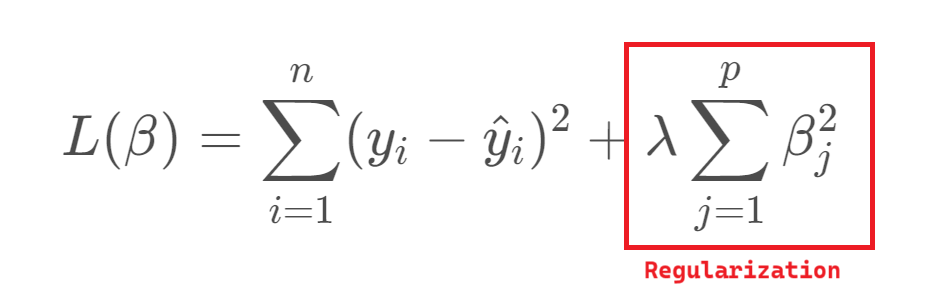
****

* Shrinks all coefficients but keeps them nonzero.
* Useful when all predictors contribute slightly.

| from sklearn.linear\_model import Ridge  from sklearn.metrics import mean\_squared\_error  ridge = Ridge(alpha=1.0)  ridge.fit(X, y) # Train model  print("Ridge Coefficients:", ridge.coef\_) |
| --- |

## Lasso Regression (L1 Regularization)

**Adds an absolute penalty:**



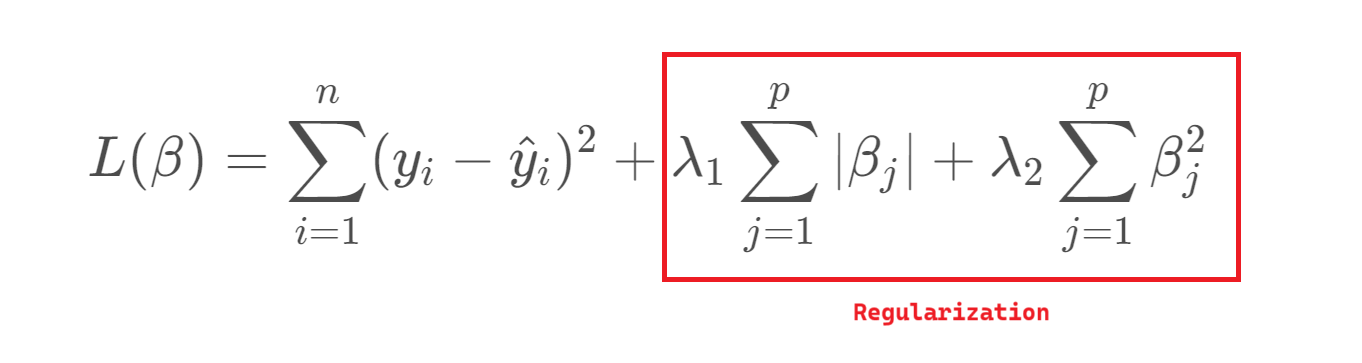
* Some coefficients become exactly zero → **automatic feature selection**.
* Best when only a few features are important.

| from sklearn.linear\_model import Lasso  lasso = Lasso(alpha=0.05)  lasso.fit(X, y)  print("Lasso Coefficients:", lasso.coef\_) |
| --- |

## 

## Elastic Net (Combination of L1 and L2)

**Combines Ridge and Lasso penalties:**

****

* Balances Lasso’s sparsity and Ridge’s stability.
* Works well for correlated features.

| from sklearn.linear\_model import ElasticNet  elastic = ElasticNet(alpha=0.5, l1\_ratio=0.5)  elastic.fit(X, y)  print("Elastic Net Coefficients:", elastic.coef\_) |
| --- |

## 

## Selecting the Best Alpha with Cross-Validation

* Scikit-learn provides built-in models that automatically choose the best alpha (regularization strength) using K-Fold Cross-Validation.
  + **RidgeCV**

| from sklearn.linear\_model import RidgeCV  # Define candidate alpha values  alphas = [0.01, 0.1, 1.0, 10.0]  # Automatically select best alpha via 5-fold CV  ridge\_cv = RidgeCV(alphas=alphas, cv=5)  ridge\_cv.fit(X, y)  print("Best alpha (RidgeCV):", ridge\_cv.alpha\_)  print("R² Score:", ridge\_cv.score(X, y)) |
| --- |

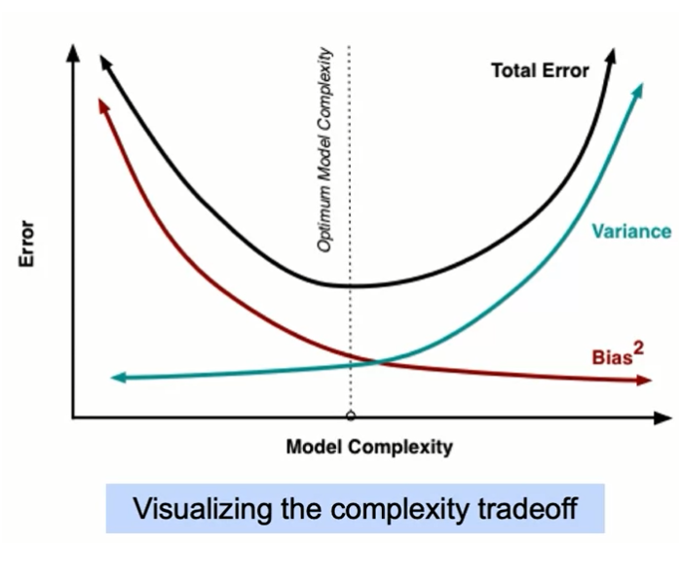
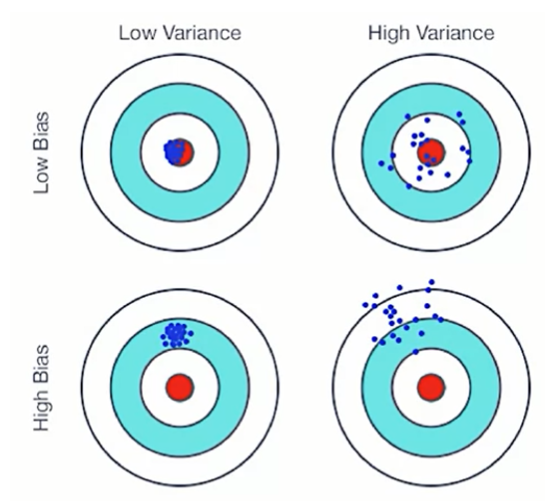
* + **LassoCV**

| from sklearn.linear\_model import LassoCV  # Automatically tunes alpha using cross-validation  lasso\_cv = LassoCV(alphas=[0.001, 0.01, 0.1, 1.0], cv=5, random\_state=42)  lasso\_cv.fit(X, y)  print("Best alpha (LassoCV):", lasso\_cv.alpha\_)  print("Selected Coefficients:", lasso\_cv.coef\_)  print("R² Score:", lasso\_cv.score(X, y)) |
| --- |

* + **ElsticNetCV**

| from sklearn.linear\_model import ElasticNetCV  # Elastic Net tunes both alpha and l1\_ratio  elastic\_cv = ElasticNetCV(  alphas=[0.001, 0.01, 0.1, 1.0],  l1\_ratio=[0.2, 0.5, 0.8],  cv=5,  random\_state=42  )  elastic\_cv.fit(X, y)  print("Best alpha (ElasticNetCV):", elastic\_cv.alpha\_)  print("Best l1\_ratio:", elastic\_cv.l1\_ratio\_)  print("R² Score:", elastic\_cv.score(X, y)) |
| --- |

## Bias–Variance Trade-Off



* Increasing regularization (larger alpha):
  + Increases **bias**
  + Decreases **variance**
* Too little regularization → overfitting
* Too much regularization → underfitting
* The best model strikes a balance between the two.

# Key Takeaways

* **Cross-Validation** Evaluates model stability by rotating train-test splits for fair performance estimation.
* **K-Fold Variants**
  + *Standard:* Random splits.
  + *Stratified:* Keeps class proportions.
  + *Group:* Prevents leakage across related samples.
* **Model Complexity** K-Fold helps detect underfitting (too simple) or overfitting (too complex).
* **Regularization** Adds penalties to limit model flexibility and reduce overfitting.
* **Ridge (L2):** Shrinks all coefficients smoothly.
* **Lasso (L1):** Performs feature selection.
* **Elastic Net:** Balances sparsity and stability.
* **Cross-Validation + Regularization:** Use CV (RidgeCV, LassoCV, ElasticNetCV) to pick the best alpha for optimal bias–variance balance.
* **Core Insight:** Combining **Cross-Validation** and **Regularization** produces models that are **accurate, interpretable, and generalizable**.