


# MOOC 2: Supervised Machine Learning: Regression

 **View Original Notion Document:** [https://www.notion.so/MOOC-2-Supervised-Machine-Learning-Regression-2802d55efe0c8068b9ecd6a40573fff3?source=copy\\_link](https://www.notion.so/MOOC-2-Supervised-Machine-Learning-Regression-2802d55efe0c8068b9ecd6a40573fff3?source=copy_link)

---

## Module 1: Introduction to Supervised Machine Learning and Linear Regression

### Giới thiệu về Supervised Machine Learning



**Supervised Learning** là phương pháp học máy sử dụng dữ liệu được gán nhãn (labeled data) để huấn luyện model dự đoán.

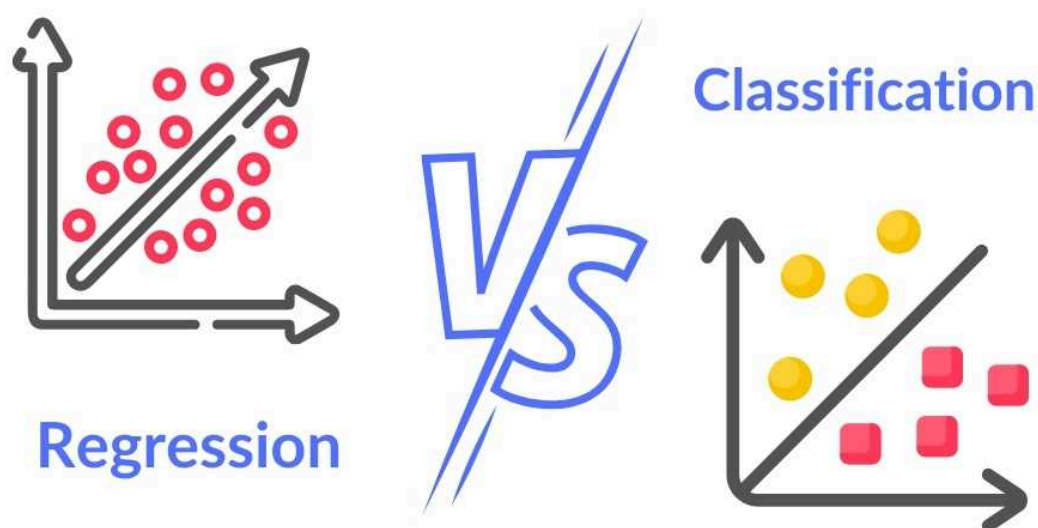
### Hai loại chính của Supervised Learning:

#### 1. Regression (Hồi quy)

- Target variable (biến mục tiêu) là **liên tục** (continuous)
- Ví dụ: Dự đoán giá nhà, nhiệt độ, doanh thu

#### 2. Classification (Phân loại)

- Target variable là **phân loại** (categorical)
- Ví dụ: Phân loại email spam/không spam, nhận diện hình ảnh



### Yêu cầu để xây dựng Classification Model:

- **Features** có thể định lượng được (quantifiable features)
- **Labeled target** hoặc outcome variable
- **Phương pháp** để đo lường độ tương đồng (similarity)

## Linear Regression (Hồi quy tuyến tính)



**Linear Regression** mô hình hóa mối quan hệ giữa một biến liên tục (dependent variable) và một hoặc nhiều biến độc lập (independent variables).

### Công thức toán học:

#### Simple Linear Regression:

$$y = \beta_0 + \beta_1 x + \epsilon$$

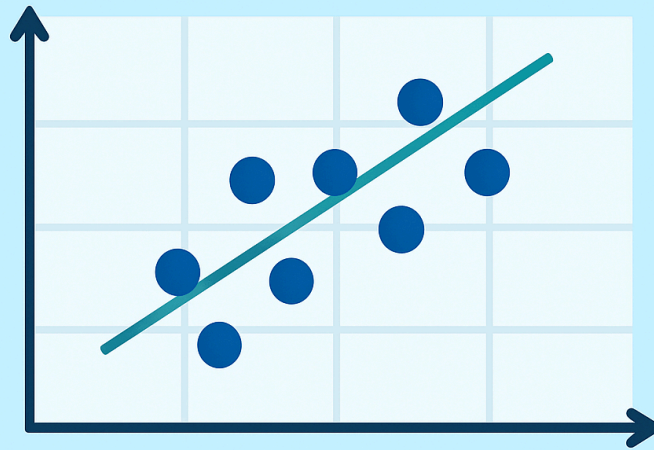
#### Multiple Linear Regression:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

Trong đó:

- $y$  = biến phụ thuộc (dependent variable)
- $\beta_0$  = hệ số chặn (intercept)
- $\beta_1, \beta_2, \dots, \beta_n$  = hệ số hồi quy (coefficients)
- $x_1, x_2, \dots, x_n$  = biến độc lập (independent variables)
- $\epsilon$  = sai số (error term)

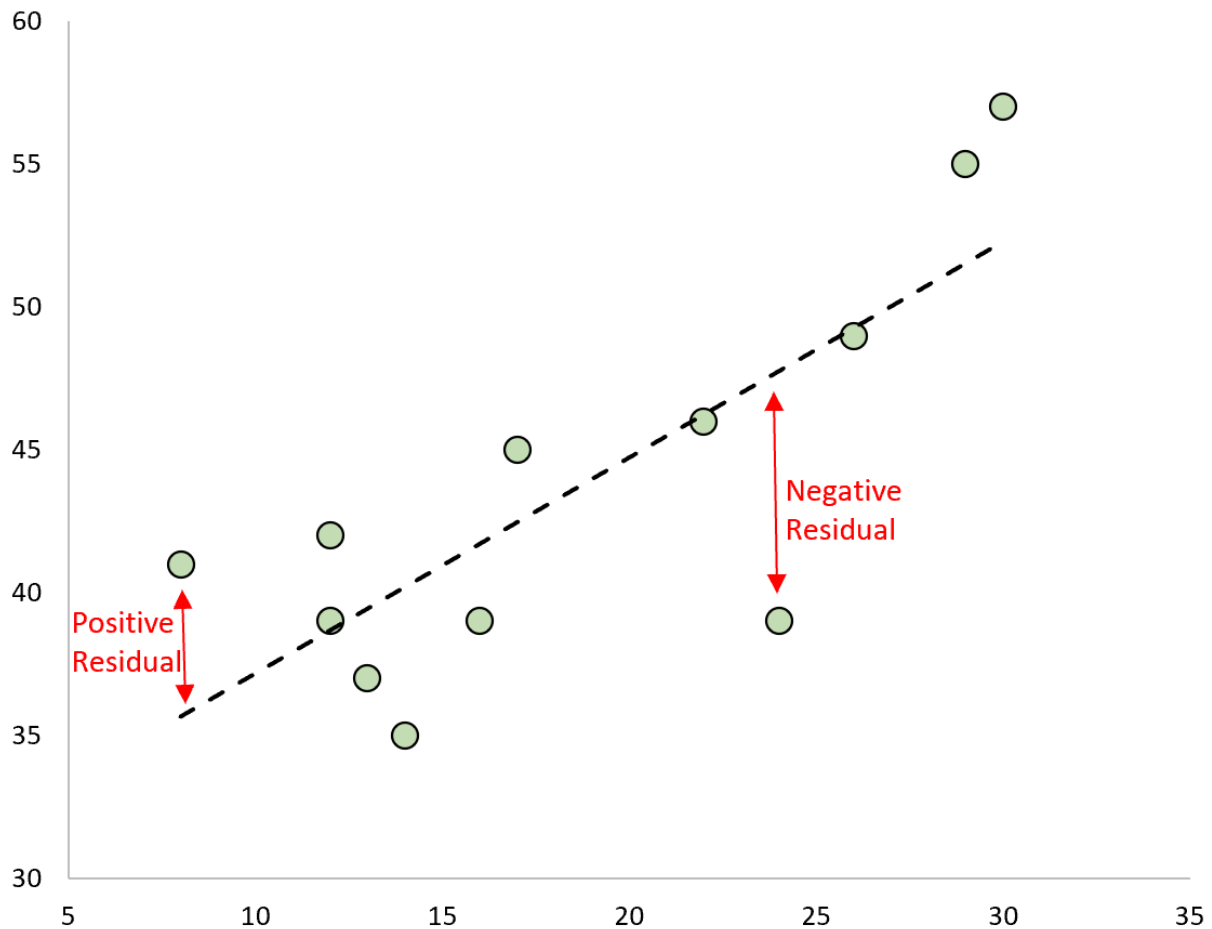
## WHAT IS LINEAR REGRESSION



### Residuals (Phần dư)

**Residual** được định nghĩa là **hiệu số giữa giá trị thực tế và giá trị dự đoán**

$$\text{Residual}_i = y_i - \hat{y}_i$$



## Ba thước đo lỗi (Error Measures) phổ biến

### 1. Sum of Squared Errors (SSE)

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

- Tổng bình phương của các phần dư
- **Càng nhỏ càng tốt**

### 2. Total Sum of Squares (TSS)

$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2$$

- Đo lường tổng biến thiên của data

- $\bar{y}$  là giá trị trung bình

### 3. Coefficient of Determination ( $R^2$ )

$$R^2 = 1 - \frac{SSE}{TSS}$$

- Thước đo mức độ fit của model
- **$R^2$  càng gần 1 càng tốt** ( $0 \leq R^2 \leq 1$ )
- Ý nghĩa: % biến thiên được giải thích bởi model

## Code Implementation với Scikit-learn

### Cài đặt thư viện:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

### Train Linear Regression Model:

```
# Import LinearRegression
from sklearn.linear_model import LinearRegression

# Khởi tạo model
LR = LinearRegression()

# Fit model với training data
LR = LR.fit(X_train, y_train)

# Hoặc viết gọn lại:
# LR = LinearRegression().fit(X_train, y_train)
```

### Dự đoán (Prediction):

```
# Dự đoán trên test set  
y_predict = LR.predict(X_test)
```

## Module 2: Data Splits and Polynomial Regression

### Training and Test Splits



**Chia data thành training và test sets** giúp bạn chọn được model có khả năng tổng quát hóa (generalize) tốt và tránh overfitting.

#### Mục đích của mỗi set:

##### Training Data

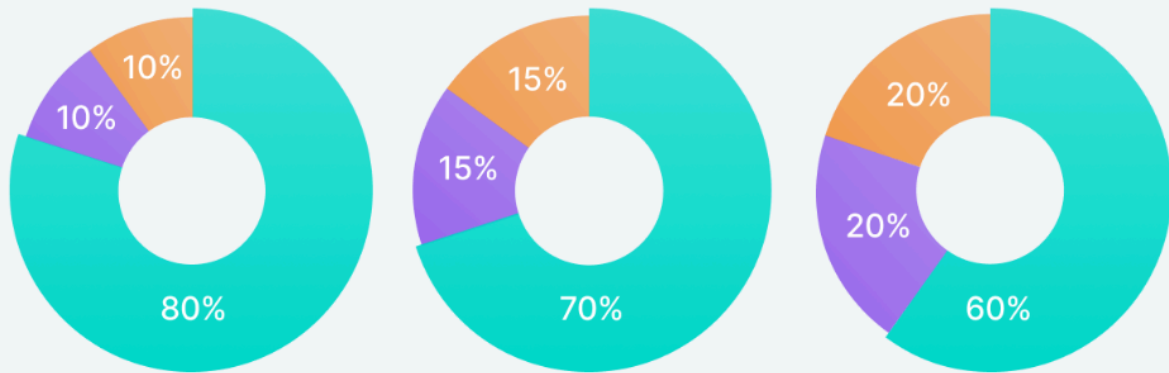
- Dùng để **fit model** (huấn luyện các parameters)
- Thường chiếm 70-80% tổng data

##### Test Data

- Dùng để **đo lường error và performance**
- Thường chiếm 20-30% tổng data
- **KHÔNG được dùng trong quá trình training!**

## Data Training Needs

● Training data      ● Validation data      ● Test data



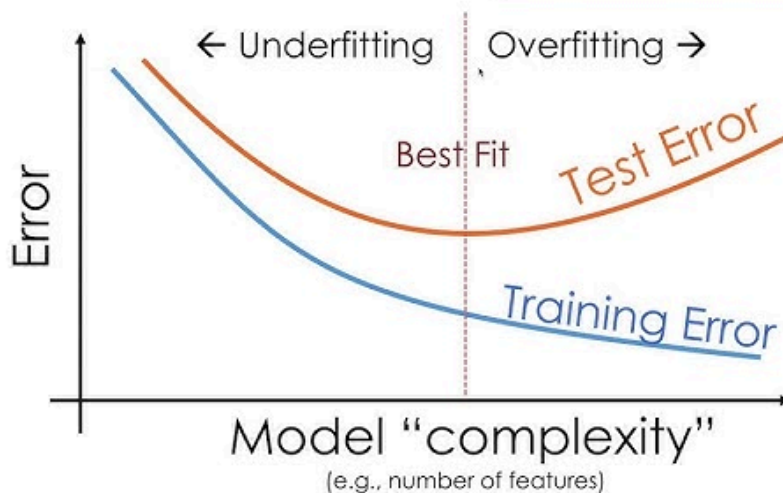
**Quan sát quan trọng:**

**Training error** có xu hướng **giảm** khi model phức tạp hơn

**Nhưng** test error có thể **tăng** → Dấu hiệu của **overfitting**!

## Training vs Test Error

Training error typically under estimates test error.



⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿

## Polynomial Regression



**Polynomial terms** giúp bạn capture được **nonlinear effects** của features trong data.

## Công thức Polynomial Regression:

**Degree 2 (Quadratic):**

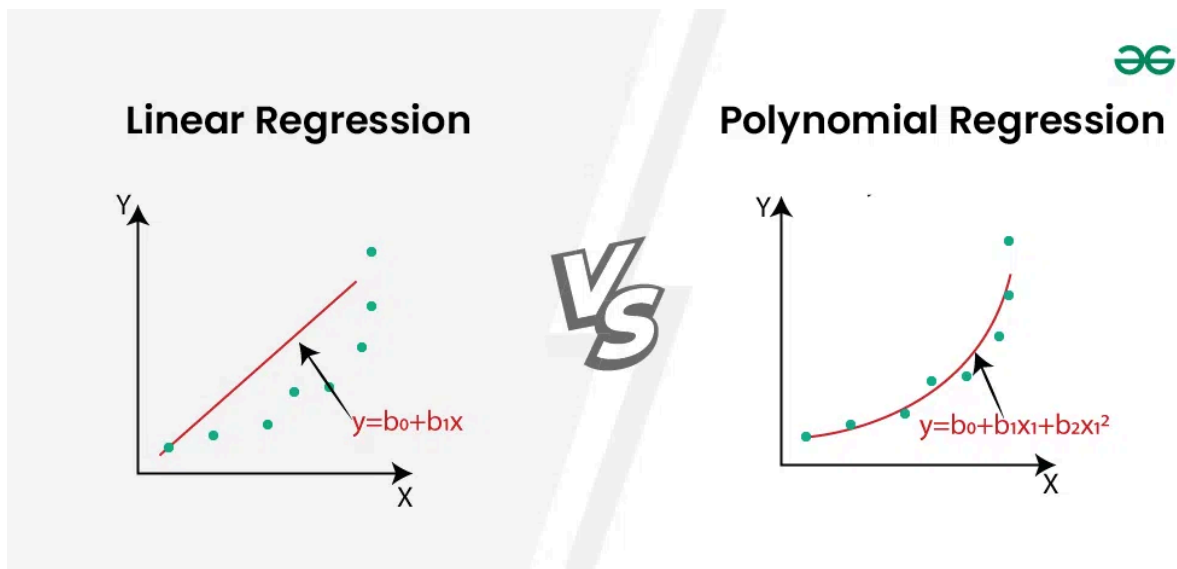
$$y = \beta_0 + \beta_1x + \beta_2x^2 + \epsilon$$

**Degree n:**

$$y = \beta_0 + \beta_1x + \beta_2x^2 + \dots + \beta_nx^n + \epsilon$$

**Ví dụ so sánh:**

| Degree        | Equation   | Use Case                |
|---------------|--|-------------------------|
| 1 (Linear)    | $y = \beta_0 + \beta_1x$                           | Mối quan hệ tuyến tính  |
| 2 (Quadratic) | $y = \beta_0 + \beta_1x + \beta_2x^2$              | Đường cong parabol      |
| 3 (Cubic)     | $y = \beta_0 + \beta_1x + \beta_2x^2 + \beta_3x^3$ | Đường cong phức tạp hơn |



## Code Implementation - Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```



```

from sklearn.pipeline import Pipeline
import numpy as np
import matplotlib.pyplot as plt

# Tạo sample data với nonlinear relationship
np.random.seed(42)
X = np.sort(np.random.rand(100, 1) * 10, axis=0)
y = 0.5 * X**2 - 3 * X + 5 + np.random.randn(100, 1) * 5

# So sánh các polynomial degrees
degrees = [1, 2, 3, 5]
plt.figure(figsize=(12, 8))

for i, degree in enumerate(degrees, 1):
    # Tạo polynomial features và fit model
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(X)

    model = LinearRegression()
    model.fit(X_poly, y)

    # Predict
    X_test = np.linspace(0, 10, 100).reshape(-1, 1)
    X_test_poly = poly.transform(X_test)
    y_pred = model.predict(X_test_poly)

    # Plot
    plt.subplot(2, 2, i)
    plt.scatter(X, y, alpha=0.5, label='Data')
    plt.plot(X_test, y_pred, color='red', linewidth=2,
             label=f'Degree {degree}')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.legend()
    plt.title(f'Polynomial Regression (Degree {degree})')

plt.tight_layout()
plt.show()

```

## Sử dụng Pipeline (Cách chuyên nghiệp):

```
# Tạo pipeline kết hợp polynomial features và linear regression
poly_model = Pipeline([
    ('poly_features', PolynomialFeatures(degree=3)),
    ('linear_regression', LinearRegression())
])

# Fit và predict chỉ với 2 dòng code
poly_model.fit(X_train, y_train)
y_pred = poly_model.predict(X_test)

# Đánh giá
from sklearn.metrics import r2_score, mean_squared_error
print(f"R2 Score: {r2_score(y_test, y_pred):.4f}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.4f}")
```

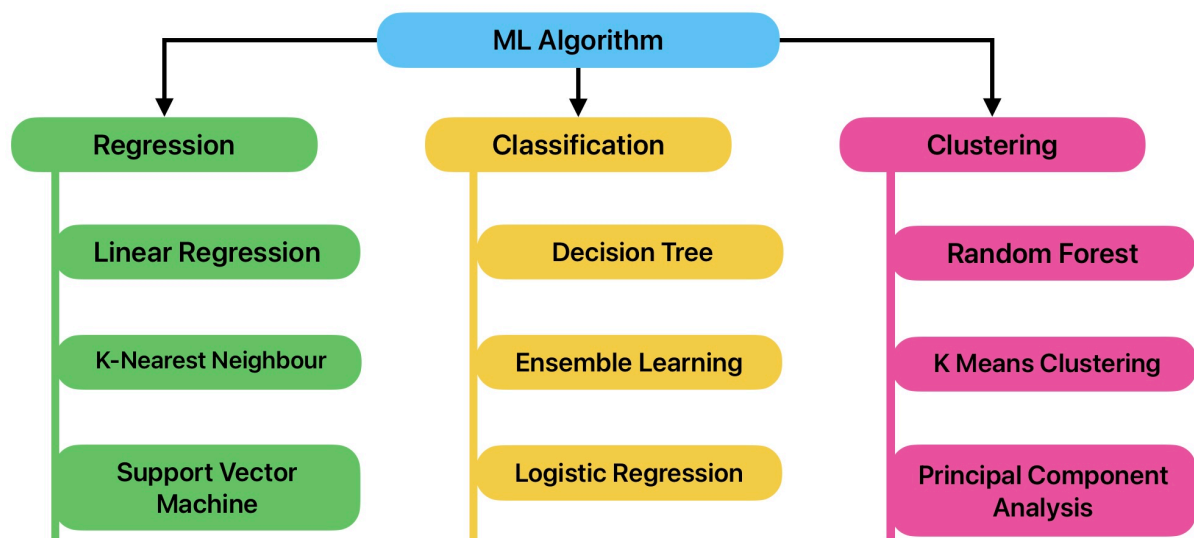


**Lưu ý:** Polynomial degree càng cao, model càng dễ bị **overfitting**. Cần sử dụng validation để chọn degree phù hợp!

## Các thuật toán mở rộng từ Linear Models

Ngoài Polynomial Regression, bạn có thể sử dụng:

- **Logistic Regression** - Classification
- **K-Nearest Neighbors (KNN)** - Classification/Regression
- **Decision Trees** - Classification/Regression
- **Support Vector Machines (SVM)** - Classification/Regression
- **Random Forests** - Ensemble method
- **Ensemble Methods** - Kết hợp nhiều models
- **Deep Learning Approaches** - Neural Networks



## Module 3: Cross Validation

### Giới thiệu Cross Validation



**Cross Validation** là kỹ thuật chia data thành nhiều folds để đánh giá model một cách đáng tin cậy hơn, tránh phụ thuộc vào một lần split cụ thể.

#### Ba loại Cross Validation phổ biến:

##### 1. K-Fold Cross Validation 📊

- Chia data thành K folds bằng nhau
- Train trên K-1 folds, test trên 1 fold còn lại
- Lặp lại K lần, mỗi fold làm test set 1 lần
- **Phổ biến nhất, thường dùng K=5 hoặc K=10**

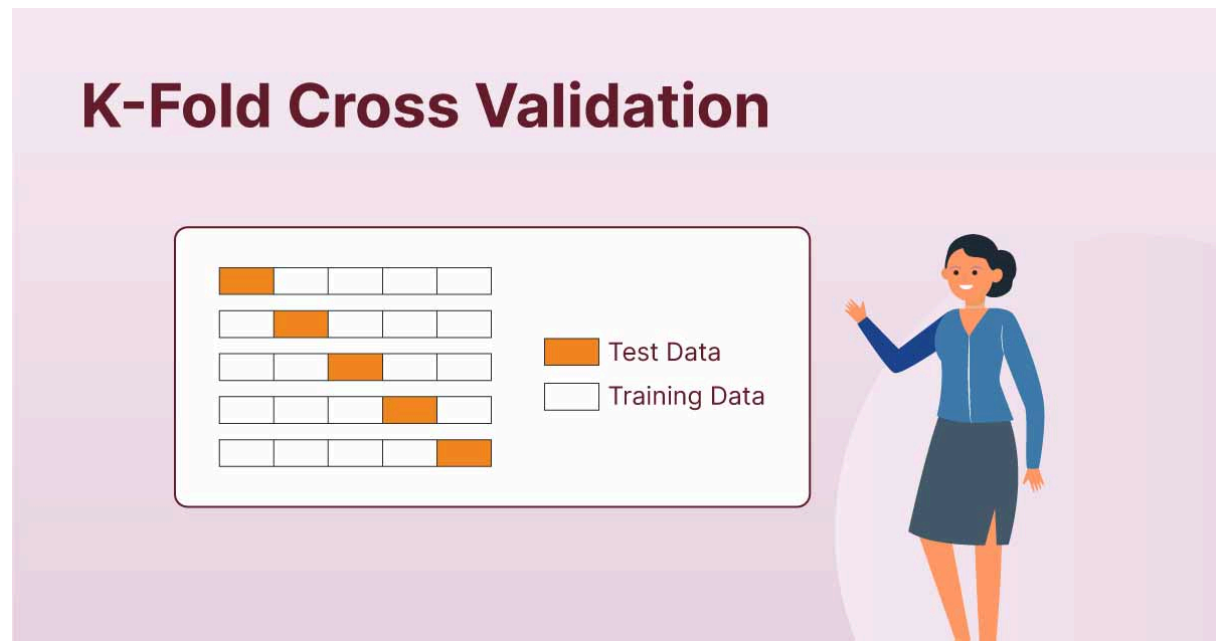
##### 2. Leave-One-Out Cross Validation (LOOCV) 🎲

- Trường hợp đặc biệt:  $K = n$  (số samples)
- Mỗi lần test trên 1 sample duy nhất

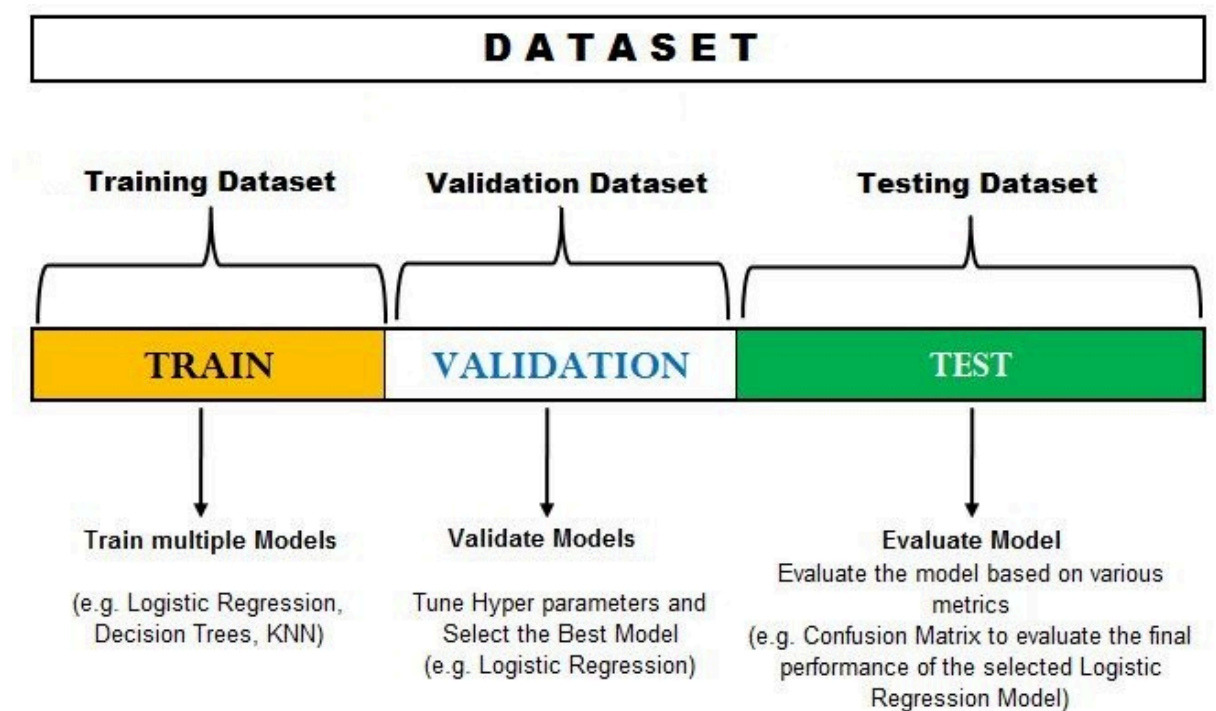
- Tốn thời gian với dataset lớn

### 3. Stratified Cross Validation ⚖️

- Đảm bảo tỷ lệ classes giống nhau ở mỗi fold
- Quan trọng với imbalanced data



## Ba loại Data Sets trong Cross Validation



# Code Implementation - Cross Validation

## 1. Train-Test Split cơ bản:

```
from sklearn.model_selection import train_test_split

# Chia data thành train và test (80-20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,    # 20% cho test
    random_state=42   # Để kết quả reproducible
)

print(f"Training samples: {len(X_train)}")
print(f"Test samples: {len(X_test)}")
```

## 2. K-Fold Cross Validation:

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression

# Khởi tạo K-Fold
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Train model và tính CV scores
model = LinearRegression()
cv_scores = cross_val_score(
    model, X, y,
    cv=kfold,
    scoring='r2' # Có thể dùng 'neg_mean_squared_error', 'r2', etc.
)

print(f"CV Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean():.4f}")
print(f"Std CV Score: {cv_scores.std():.4f}")
```

### 3. Cross Validation với Predictions:

```
from sklearn.model_selection import cross_val_predict

# Lấy predictions từ cross validation
y_pred_cv = cross_val_predict(
    model, X, y,
    cv=5
)

# Đánh giá
from sklearn.metrics import r2_score, mean_squared_error
print(f"CV R2 Score: {r2_score(y, y_pred_cv):.4f}")
print(f"CV RMSE: {np.sqrt(mean_squared_error(y, y_pred_cv)):.4f}")
```

### 4. GridSearchCV - Tìm Hyperparameters tốt nhất:

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

# Tạo pipeline
pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('linear', LinearRegression())
])

# Định nghĩa parameter grid
param_grid = {
    'poly__degree': [1, 2, 3, 4, 5], # Test các polynomial degrees
}

# GridSearchCV
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=5, # 5-fold CV
    scoring='r2',
```

```

    verbose=1
)

# Fit và tìm best parameters
grid_search.fit(X_train, y_train)

print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best CV Score: {grid_search.best_score_:.4f}")

# Sử dụng best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

```

## Tóm tắt Scikit-Learn Methods

| Method                         | Mục đích                 | Output                    |
|--------------------------------|--------------------------|---------------------------|
| <code>train_test_split</code>  | Chia train/test một lần  | Train & test sets         |
| <code>KFold</code>             | Tạo K-fold splits        | Cross validation iterator |
| <code>cross_val_score</code>   | Đánh giá score qua CV    | Array of scores           |
| <code>cross_val_predict</code> | Predictions qua CV       | Out-of-bag predictions    |
| <code>GridSearchCV</code>      | Tìm best hyperparameters | Best model & parameters   |

## Module 4, 5: Bias Variance Trade-off and Regularization Techniques

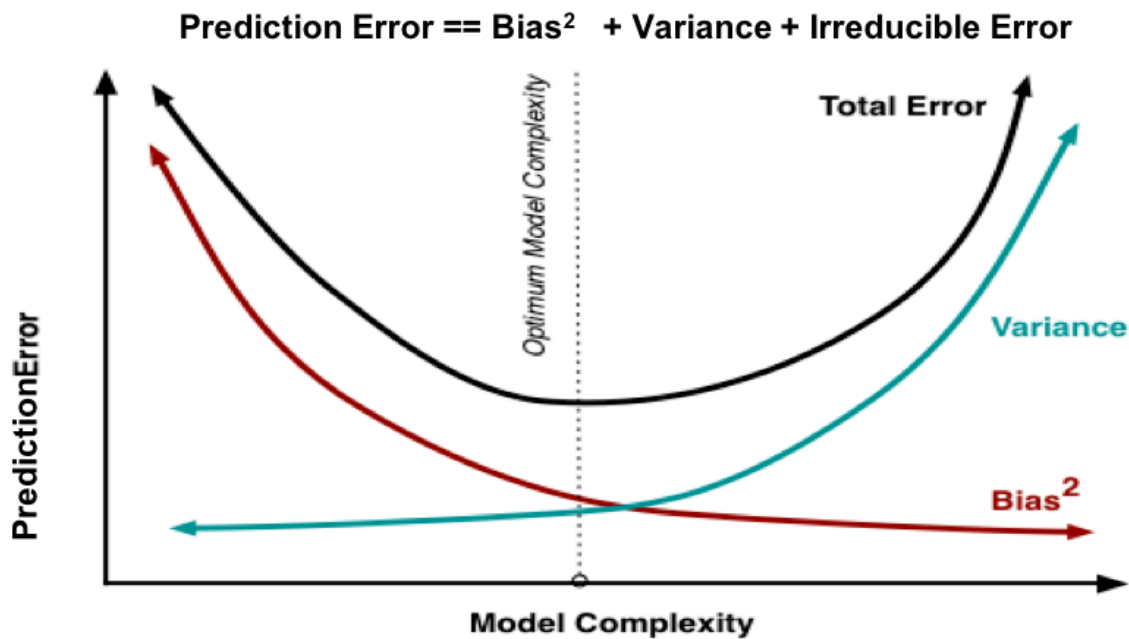
### Bias-Variance Tradeoff



**Bias-Variance Tradeoff** phân tích Mean Square Error (MSE) của model thành hai thành phần: **Bias** và **Variance**.

#### Công thức phân tích:

$$\text{Expected MSE} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



## Bias (Thiên lệch)

**Bias** đo lường mức độ model **gần với hàm thực tế** mà chúng ta đang cố gắng mô hình hóa.

### High Bias = **Underfitting**

Đặc điểm:

- Model **quá đơn giản**
- Không capture được patterns trong data
- **Kém trên cả training và test data**
- Training error và test error đều cao và gần nhau

**Ví dụ:** Dùng linear regression cho data có quan hệ phi tuyến phức tạp

### Cách cải thiện Bias:

**Làm model phức tạp hơn:**

- Thêm polynomial terms bậc cao hơn
- Thêm nhiều features hơn
- Sử dụng algorithms phức tạp hơn (Random Forest, Neural Networks)

**Thu thập thêm data** (nếu có thể)



**Feature engineering:** Tạo features mới có ý nghĩa

---

## Variance (Phương sai)

**Variance** đo lường **độ thay đổi** của predictions khi train trên các training sets khác nhau.

### High Variance = **Overfitting**

**Đặc điểm:**

- Model **quá phức tạp**
- "Học thuộc" training data, kể cả noise
- **Tốt trên training data, kém trên test data**
- Training error thấp, nhưng test error cao

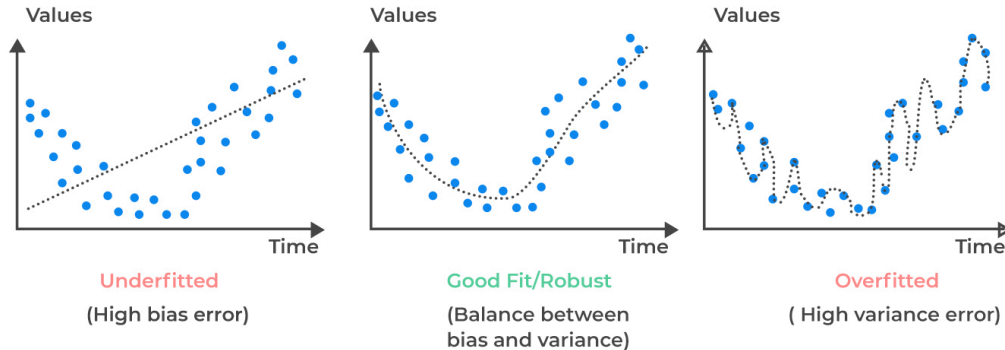
**Ví dụ:** Polynomial regression degree quá cao (degree 20 cho data đơn giản)

### Cách cải thiện Variance:

- **Làm model đơn giản hơn:**
  - Giảm polynomial degree
  - Loại bỏ features không cần thiết
  - Sử dụng feature selection
- **Thu thập thêm data** (càng nhiều càng tốt)
- **Sử dụng Regularization** ★ (Ridge, LASSO, Elastic Net)
- **Cross Validation** để đánh giá chính xác



## Generalization and Overfitting



## Ba kỹ thuật Regularization



**Regularization** giúp giảm overfitting bằng cách **penalize (phạt)** các **coefficients lớn**, làm cho model đơn giản hơn.

### 1 Ridge Regression (L2 Regularization)

Công thức Cost Function:

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

$$J(\beta) = \text{MSE} + \lambda \cdot \|\beta\|_2^2$$

Trong đó:

- $\lambda$  (lambda) = **regularization strength** (hyperparameter)
- $\|\beta\|_2^2$  = sum of squared coefficients (L2 norm)

**Đặc điểm của Ridge:**

- **Penalize** bình phương của coefficients

- **Shrink** coefficients về gần 0, nhưng **không bằng 0**
- **Giữ lại tất cả features**, chỉ minimize ảnh hưởng của features không quan trọng
- **Faster to train** so với LASSO
- Phù hợp khi **tất cả features đều có ích phần nào**

### Ảnh hưởng của $\lambda$ :

- $\lambda = 0$ : Model = Linear Regression thông thường
- $\lambda$  **nhỏ**: Ít regularization, có thể vẫn overfit
- $\lambda$  **lớn**: Nhiều regularization, coefficients  $\rightarrow 0$ , có thể underfit
- $\lambda$  **optimal**: Cần tìm bằng Cross Validation

### Code Implementation:

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# Tạo sample data
np.random.seed(42)
X = np.random.randn(100, 5)
y = 3*X[:, 0] + 2*X[:, 1] + X[:, 2] + np.random.randn(100) * 0.5

# QUAN TRỌNG: Scale features trước khi regularization!
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Ridge Regression với alpha (tương đương  $\lambda$ )
ridge = Ridge(alpha=1.0) # alpha =  $\lambda$ 
```

```

ridge.fit(X_train, y_train)

# Evaluate
train_score = ridge.score(X_train, y_train)
test_score = ridge.score(X_test, y_test)

print(f"Ridge Coefficients: {ridge.coef_}")
print(f"Train R2: {train_score:.4f}")
print(f"Test R2: {test_score:.4f}")

```

## Tìm Alpha tốt nhất với Cross Validation:

```

from sklearn.linear_model import RidgeCV

# Test nhiều alpha values
alphas = np.logspace(-4, 4, 100) # 10-4 đến 104

# RidgeCV tự động tìm best alpha qua CV
ridge_cv = RidgeCV(alphas=alphas, cv=5, scoring='r2')
ridge_cv.fit(X_train, y_train)

print(f"Best Alpha: {ridge_cv.alpha_:.4f}")
print(f"Best CV Score: {ridge_cv.score(X_test, y_test):.4f}")

# Visualize coefficients vs alpha
plt.figure(figsize=(10, 6))
coefs = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    coefs.append(ridge.coef_)

plt.plot(alphas, coefs)
plt.xscale('log')
plt.xlabel('Alpha ( $\lambda$ )', fontsize=12)
plt.ylabel('Coefficients', fontsize=12)
plt.title('Ridge Coefficients vs Regularization Strength', fontsize=14)
plt.axvline(ridge_cv.alpha_, color='red', linestyle='--', label='Best Alpha')

```

```
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```



**Lưu ý quan trọng:** Luôn **scale features** (StandardScaler) trước khi dùng regularization! Nếu không, features với scale lớn sẽ bị penalize nhiều hơn một cách không công bằng.

## 2 LASSO Regression (L1 Regularization)

**LASSO** = Least **A**bsolute **S**hrinkage and **S**election **O**perator

**Công thức Cost Function:**

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

$$J(\beta) = \text{MSE} + \lambda \cdot \|\beta\|_1$$

Trong đó:

- $\|\beta\|_1$  = sum of absolute values of coefficients (L1 norm)

**Đặc điểm của LASSO:**

- **Penalize** giá trị tuyệt đối của coefficients
- **Set coefficients = 0** cho features không quan trọng
- **Automatic feature selection**
- Tạo **sparse models** (nhiều coefficients = 0)
- **Interpretable:** Dễ hiểu model hơn vì chỉ giữ lại features quan trọng
- Phù hợp khi có **nhiều features không cần thiết**

**Code Implementation:**

```
from sklearn.linear_model import Lasso, LassoCV
from sklearn.preprocessing import StandardScaler
```

```

import numpy as np
import matplotlib.pyplot as plt

# Tạo data với một số features không quan trọng
np.random.seed(42)
X = np.random.randn(100, 10) # 10 features
# Chỉ 3 features đầu thực sự ảnh hưởng đến y
y = 3*X[:, 0] + 2*X[:, 1] + X[:, 2] + np.random.randn(100) * 0.5

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

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

print("LASSO Coefficients:")
for i, coef in enumerate(lasso.coef_):
    print(f" Feature {i}: {coef:.4f}")

# Đếm số features được giữ lại
non_zero = np.sum(lasso.coef_ != 0)
print(f"\nNumber of features selected: {non_zero} / {len(lasso.coef_)}")
print(f"Test R2: {lasso.score(X_test, y_test):.4f}")

```

## Tìm Alpha tốt nhất:

```

# LassoCV với cross validation
alphas = np.logspace(-4, 1, 100)
lasso_cv = LassoCV(alphas=alphas, cv=5, random_state=42, max_iter=10000)
lasso_cv.fit(X_train, y_train)

```

```

print(f"Best Alpha: {lasso_cv.alpha_:.4f}")
print(f"Number of features selected: {np.sum(lasso_cv.coef_ != 0)}")

# Visualize: Coefficients vs Alpha (LASSO Path)
from sklearn.linear_model import lasso_path

alphas_path, coefs_path, _ = lasso_path(X_train, y_train, alphas=alphas)

plt.figure(figsize=(12, 6))
plt.plot(alphas_path, coefs_path.T)
plt.xscale('log')
plt.xlabel('Alpha ( $\lambda$ )', fontsize=12)
plt.ylabel('Coefficients', fontsize=12)
plt.title('LASSO Path: Coefficients vs Regularization Strength', fontsize=14)
plt.axvline(lasso_cv.alpha_, color='red', linestyle='--',
            linewidth=2, label='Best Alpha (CV)')
plt.legend([f'Feature {i}' for i in range(X.shape[1])] + ['Best Alpha'])
plt.grid(True, alpha=0.3)
plt.show()

```

### 3 Elastic Net (L1 + L2 Regularization)

**Công thức Cost Function:**

$$J(\beta) = \text{MSE} + \lambda_1 \sum_{j=1}^p |\beta_j| + \lambda_2 \sum_{j=1}^p \beta_j^2$$

Hoặc viết theo dạng có **mixing parameter  $\alpha$** :

$$J(\beta) = \text{MSE} + \lambda \left( \alpha \|\beta\|_1 + \frac{1-\alpha}{2} \|\beta\|_2^2 \right)$$

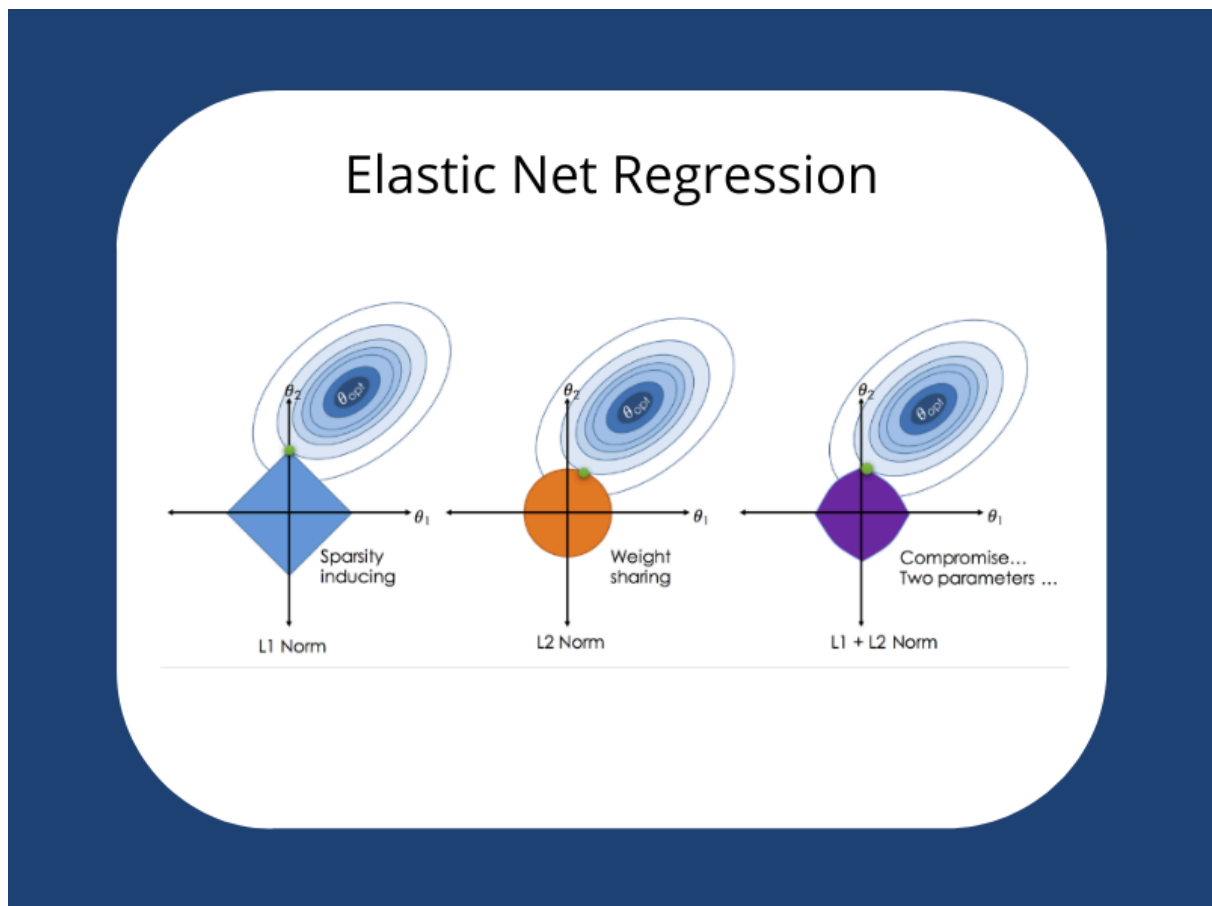
Trong đó:

- $\alpha$  = mixing parameter ( $0 \leq \alpha \leq 1$ )
  - $\alpha = 0$ : Pure Ridge (L2)

- $\alpha = 1$ : Pure LASSO (L1)
- $0 < \alpha < 1$ : Combination (thường dùng)

### Đặc điểm của Elastic Net:

- **Combines penalties** từ cả L1 và L2
- **Feature selection** như LASSO (set coefficients = 0)
- **Stability** như Ridge (coefficients ít fluctuate hơn)
- Tốt khi có **correlated features**
- **Flexible**: Tune được  $\alpha$  để điều chỉnh L1/L2 ratio



### Code Implementation:

```
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.preprocessing import StandardScaler
import numpy as np
```



```

# Tạo data với correlated features
np.random.seed(42)
X = np.random.randn(100, 10)
# Tạo correlation: Feature 4 tương tự Feature 0
X[:, 4] = X[:, 0] + np.random.randn(100) * 0.1
# Tạo correlation: Feature 5 tương tự Feature 1
X[:, 5] = X[:, 1] + np.random.randn(100) * 0.1

y = 3*X[:, 0] + 2*X[:, 1] + X[:, 2] + np.random.randn(100) * 0.5

# Scale
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Elastic Net với alpha ( $\lambda$ ) và l1_ratio ( $\alpha$  trong công thức)
elastic_net = ElasticNet(
    alpha=0.1,      # regularization strength ( $\lambda$ )
    l1_ratio=0.5,   # mixing parameter ( $\alpha$ ): 0.5 = equal L1 and L2
    random_state=42
)
elastic_net.fit(X_train, y_train)

print("Elastic Net Coefficients:")
for i, coef in enumerate(elastic_net.coef_):
    print(f" Feature {i}: {coef:.4f}")

print(f"\nNon-zero coefficients: {np.sum(elastic_net.coef_ != 0)}")
print(f"Test R2: {elastic_net.score(X_test, y_test):.4f}")

```

**Tìm hyperparameters tốt nhất:**

```
# ElasticNetCV tự động tìm alpha và l1_ratio
elastic_cv = ElasticNetCV(
    l1_ratio=[.1, .5, .7, .9, .95, .99, 1], # test các l1_ratio
    alphas=np.logspace(-4, 1, 50),        # test các alpha
    cv=5,
    random_state=42,
    max_iter=10000
)
elastic_cv.fit(X_train, y_train)

print(f"Best Alpha ( $\lambda$ ): {elastic_cv.alpha_:.4f}")
print(f"Best L1 Ratio ( $\alpha$ ): {elastic_cv.l1_ratio:.4f}")
print(f"Non-zero coefficients: {np.sum(elastic_cv.coef_ != 0)}")
print(f"Test R2: {elastic_cv.score(X_test, y_test):.4f}")
```

## Ba cách hiểu Regularization



Regularization có **3 interpretations** khác nhau, tất cả đều đúng và bổ sung cho nhau!

### 1. Analytical Interpretation (Toán học)

Thêm penalty term vào cost function:

$$\min_{\beta} \{ \text{MSE}(\beta) + \lambda \cdot \text{Penalty}(\beta) \}$$

- Penalty lớn  $\rightarrow$  Coefficients bị shrink
- Trade-off giữa fit data và keep coefficients small

### 2. Geometric Interpretation (Hình học)

Constraint optimization problem:

$$\min_{\beta} \text{MSE}(\beta) \quad \text{subject to} \quad \|\beta\| \leq C$$

- Ridge: Constraint region là **sphere** (L2 ball)

- LASSO: Constraint region là **diamond** (L1 ball)
- Solution = điểm MSE contours chạm constraint region

### 3. Probabilistic Interpretation (Xác suất)

Regularization = **Prior distribution** trên coefficients

**Ridge = Gaussian prior:**

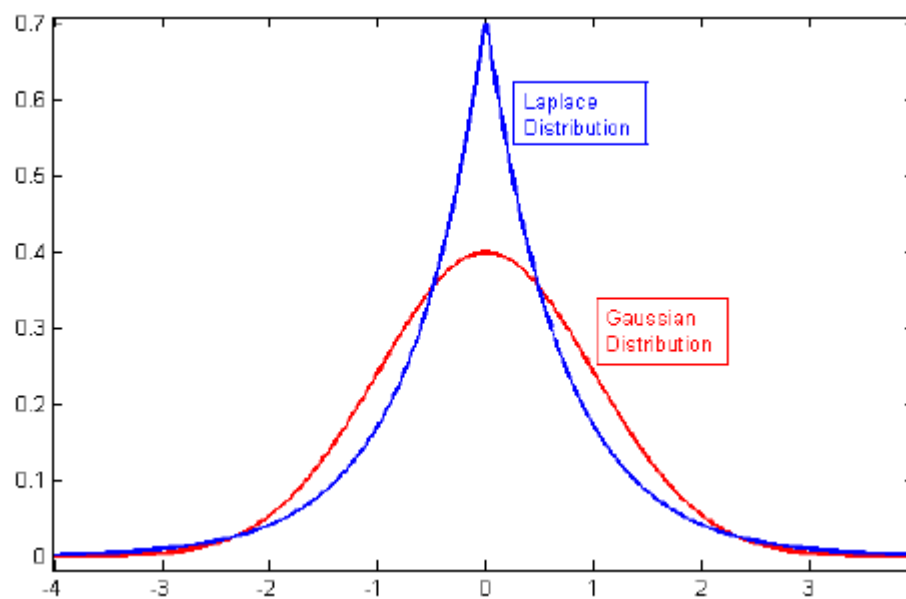
$$\beta_j \sim \mathcal{N}(0, \tau^2)$$

- Believe coefficients are normally distributed around 0

**LASSO = Laplace prior:**

$$p(\beta_j) \propto \exp(-\lambda|\beta_j|)$$

- Believe coefficients have sharp peak at 0 (sparse)



## Final Recommendations



**Mẹo nhỏ:**

- ◆ **Start with Ridge** nếu không chắc
- ◆ **Try LASSO** nếu có nhiều features ( $>50$ ) và nghĩ nhiều features không quan trọng
- ◆ **Use Elastic Net** nếu features correlated hoặc LASSO không stable
- ◆ **Always use Cross Validation** để chọn hyperparameters
- ◆ **Always scale features** trước khi regularize!