CE49X: Introduction to Machine Learning

Foundations of Machine Learning with Scikit-Learn

Dr. Eyuphan Koc

Department of Civil Engineering
Bogazici University

Fall 2025

Based on *Python Data Science Handbook* by Jake VanderPlas
Chapter 5: Machine Learning (Sections 5.0–5.6)
https://jakevdp.github.io/PythonDataScienceHandbook/

Lecture Outline

Outline

What is Machine Learning?

Definition:

- Machine Learning is about building **mathematical models** to understand data
- Fundamentally, it's a **data-driven approach** to learning patterns
- Models learn from examples rather than explicit programming

Key Idea:

"Instead of programming explicit rules, we provide examples and let the algorithm discover the patterns."

Civil Engineering Example

Rather than coding rules for predicting concrete strength, we provide examples of mix designs and their measured strengths—the model learns the relationship.

Categories of Machine Learning

Supervised Learning

- Learn from **labeled** data
- Have input-output pairs
- Goal: predict outputs for new inputs

Two Types:

1. Classification: Predict discrete labels
2. Regression: Predict continuous values

Unsupervised Learning

- Learn from **unlabeled** data
- No predefined outputs
- Goal: discover structure

Two Types:

1. Clustering: Group similar items
2. Dimensionality Reduction: Compress data

Key Difference

Supervised: "Here are examples with answers" vs Unsupervised: "Find patterns in this data"

Supervised Learning: Classification vs Regression

Classification

- Predict **discrete categories**
- Output is a label or class

Examples:

- Email: spam or not spam?
- Image: cat or dog?
- Soil type: clay, sand, or silt?
- Structure: safe or unsafe?

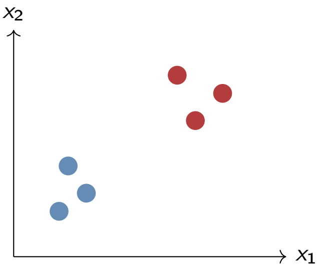
Regression

- Predict **continuous values**
- Output is a number

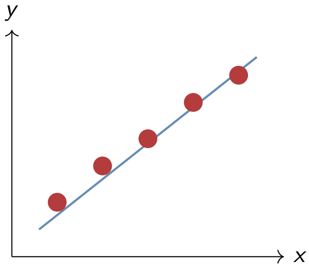
Examples:

- House price prediction
- Temperature forecasting
- Concrete strength from mix design
- Bridge deflection under load

Visualizing Classification and Regression



Classification



Regression

Unsupervised Learning: Clustering vs Dimensionality Reduction

Clustering

- Group similar data points
- No predefined labels
- Discover natural groupings

Examples:

- Customer segmentation
- Document organization
- Grouping similar bridge designs
- Identifying failure patterns

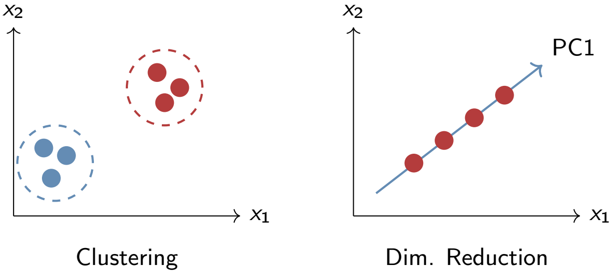
Dimensionality Reduction

- Reduce number of features
- Preserve important information
- Visualization & compression

Examples:

- Image compression
- Feature extraction
- Compress sensor data streams
- Visualize high-D material properties

Visualizing Clustering and Dimensionality Reduction

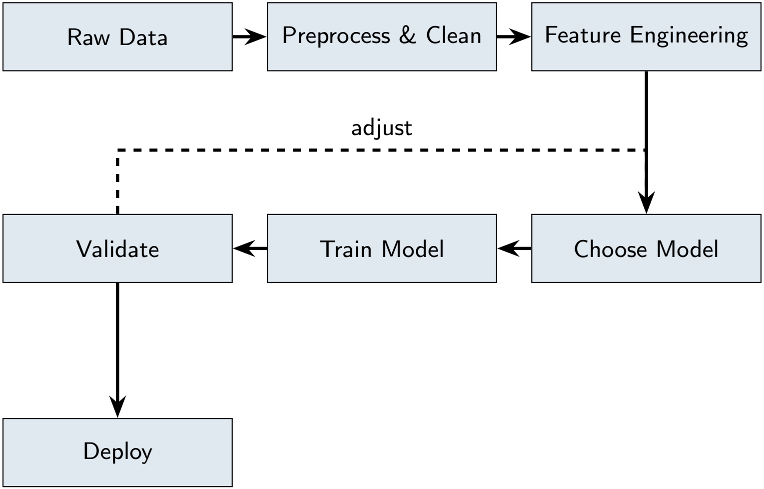


Machine Learning Workflow

Key Steps:

1. **Data Collection & Preprocessing:** Gather, clean, normalize data
2. **Feature Engineering:** Select/create informative features
3. **Model Selection & Training:** Choose algorithm, fit to data
4. **Validation:** Test on unseen data, tune hyperparameters
5. **Deployment:** Use model in production

Machine Learning Workflow Diagram



Outline

What is Scikit-Learn?

Scikit-Learn is Python's premier machine learning library

Key Features:

- **Consistent API:** All models follow the same interface
- **Comprehensive:** Classification, regression, clustering, dimensionality reduction
- **Well-documented:** Excellent documentation and examples
- **Built on NumPy/SciPy:** Fast and efficient
- **Open-source:** Free and actively maintained

Installation:

```
pip install scikit-learn
# or
conda install scikit-learn
```

Why Scikit-Learn?

Unified interface means learning one model teaches you all models!

Data Representation in Scikit-Learn

Two fundamental data structures:

1. **Features Matrix:** X

- Shape: $[n_{samples}, n_{features}]$
- Usually denoted as X
- Each row: one sample
- Each column: one feature
- Typically a 2D NumPy array or pandas DataFrame

2. **Target Array:** y

- Shape: $[n_{samples}]$
- Usually denoted as y
- Labels (classification) or values (regression)
- Typically a 1D NumPy array or pandas Series

The Estimator API

All Scikit-Learn models follow the same pattern:

1. **Choose a model class** and import it
2. **Choose hyperparameters** by instantiating the class
3. **Arrange data** into features matrix X and target vector y
4. **Fit the model** to your data with `.fit()`
5. **Apply the model** with `.predict()` or `.transform()`

Universal Interface:

```
from sklearn.some_module import SomeModel

# 1. Choose model and hyperparameters
model = SomeModel(hyperparameter1=value1,
                  hyperparameter2=value2)

# 2. Fit to data
model.fit(X, y)

# 3. Predict on new data
predictions = model.predict(X_new)
```

Problem: Predict concrete strength from water-cement ratio

```python
import numpy as np
from sklearn.linear_model import LinearRegression

# Generate sample data (water-cement ratio vs strength)
X = np.array([[0.4], [0.45], [0.5], [0.55], [0.6], [0.65]])
y = np.array([45, 40, 35, 30, 25, 20])  # Strength in MPa

# 1. Choose model
model = LinearRegression()

# 2. Fit model
model.fit(X, y)

# 3. Make predictions
X_new = np.array([[0.48], [0.58]])
predictions = model.predict(X_new)

print(f"Predictions: {predictions}")
print(f"Slope: {model.coef_[0]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")
```

Output

```
Predictions:  [37.5 27.5] | Slope:  -50.00 | Intercept:  65.00
```

Example: Classification with Iris Dataset

Problem: Classify iris flowers based on petal/sepal measurements

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Load data
iris = load_iris()
X, y = iris.data, iris.target

# Split data: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Create and train model (k=3 nearest neighbors)
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

# Evaluate accuracy
accuracy = model.score(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2%}")
```

Civil Engineering Analogy

Replace iris measurements with soil properties (grain size, moisture, density) to classify soil types (clay, silt, sand).

Unsupervised Learning: PCA Example

Principal Component Analysis (PCA): Reduce dimensionality while preserving variance

```python
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

# Load high-dimensional data (4 features)
iris = load_iris()
X = iris.data  # Shape: (150, 4)

# Reduce to 2 dimensions for visualization
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)  # Shape: (150, 2)

# How much variance is explained?
print(f"Explained variance: {pca.explained_variance_ratio_}")
print(f"Total: {sum(pca.explained_variance_ratio_):.2%}")
```

Engineering Application

Compress multi-sensor structural health monitoring data from 100 sensors to 5 principal components, retaining 95% of information.

K-Means: Group data into *k* clusters

```python
from sklearn.cluster import KMeans
import numpy as np

# Sample data: structural damage measurements
X = np.array([[1, 2], [1.5, 1.8], [5, 8],
              [8, 8], [1, 0.6], [9, 11]])

# Create 2 clusters
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)

# Get cluster labels
labels = kmeans.labels_
print(f"Cluster assignments: {labels}")

# Get cluster centers
centers = kmeans.cluster_centers_
print(f"Cluster centers:\n{centers}")
```

Civil Engineering Application

Cluster bridge inspection data to identify structures with similar damage patterns for targeted maintenance strategies.

Outline

Why Model Validation?

The Fundamental Problem:

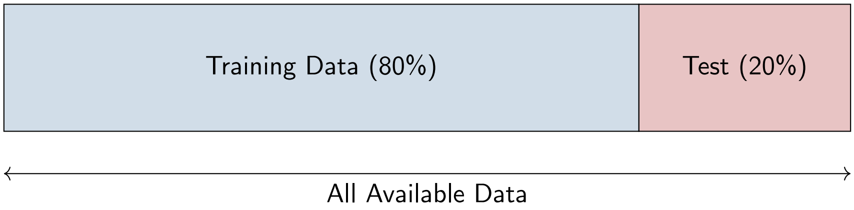
- We want models that **generalize** to new, unseen data
- Simply fitting training data is not enough
- Need to estimate performance on future data

Common Mistake: Training on Test Data

WRONG: Evaluate model on the same data used for training
Result: Overly optimistic performance estimates

Solution: Hold out a separate **test set**

Train-Test Split Visualization



Train-Test Split

Basic Approach: Split data into training and testing sets

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Split: 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Fit on training data only
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluate on test data
train_score = model.score(X_train, y_train)
test_score = model.score(X_test, y_test)

print(f"Training R^2: {train_score:.3f}")
print(f"Test R^2: {test_score:.3f}")
```

Key Points

- `random_state`: ensures reproducibility
- **Never** use test data during training or hyperparameter tuning
- Test set estimates performance on unseen data

Cross-Validation

Problem with single train-test split:

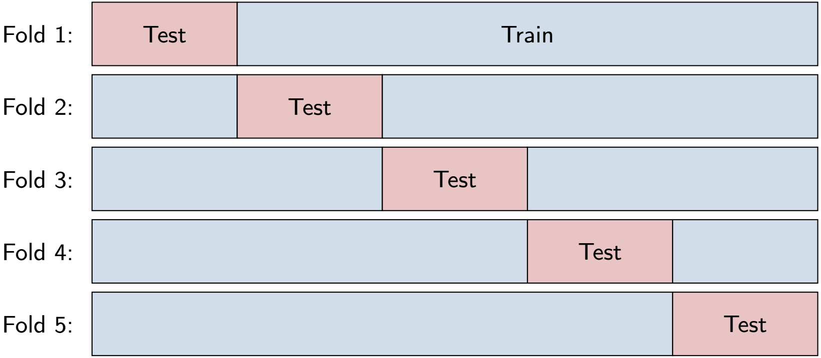
- Performance depends on which samples ended up in test set
- Wastes data (only 80% used for training)

Solution: K-Fold Cross-Validation

Process:

1. Split data into k equal parts (folds)
2. Train on $k - 1$ folds, test on the remaining fold
3. Repeat k times, each fold used as test set once
4. Average the k performance scores

K-Fold Cross-Validation Diagram



K-Fold Cross-Validation in Scikit-Learn

```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Create model
model = LinearRegression()

# Perform 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5,
                         scoring='r2')

print(f"Cross-validation scores: {scores}")
print(f"Mean R^2: {scores.mean():.3f}")
print(f"Std Dev: {scores.std():.3f}")
```

Advantages:

- More robust performance estimate
- Uses all data for both training and validation
- Provides variance estimate (standard deviation)

Typical Choice

5-fold or 10-fold cross-validation is standard. Use more folds for small datasets.

Bias-Variance Tradeoff

Two sources of model error:

Bias (Underfitting)

- Model too simple
- Cannot capture true pattern
- High training error
- High test error

Example:
Linear model for nonlinear data

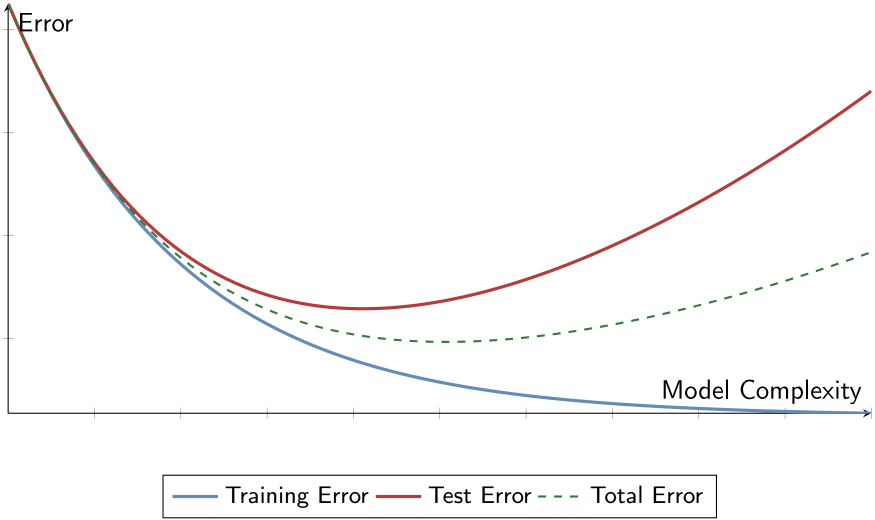
Variance (Overfitting)

- Model too complex
- Fits noise in training data
- Low training error
- High test error

Example:
High-degree polynomial

Goal: Find the sweet spot with minimum test error!

Bias-Variance Tradeoff Visualization



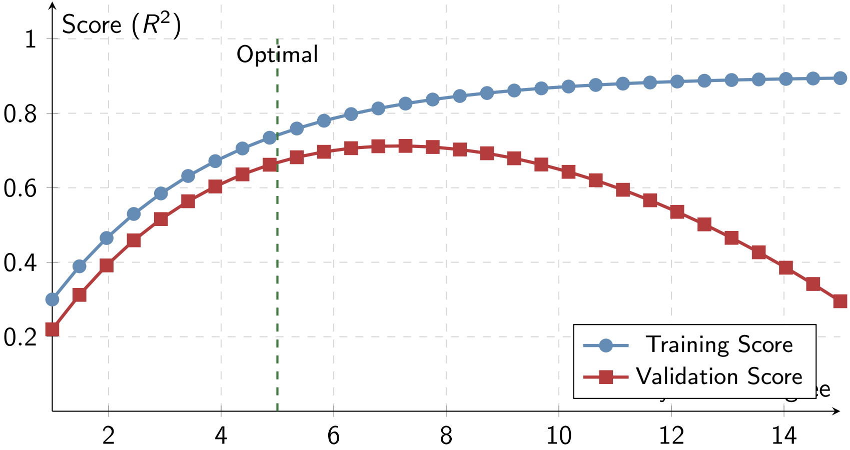
Validation Curves

Validation Curve: Plot performance vs. a single hyperparameter

Purpose:

- Visualize bias-variance tradeoff
- Select optimal hyperparameter value
- Diagnose under/overfitting

Example Validation Curve



Creating Validation Curves

```python
from sklearn.model_selection import validation_curve
from sklearn.linear_model import Ridge
import numpy as np

# Test different regularization strengths
param_range = np.logspace(-4, 4, 10)

train_scores, val_scores = validation_curve(
    Ridge(), X, y,
    param_name='alpha',
    param_range=param_range,
    cv=5,
    scoring='r2'
)

# Average across folds
train_mean = train_scores.mean(axis=1)
val_mean = val_scores.mean(axis=1)

# Find best alpha
best_alpha = param_range[val_mean.argmax()]
print(f"Best alpha: {best_alpha:.4f}")
```

Engineering Application

Tune regularization strength when predicting structural response to prevent overfitting to measurement noise.

Learning Curves

Learning Curve: Plot performance vs. training set size

Purpose:

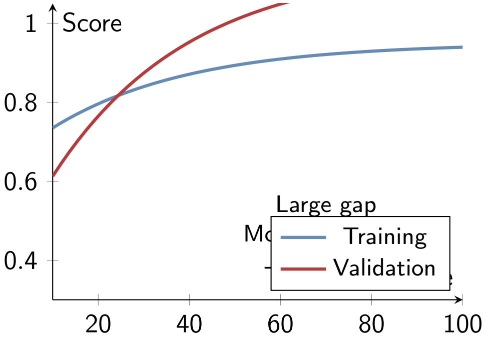
- Diagnose whether more data will help
- Identify high bias vs. high variance

Diagnosis:

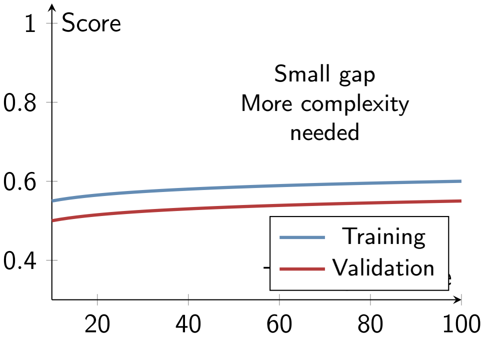
- **Large gap:** High variance $\rightarrow$ more data or regularization
- **Converged low:** High bias $\rightarrow$ more complex model

Learning Curves: High Variance vs High Bias

High Variance (Overfitting)



High Bias (Underfitting)



Grid Search for Hyperparameter Tuning

Problem: Many models have multiple hyperparameters to tune

Grid Search: Try all combinations of hyperparameters

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Define parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.001, 0.01, 0.1, 1],
    'kernel': ['rbf', 'linear']
}

# Create grid search with 5-fold CV
grid = GridSearchCV(SVC(), param_grid, cv=5,
                    scoring='accuracy')

# Fit searches all combinations
grid.fit(X_train, y_train)

print(f"Best parameters: {grid.best_params_}")
print(f"Best CV score: {grid.best_score_:.3f}")

# Use best model for predictions
best_model = grid.best_estimator_
```

Outline

Linear Regression: The Foundation

Goal: Fit a linear relationship between features and target

Model:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p = w_0 + \sum_{j=1}^{p} w_j x_j$$

Where: $\hat{y}$ = predicted value, x_j = features, w_j = weights, w_0 = intercept

Learning Objective: Find weights that minimize error

$$\text{minimize} \quad \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - w_0 - \sum_{j=1}^{p} w_j x_{ij} \right)^2$$

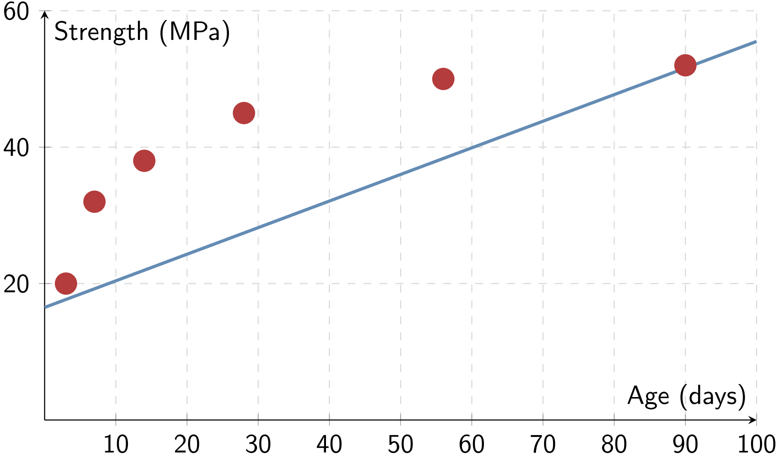
Civil Engineering Example

Predict concrete compressive strength from: cement content, water ratio, age, aggregate size, etc.

Simple Linear Regression Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Data: Age vs Concrete Strength
age = np.array([3, 7, 14, 28, 56, 90])
strength = np.array([20, 32, 38, 45, 50, 52])

X = age.reshape(-1, 1)
y = strength

# Fit model
model = LinearRegression()
model.fit(X, y)

# Coefficients
print(f"Slope: {model.coef_[0]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")
print(f"R^2: {model.score(X, y):.3f}")

# Predict
age_new = np.array([[21], [42]])
pred = model.predict(age_new)
```

Linear Regression Fit Example



Interpretation:

- Each day adds 0.39 MPa

Polynomial Regression

Idea: Use polynomial features to fit nonlinear relationships

Transform:

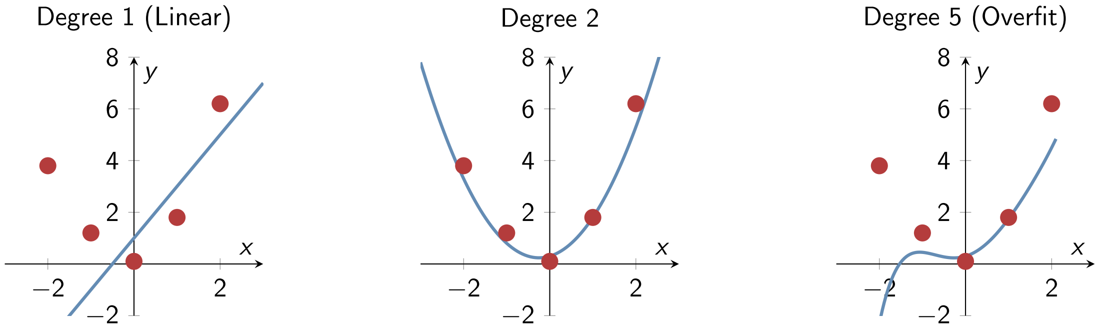
$$x \to [x, x^2, x^3, \ldots, x^d]$$

Then apply linear regression: $\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$

Warning

Higher degree $\to$ more flexibility $\to$ risk of overfitting!

Polynomial Regression: Degree Comparison



Polynomial Features in Scikit-Learn

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

# Original data
X = np.array([[x] for x in range(10)])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# Create polynomial regression pipeline
# Degree 3: [1, x, x^2, x^3]
model = make_pipeline(
    PolynomialFeatures(degree=3),
    LinearRegression()
)

# Fit and predict
model.fit(X, y)
y_pred = model.predict(X)

# Evaluate
r2 = model.score(X, y)
print(f"R^2 score: {r2:.3f}")
```

Pipeline: Chains transformations automatically!

Regularization: Controlling Complexity

Problem: Complex models overfit to training data

Solution: Add penalty for large coefficients

Ridge Regression (L2)

$$\text{minimize} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{p} w_j^2$$

- Shrinks coefficients
- Keeps all features
- α: regularization strength

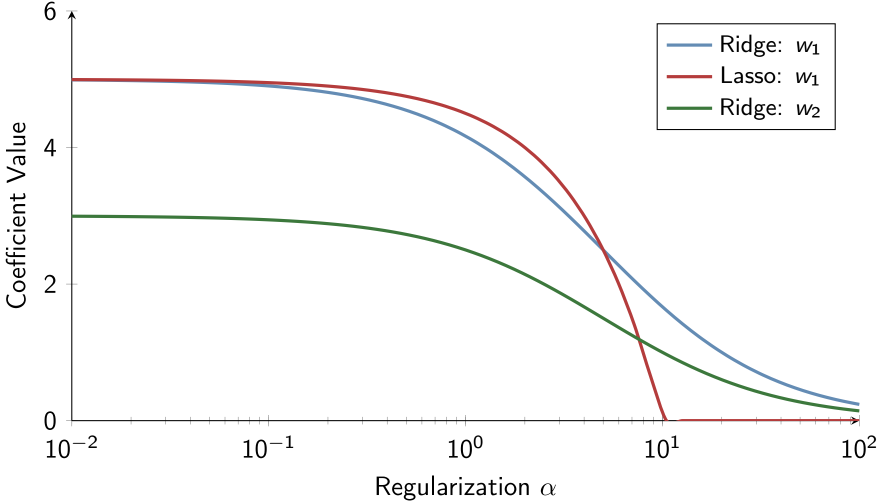
Lasso Regression (L1)

$$\text{minimize} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{p} |w_j|$$

- Shrinks some to exactly zero
- Performs feature selection
- α: regularization strength

Key: Larger $\alpha \rightarrow$ stronger regularization $\rightarrow$ simpler model

Effect of Regularization on Coefficients



Ridge and Lasso in Scikit-Learn

```python
from sklearn.linear_model import Ridge, Lasso

# Ridge Regression (L2)
ridge = Ridge(alpha=1.0)  # Try 0.1, 1.0, 10.0
ridge.fit(X_train, y_train)
ridge_score = ridge.score(X_test, y_test)

# Lasso Regression (L1)
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
lasso_score = lasso.score(X_test, y_test)

# Compare coefficients
print(f"Ridge coefficients: {ridge.coef_}")
print(f"Lasso coefficients: {lasso.coef_}")
print(f"Non-zero Lasso features: {np.sum(lasso.coef_ != 0)}")
```

When to Use Which?

Ridge: When all features are potentially relevant
Lasso: When you want automatic feature selection

Real-World Example: Bicycle Traffic Prediction

Problem: Predict daily bicycle traffic on Seattle's Fremont Bridge

Features:

- Temperature, precipitation
- Day of week, month
- Holiday indicator
- Hour of day (if hourly data)

Approach:

1. Feature engineering: add polynomial features for temperature
2. Add interaction terms (e.g., temp $\times$ weekend)
3. Use Ridge regression to prevent overfitting
4. Validate with cross-validation

Outline

Key Takeaways

1. Machine Learning Fundamentals

- Supervised (classification, regression) vs Unsupervised (clustering, dim reduction)
- Data-driven approach to building predictive models

2. Scikit-Learn Workflow

- Consistent API: `fit()`, `predict()`, `transform()`
- Data representation: features matrix X, target vector y

3. Model Validation

- Never evaluate on training data
- Use train-test split or cross-validation
- Understand bias-variance tradeoff

4. Linear Regression

- Foundation for many ML algorithms
- Polynomial features for nonlinearity
- Regularization (Ridge/Lasso) prevents overfitting

Civil Engineering Applications

Machine Learning is transforming civil engineering:

1. **Structural Health Monitoring**
   - Classify damage types from sensor data
   - Predict remaining service life
2. **Material Science**
   - Predict concrete/steel properties from composition
   - Optimize mix designs
3. **Traffic & Transportation**
   - Traffic flow prediction and optimization
   - Route planning and demand forecasting
4. **Construction Management**
   - Project cost and duration estimation
   - Risk assessment and safety prediction
5. **Environmental Engineering**
   - Water quality prediction
   - Climate impact assessment

Next Steps in Machine Learning

Coming in Week 8:

- **Naive Bayes:** Probabilistic classification
- **Support Vector Machines:** Maximum-margin classifiers
- **Decision Trees & Random Forests:** Ensemble methods
- **Clustering:** K-Means, hierarchical clustering
- **Dimensionality Reduction:** PCA deep dive

Practice Resources:

- **Scikit-Learn Documentation:** `https://scikit-learn.org`
- **Kaggle:** Real-world datasets and competitions
- **Course Notebooks:** Hands-on examples in repository

Questions?

Thank you!

Dr. Eyuphan Koc
eyuphan.koc@bogazici.edu.tr

Office Hours: By appointment

Next Lecture: Advanced ML Algorithms
(Naive Bayes, SVM, Random Forests)