Machine Learning Project 4: Clustering Oscillator Trajectories

Clustering is an unsupervised learning method where you group data without predefined labels, unlike the supervised classification in Option 3. We’ll use the same damped oscillator trajectories from Option 3, extract features, and apply the K-Means algorithm to cluster them into groups (e.g., underdamped vs. overdamped) without telling the model the true labels upfront. This mimics real-world scenarios where you might explore patterns in experimental data without knowing the categories in advance.

Step 1: Setup in PyCharm

You’ll need numpy, matplotlib, scipy, and scikit-learn (already installed from Option 3). Create a new file called ml\_oscillator\_clustering.py.

Code: Clustering Oscillator Trajectories in PyCharm

Here’s the code:

python

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

from sklearn.cluster import KMeans

# Step 1: Define oscillator function

def oscillator(state, t, m, b, k):

x, v = state

dx\_dt = v

dv\_dt = -(b/m) \* v - (k/m) \* x

return [dx\_dt, dv\_dt]

# Step 2: Generate synthetic dataset with varying b

np.random.seed(0)

t = np.linspace(0, 10, 100) # Time array, 0 to 10 seconds

m = 1.0 # Mass (kg)

k = 2.0 # Spring constant (N/m)

b\_values = np.concatenate([np.linspace(0.1, 1.9, 50), np.linspace(2.1, 4.0, 50)]) # 50 underdamped, 50 overdamped

X = [] # Trajectories

true\_labels = [] # For comparison (not used in clustering)

critical\_damping = 2 \* np.sqrt(m \* k) # b\_crit = 2.0

for b in b\_values:

state0 = [1.0, 0.0] # Initial position=1, velocity=0

solution = odeint(oscillator, state0, t, args=(m, b, k))

X.append(solution[:, 0]) # Store position trajectory

true\_labels.append(0 if b < critical\_damping else 1) # True label (for validation)

X = np.array(X) # Shape: (100, 100)

true\_labels = np.array(true\_labels)

# Step 3: Feature engineering

X\_features = np.column\_stack([

np.max(np.abs(X), axis=1), # Max amplitude

np.sum(np.diff(X, axis=1) > 0, axis=1), # Number of direction changes

-np.log(np.max(np.abs(X[:, 50:]), axis=1) / np.max(np.abs(X), axis=1)) / t[50] # Decay rate

])

# Step 4: Apply K-Means clustering

kmeans = KMeans(n\_clusters=2, random\_state=0)

cluster\_labels = kmeans.fit\_predict(X\_features)

# Step 5: Compare with true labels (for learning purposes)

from sklearn.metrics import adjusted\_rand\_score

ari = adjusted\_rand\_score(true\_labels, cluster\_labels)

print(f"Adjusted Rand Index (similarity to true labels): {ari:.2f}")

# Step 6: Plot trajectories colored by cluster

plt.figure(figsize=(10, 6))

for i in range(len(X)):

color = 'blue' if cluster\_labels[i] == 0 else 'red'

plt.plot(t, X[i], color=color, alpha=0.3)

plt.xlabel('Time (s)')

plt.ylabel('Position (x)')

plt.title('Oscillator Trajectories by Cluster (Blue = Cluster 0, Red = Cluster 1)')

plt.grid(True)

plt.show()

# Step 7: Scatter plot of features with cluster labels

plt.figure(figsize=(8, 6))

plt.scatter(X\_features[:, 0], X\_features[:, 2], c=cluster\_labels, cmap='bwr', alpha=0.6)

plt.xlabel('Max Amplitude')

plt.ylabel('Decay Rate')

plt.title('Clusters in Feature Space (0 = Blue, 1 = Red)')

plt.grid(True)

plt.show()

# Step 8: Plot true labels for comparison

plt.figure(figsize=(8, 6))

plt.scatter(X\_features[:, 0], X\_features[:, 2], c=true\_labels, cmap='bwr', alpha=0.6)

plt.xlabel('Max Amplitude')

plt.ylabel('Decay Rate')

plt.title('True Labels (0 = Underdamped, 1 = Overdamped)')

plt.grid(True)

plt.show()

What’s Happening Here?

1. Data Generation: Same as Option 3—100 trajectories with ( b ) from 0.1 to 4.0, split around

bcrit=2.0b\_{crit} = 2.0b\_{crit} = 2.0

. We keep true\_labels for validation but don’t use them for clustering.

1. Feature Engineering: Extracted max amplitude, direction changes, and decay rate (same as Option 3).
2. K-Means Clustering:
   * KMeans(n\_clusters=2) assumes two groups (we expect underdamped and overdamped).
   * fit\_predict assigns each trajectory to a cluster (0 or 1).
3. Evaluation: Adjusted Rand Index (ARI) measures how well clusters match true labels (1.0 = perfect match, 0 = random).
4. Visualization:
   * Trajectories colored by cluster (blue = cluster 0, red = cluster 1).
   * Scatter plot of features (amplitude vs. decay rate) with cluster labels.
   * Scatter plot with true labels for comparison.

Running in PyCharm

1. Save and Run: Save (Ctrl+S) and hit “Run” (Shift+F10).
2. Output:
   * Console: ARI score (e.g., 0.9+ if clusters align well with underdamped/overdamped).
   * Plot 1: Trajectories colored by cluster assignment.
   * Plot 2: Feature scatter plot with cluster labels.
   * Plot 3: Feature scatter plot with true labels.

What You’ll See

* Trajectories: One cluster (e.g., blue) should show oscillatory behavior (underdamped), the other (e.g., red) smooth decay (overdamped).
* Feature Space: Clusters should separate based on decay rate (higher for overdamped) and amplitude/direction changes.
* ARI: High if clusters match true damping regimes, lower if they don’t (e.g., due to feature overlap near

b=2.0b = 2.0b = 2.0

).

Why Clustering?

* Unsupervised Learning: Unlike Option 3, we don’t need labels upfront—great for exploring unknown data (e.g., experimental signals).
* Physics Insight: Tests if the model can naturally group behaviors you understand (damping types).
* ML Skill: Adds a new technique to your toolkit beyond regression and classification.

Your Turn

Experiment in PyCharm:

1. Change Clusters: Set n\_clusters=3 in KMeans—does it find a third group (e.g., near-critical damping)?
2. Add Noise: After X = np.array(X), add X += np.random.randn(\*X.shape) \* 0.05. Does clustering hold up?
3. Elbow Method: Test how many clusters are optimal:

python

inertias = []

for k in range(1, 6):

kmeans = KMeans(n\_clusters=k, random\_state=0)

kmeans.fit(X\_features)

inertias.append(kmeans.inertia\_)

plt.plot(range(1, 6), inertias, 'bo-')

plt.xlabel('Number of Clusters')

plt.ylabel('Inertia')

plt.title('Elbow Method')

plt.show()

Look for an “elbow” (e.g., at 2) where adding clusters stops reducing inertia much.