SUPERVISED MACHINE LEARNING: REGRESSION

MODULE 3:

CROSS VALIDATION

TABLE OF CONTENTS

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.

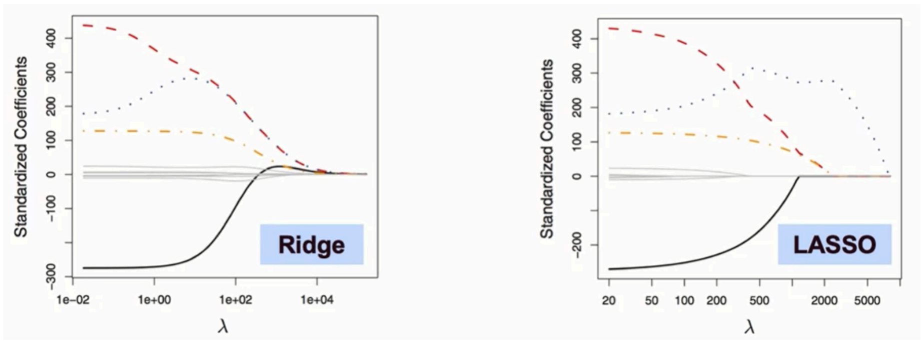
```
from sklearn.model_selection import GroupKFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# 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:

$$L(\beta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \underbrace{\boxed{\lambda \sum_{j=1}^{p} |\beta_j|}}_{\text{Regularization}}$$

- 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:

$$L(\beta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \underbrace{\boxed{\lambda \sum_{j=1}^{p} \beta_j^2}}_{\text{Regularization}}$$

- 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:

$$L(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \underbrace{\boxed{\lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2}}_{\text{Regularization}}$$

- 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**

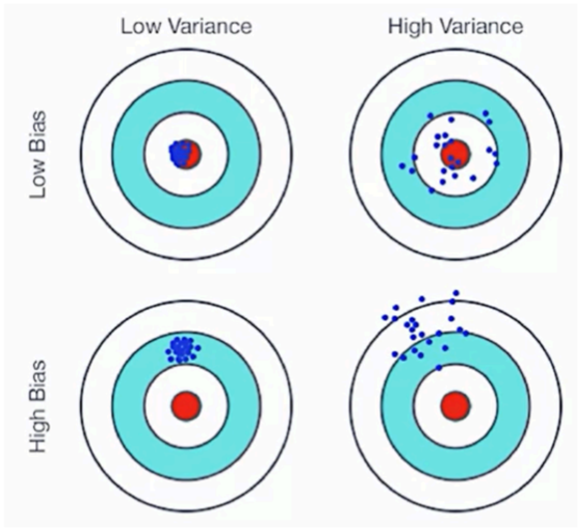
```
from sklearn.linear_model import LassoCV

# 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))
```

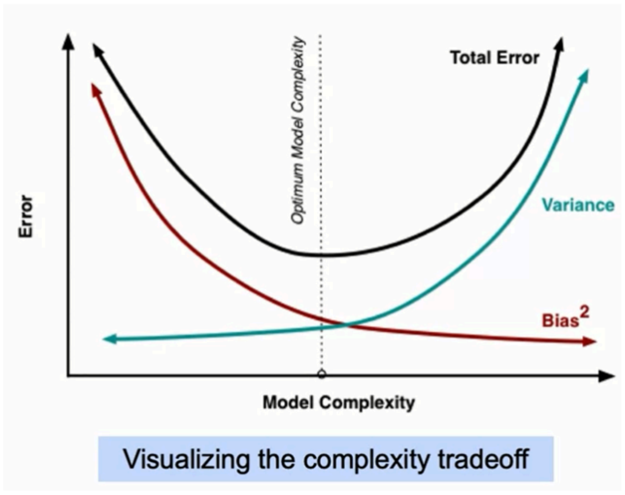
- **ElsticNetCV**

```
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**.