Clustering & Principal Component Analysis

AERO 689: Machine Learning for Aerospace Engineers

Raktim Bhattacharya

Texas A&M University

Overview

Topics Covered

1. Classification vs Clustering
2. Clustering Algorithms
3. Cluster Validation
4. Principal Component Analysis (PCA)
5. PCA Applications
6. Integration with Clustering

Classification vs Clustering

The Fundamental Difference

Classification (Week 4): Supervised learning

- Requires labeled training data
- Predicts known categories
- Goal: Generalization to new examples
- Example: "Is this bearing fault type A, B, or C?"

Clustering (Week 5): Unsupervised learning

- No labels required
- Discovers hidden patterns
- Goal: Finding natural groupings
- Example: "What natural fault patterns exist?"

Mathematical Distinction

Classification: Learn $f : X \to Y$ where $Y = \{0, 1, \ldots, K-1\}$ is known

- Training data: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Labels y_i are given
- Optimize to minimize prediction error

Clustering: Find $f : X \to \{1, 2, \ldots, k\}$ discovered from data

- Training data: $x_1, x_2, \ldots, x_n$ (no labels!)
- Number of clusters k may be unknown
- Optimize to find natural groups

When to Use Each Method

Use Classification when:

- You have labeled examples
- Categories are well-defined (taxi, takeoff, cruise)
- Regulatory requirements demand specific fault detection
- Problem: Requires expensive labeling effort

Use Clustering when:

- No labeled data available
- Exploring new operational regimes
- Discovering anomalies never seen before
- Initial data exploration before labeling

Application Context

Classification Example (Week 4):

- Trained on known fault types (normal, fault_a, fault_b, fault_c)
- Predicts fault type on new sensor data
- Requires historical fault database

Clustering Example (Week 5):

- Discovers natural patterns from unlabeled data
- Finds new degradation modes without prior knowledge
- Explores operational envelope
- Creates labels for future classification

Key Insight: Clustering creates labels; Classification uses labels

Decision Framework

Do you have labeled data?

- **YES** $\rightarrow$ Classification
  - Logistic regression
  - Decision trees
  - SVM
  - Neural networks

Do you have labeled data?

- **NO** $\rightarrow$ Clustering
  - K-means
  - DBSCAN
  - Hierarchical
  - Gaussian Mixture Models

Validation Challenge: How do we know clustering is "correct" without ground truth?

Clustering Fundamentals

What is Clustering?

Goal: Partition data into groups (clusters) where:

- Points in same cluster are similar
- Points in different clusters are dissimilar

Mathematical Formulation:

Minimize within-cluster variance:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

where $\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$ is cluster center

K-Means Algorithm

Lloyd's Algorithm:

1. **Initialize**: Randomly select k cluster centers $\mu_1, \ldots, \mu_k$
2. **Assignment**: Assign each point to nearest center

$$c_i = \arg\min_j \|x_i - \mu_j\|^2$$

3. **Update**: Recompute centers as mean of assigned points

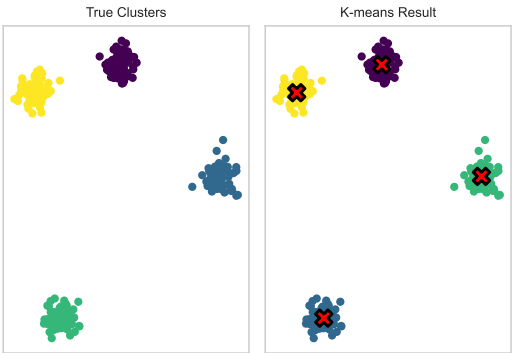
$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

4. **Repeat** steps 2-3 until convergence

Convergence: Guaranteed (objective function monotonically decreases)

Limitation: May converge to local minimum (solution: run multiple times)

K-Means Strength: Well-Separated Clusters



When K-means works perfectly:

- ✓ Spherical clusters
- ✓ Well-separated
- ✓ Similar sizes
- ✓ Similar densities

Result: Perfect match between true clusters and K-means assignments

K-Means Strength: Code

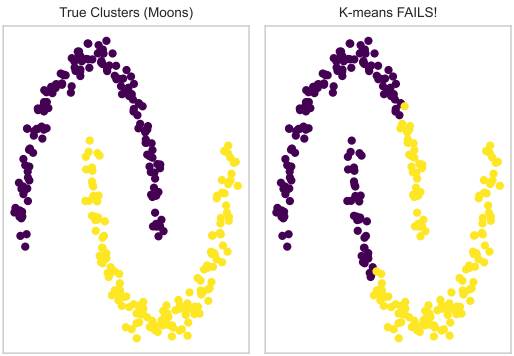
```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate well-separated clusters
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.6, random_state=42)

# Apply K-means
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
y_pred = kmeans.fit_predict(X)

# Plot comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.scatter(X[:, 0], X[:, 1], c=y_true, cmap='viridis')
ax2.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
ax2.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            marker='X', s=200, c='red', edgecolors='black')
```

K-Means Weakness 1: Non-Spherical Shapes



Problem: K-means assumes spherical clusters

Failure Mode:

- X Curved/non-convex shapes
- X K-means draws straight boundaries
- X Incorrectly splits moon shapes

Solution: Use DBSCAN or hierarchical clustering

K-Means Weakness 1: Code

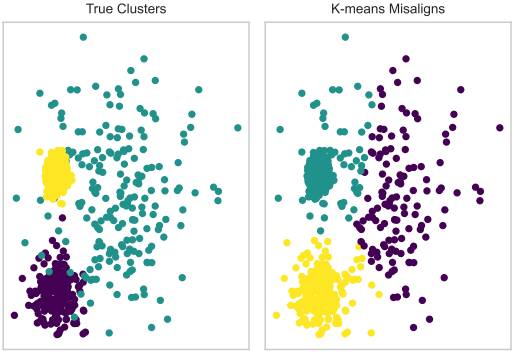
```
from sklearn.datasets import make_moons

# Generate moon-shaped clusters
X_moons, y_moons = make_moons(n_samples=300, noise=0.05,
                              random_state=42)

# K-means fails on non-spherical shapes
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
y_pred = kmeans.fit_predict(X_moons)

# Plot
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.scatter(X_moons[:, 0], X_moons[:, 1], c=y_moons)
ax1.set_title('True Clusters')
ax2.scatter(X_moons[:, 0], X_moons[:, 1], c=y_pred)
ax2.set_title('K-means Result')
```

K-Means Weakness 2: Different Densities



Problem: K-means assumes equal variance

Failure Mode:

- Different cluster spreads
- Large variance cluster dominates
- Boundary misplaced

Solution:

Standardize features: transform each feature to mean=0, std=1

$$z = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Or use GMM (handles different covariances)

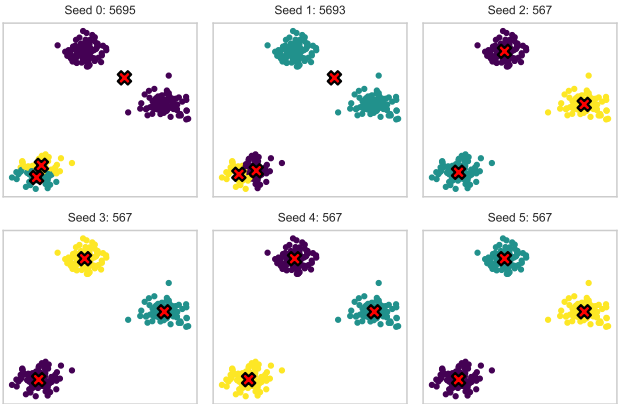
K-Means Weakness 2: Code

```
# Clusters with different variances
X1 = np.random.normal(0, 1, (200, 2))       # Std = 1
X2 = np.random.normal(5, 3, (200, 2))       # Std = 3
X3 = np.random.normal([0, 6], 0.5, (200, 2)) # Std = 0.5
X = np.vstack([X1, X2, X3])

# K-means struggles with different variances
kmeans = KMeans(n_clusters=3, random_state=42)
y_pred = kmeans.fit_predict(X)

# Large variance cluster dominates
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
```

K-Means Weakness 3: Initialization Sensitivity



Problem: Random initialization → different local minima

Failure Mode:

- Different results each run
- Some worse than others
- WCSS varies (Within-Cluster Sum of Squares)
  - **Lower WCSS** = better clustering (tighter clusters)
  - **Higher WCSS** = worse clustering (spread out)
  - Different seeds → different WCSS values

Observation: Seeds 0,2,4 good; Seeds 1,3,5 suboptimal

Solution: Use `n_init=10` (run 10 times, keep best)

K-Means Weakness 3: Code

```
# Demonstrate initialization sensitivity
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0)

# Run with different random seeds
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
for i, ax in enumerate(axes.flat):
    km = KMeans(n_clusters=3, init='random',
                n_init=1, random_state=i)
    y = km.fit_predict(X)
    ax.scatter(X[:, 0], X[:, 1], c=y)
    ax.scatter(km.cluster_centers_[:, 0],
               km.cluster_centers_[:, 1],
               marker='X', s=200, c='red')
    ax.set_title(f'WCSS={km.inertia_:.0f}')

# Solution: n_init=10 (default) runs 10 times, keeps best
```

Cluster Validation

How Do We Know Clustering is Good?

Challenge: No ground truth labels!

Validation Strategies:

1. **Internal validation**: Use only the data
   - Silhouette score
   - Within-cluster sum of squares (WCSS)
   - Davies-Bouldin index
2. **External validation**: Compare with known labels (if available)
   - Adjusted Rand Index (ARI)
   - Normalized Mutual Information (NMI)
3. **Domain validation**: Domain expertise
   - Do clusters make physical sense?
   - Can we interpret clusters meaningfully?

Elbow Method

Goal: Choose optimal number of clusters k

Procedure:

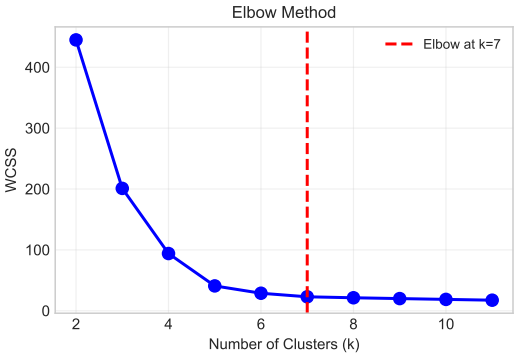
1. Run K-means for $k = 2, 3, \ldots, K_{\max}$
2. Compute Within-Cluster Sum of Squares (WCSS):

$$\text{WCSS}(k) = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

3. Plot WCSS vs k
4. Look for "elbow" where improvement diminishes

Intuition: Balance fit quality vs model complexity

Elbow Method: Visualization



Interpretation:

- **Before k=7**: Steep decrease (adding clusters helps)
- **At k=7**: Elbow (optimal)
- **After k=7**: Flat (diminishing returns)

Rule: Choose k at the elbow

Application: k=7 optimal clusters for this dataset

Elbow Method: Code

```
# Elbow Method implementation
X_scaled = StandardScaler().fit_transform(X)

# Compute WCSS for different k
wcss = []
for k in range(2, 12):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plot elbow curve
plt.plot(range(2, 12), wcss, 'bo-')
plt.axvline(x=7, color='red', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
```

Interpretation: Elbow at $k = 7$

Silhouette Analysis

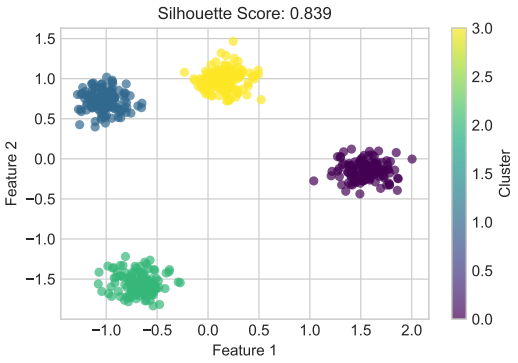
Silhouette Score: How well does point i fit its cluster?

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \in [-1, 1]$$

- $a(i)$: Mean distance to same-cluster points
- $b(i)$: Mean distance to nearest other cluster

Interpretation: $s \approx 1$ = well clustered, $s < 0$ = wrong cluster

Silhouette Score: Visualization



Result: $s = 0.713$ (good clustering)

Interpretation:

- $s > 0.7$: Strong structure
- $0.5 < s < 0.7$: Reasonable
- $0.25 < s < 0.5$: Weak
- $s < 0.25$: No structure

Application: Points with $s < 0$ are ambiguous cases

Silhouette Score: Code

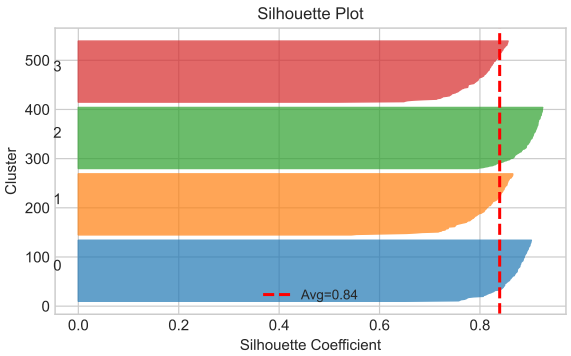
```python
from sklearn.metrics import silhouette_score

# Compute average silhouette score
s_avg = silhouette_score(X_scaled, clusters)
print(f"Silhouette Score: {s_avg:.3f}")

# Good: s > 0.7
# Reasonable: 0.5 < s < 0.7
# Weak: s < 0.5
```

Interpretation: Points with $s < 0$ are ambiguous conditions

Silhouette Plot Visualization



Reading the plot:

- **Width**: Cluster size
- **Beyond avg**: Well-separated
- **Below avg**: Poorly separated
- **Negative**: Wrong cluster

Observation: All clusters have $s > 0.5$ (good)

Application: Detect ambiguous cases

Silhouette Plot: Code

```
from sklearn.metrics import silhouette_samples

# Compute per-sample silhouette scores
s_samples = silhouette_samples(X_scaled, clusters)

# Plot silhouette for each cluster
for k in range(n_clusters):
    cluster_vals = s_samples[clusters == k]
   cluster_vals.sort()
    ax.fill_betweenx(y_range, 0, cluster_vals)

ax.axvline(x=s_avg, color='red', linestyle='--')
```

Reading: Wide bars = large clusters, $s > 0$ = well-clustered

Validation Against Ground Truth

When labels are available, compare:

Adjusted Rand Index (ARI):

$$\text{ARI} = \frac{\text{RI} - \text{Expected RI}}{\text{Max RI} - \text{Expected RI}}$$

- **Range**: $[-1, 1]$, where 1 = perfect match
- **Advantage**: Corrects for chance

```
from sklearn.metrics import adjusted_rand_score

ari = adjusted_rand_score(true_labels, clusters)
print(f"ARI: {ari:.3f}")

# ARI = 1.0: Perfect clustering
# ARI = 0.0: Random clustering
# ARI < 0.0: Worse than random
```

Case Study: K-Means on Bearing Fault Data

Case Study: CWRU Bearing Dataset

Dataset: Case Western Reserve University Bearing Data Center

Data Details:

- Vibration signals from accelerometers
- Sampling rate: 12 kHz
- Normal + various fault conditions
- MATLAB .mat format

Our Sample:

- 4 Normal bearings
- 60 Faulty bearings
- 7 extracted features
- **Highly imbalanced!**

Goal: Can K-means discover Normal vs Fault without labels?

True Labels (Supervised View)

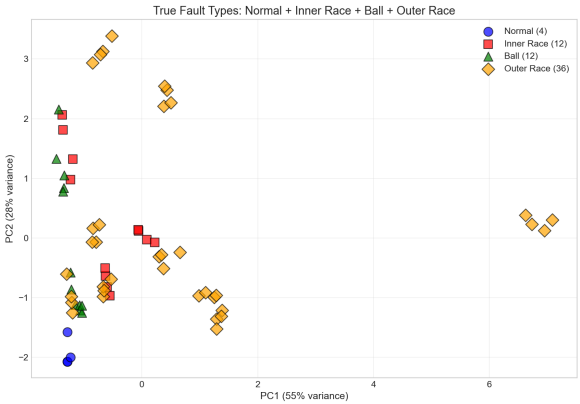


Figure 1: True Labels in PCA Space

Ground Truth (4 fault types):

- **Blue circles**: Normal (4)
- **Red squares**: Inner Race (12)
- **Green triangles**: Ball (12)
- **Orange diamonds**: Outer Race (36)

Observation:

- Normal samples cluster tightly (bottom-left)
- Fault types overlap significantly
- Rightmost points = highest vibration (severe faults)

Challenge: Can K-means find these groups?

K-Means Results (k=2)

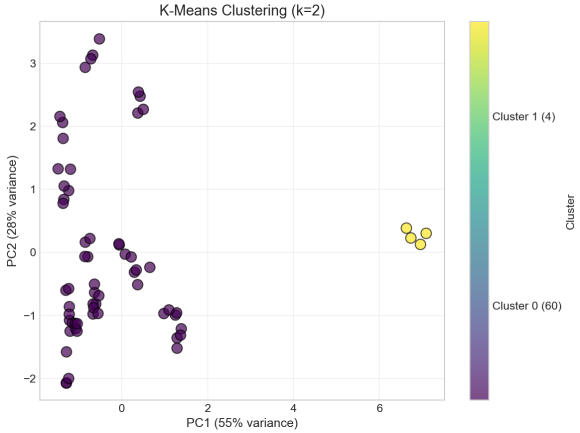


Figure 2: K-Means Clustering

K-Means Found:

- **Cluster 0** (60 samples): Normal + most faults
- **Cluster 1** (4 samples): Severe faults only

Metrics:

- Silhouette: 0.66 (good!)
- WCSS: 246.2

Problem: K-means grouped by **severity**, not by fault presence!

Why K-Means Failed

The Data Imbalance Problem:

Class	Count	Percentage
Normal	4	6.25%
Fault	60	93.75%

What K-Means saw:

- 4 extreme outliers (severe faults with very high RMS)
- 60 "similar" samples (normal + mild/moderate faults)

What K-Means did:

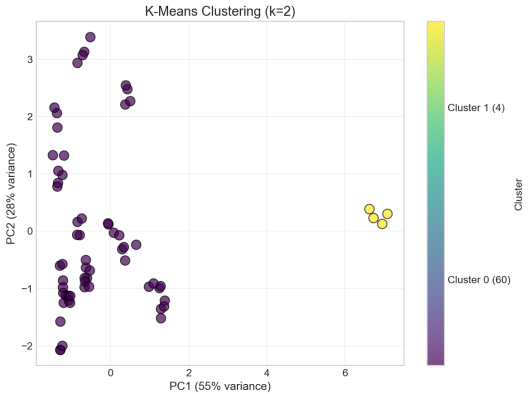
- Cluster 0: Everything with lower vibration
- Cluster 1: Only the 4 severe faults

Lesson: K-means finds density patterns, not our desired labels!

Comparison: True vs K-Means

True Labels

4 fault types (colors = fault location)



K-Means (k=2)

2 clusters (by vibration severity)

Key Insight: Clustering finds patterns that **exist in feature space**, not necessarily the patterns we want!

Lessons Learned

When K-Means Works:

- Balanced cluster sizes
- Well-separated clusters
- Spherical cluster shapes

When K-Means Struggles:

- **Imbalanced data** (like our 4 vs 60)
- Feature overlap between classes
- Non-spherical cluster shapes

Better Approaches for Fault Detection:

- **Anomaly detection**: Train on normal data, flag outliers
- **Supervised learning**: Use labels when available
- **More data**: Collect balanced training set

K-Means Limitations

Assumptions:

- Spherical clusters (assumes equal variance)
- Same cluster sizes
- Well-separated clusters

Sensitive to:

- Outliers (pull cluster centers)
- Initialization (local minima)
- Feature scaling (always standardize!)

When K-means fails:

- Non-spherical shapes
- Varying densities
- Highly imbalanced data

Other Clustering Methods

Algorithm	Key Idea	When to Use
DBSCAN	Density-based regions	Unknown k, outlier detection
Hierarchical	Build cluster tree	Explore hierarchy, dendrogram
GMM	Probabilistic Gaussians	Soft cluster assignments

Quick Summary:

- **DBSCAN**: Groups dense regions, labels outliers as noise (-1)
- **Hierarchical**: Agglomerative (bottom-up) or Divisive (top-down)
- **GMM**: Mixture of Gaussians with soft probabilities

For this course: Focus on K-means; other methods follow similar principles

Principal Component Analysis

Dimensionality Reduction Motivation

The Curse of Dimensionality:

- Modern sensor systems: 100+ features
- High-dimensional data is hard to visualize
- Computational cost grows exponentially
- Many features are correlated (redundant)

Goal: Reduce dimensions while preserving information

PCA: Find directions of maximum variance

Example: 6 feature dataset $\rightarrow$ 3 principal components

PCA: The Big Picture

Problem: Data in $\mathbb{R}^d$ (e.g., $d = 6$ sensors)

Goal: Project to $\mathbb{R}^k$ where $k < d$ (e.g., $k = 3$)

Constraint: Minimize information loss

PCA Solution: Project onto directions of maximum variance

Key Insight: Variance = information content

- High variance dimensions contain signal
- Low variance dimensions contain noise

PCA Mathematics

Given: Data matrix $X \in \mathbb{R}^{n \times d}$ (n samples, d features)

Step 1: Center the data

$$X' = X - \bar{X} \quad \text{where } \bar{X} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Step 2: Compute covariance matrix

$$\Sigma = \frac{1}{n} X'^T X' \in \mathbb{R}^{d \times d}$$

Step 3: Eigendecomposition

$$\Sigma v_j = \lambda_j v_j$$

where v_j = eigenvector (principal component), λ_j = eigenvalue

PCA: Principal Components

Eigenvectors $v_1, \ldots, v_d$:

- Directions of maximum variance
- Orthogonal to each other
- Ordered: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$

Variance Explained by PC j: $\frac{\lambda_j}{\sum_i \lambda_i}$

Projection: $Z = X'V_k$ where $V_k = [v_1, \ldots, v_k]$

Geometric Interpretation

PCA finds coordinate system aligned with data variance

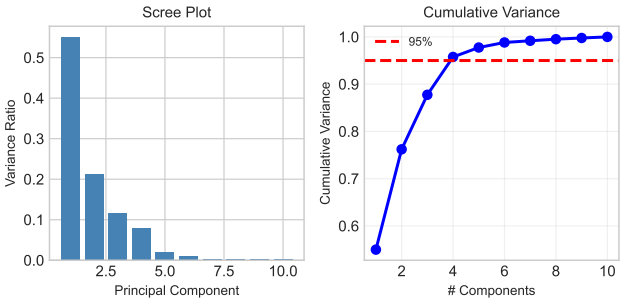
- **PC1**: Direction of maximum spread
- **PC2**: Direction of maximum spread orthogonal to PC1
- **PC3**: Direction of maximum spread orthogonal to PC1 and PC2
- ...

Projection: Rotate axes to align with principal directions

Dimensionality Reduction: Keep only top k axes

Information Loss: Variance in discarded components

Scree Plot Visualization



Interpretation:

- **PC1-3**: Capture most variance
- **PC4-6**: Diminishing returns
- **PC7-10**: Noise

Rule: Keep until 95% cumulative variance

Result: Keep 4 components (10D $\rightarrow$ 4D)

Application: 3-4 components typically capture system dynamics

Scree Plot: Code

```python
from sklearn.decomposition import PCA

# Fit PCA
pca = PCA().fit(X_scaled)
var_ratio = pca.explained_variance_ratio_

# Plot scree
plt.bar(range(1, len(var_ratio)+1), var_ratio)
plt.xlabel('Principal Component')
plt.ylabel('Variance Ratio')

# Cumulative variance
cumsum = np.cumsum(var_ratio)
n_components = np.argmax(cumsum >= 0.95) + 1
print(f"{n_components} components for 95% variance")
```

Rule: Keep components until 95% cumulative variance

Cumulative Variance

Choose number of components: Find where cumulative variance reaches threshold

```
cumsum = np.cumsum(pca.explained_variance_ratio_)
n_components_95 = np.argmax(cumsum >= 0.95) + 1
print(f"{n_components_95} components explain 95% of variance")
```

Typical Result: 3-4 components for 95% variance

Application: Often 3-4 components capture 95% of system dynamics

Connection to Spectral Analysis

Eigenanalysis Review:

Spectral Decomposition:

$$A\phi_j = \lambda_j \phi_j$$

- ϕ_j: Eigenvector
- λ_j: Eigenvalue

PCA:

$$\Sigma v_j = \lambda_j v_j$$

- v_j: Principal component
- λ_j: Variance

Insight: PCA finds "data modes" via eigenanalysis

Spectral Decomposition Analogy

Generic Spectral Analysis	PCA
Eigenvector ϕ_j	Principal component v_j
Eigenvalue λ_j	Variance λ_j
Spectral contribution	Variance explained
Reduced-order model	Dimensionality reduction
Neglect small eigenvalues	Neglect low-variance components

Both: Find directions that capture system behavior efficiently

Insight: PCA finds "data modes" via eigenanalysis

PCA Applications

Feature Reduction

Problem: System has 6 correlated features

- feature_1
- feature_2
- feature_3
- feature_4
- feature_5
- feature_6

Question: Can we reduce to 3 features without losing information?

Solution: Apply PCA, keep top 3 components

PCA Implementation

```
# Full feature set
features = ['feature_1', 'feature_2', 'feature_3', 'feature_4',
            'feature_5', 'feature_6']
X = data[features].values

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

# PCA to 3 components
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

print(f"Original dimensions: {X_scaled.shape[1]}")
print(f"Reduced dimensions: {X_pca.shape[1]}")
print(f"Variance preserved: {pca.explained_variance_ratio_.sum():.1%}")
```

Result: 6D $\rightarrow$ 3D, preserving 95%+ variance

2D PCA Visualization

Project to 2D and visualize with cluster colors:

```
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X_scaled)

plt.figure(figsize=(10, 8))
for label in data['label'].unique():
    mask = data['label'] == label
    plt.scatter(X_pca_2d[mask, 0], X_pca_2d[mask, 1],
                label=label, alpha=0.6, s=30)
plt.xlabel(f'PC1 ({pca_2d.explained_variance_ratio_[0]:.1%})')
plt.ylabel(f'PC2 ({pca_2d.explained_variance_ratio_[1]:.1%})')
plt.legend()
```

Observation: Classes naturally separate in PC space!

Feature Interpretation: Loadings

Loadings: Weights of original features on components

```
loadings = pd.DataFrame(
    pca_2d.components_.T,
    columns=['PC1', 'PC2'],
    index=features
)
print(loadings.round(3))
```

Example Interpretation:

Feature	PC1	PC2	Interpretation
feature_1	0.52	-0.18	PC1 = primary axis
feature_2	0.48	-0.22	PC1 = primary axis
feature_3	0.31	0.58	PC2 = secondary axis
feature_5	0.27	0.64	PC2 = secondary axis

Preserving Classification Performance

Critical Question: Does PCA preserve classification capability?

Approach:

1. Train classifier on **full features**
2. Train classifier on **PCA-reduced features**
3. Compare accuracy

Preserving Classification Performance (contd.)

```
from sklearn.linear_model import LogisticRegression

# Full feature classifier
clf_full = LogisticRegression()
clf_full.fit(X_scaled, fault_labels)
acc_full = clf_full.score(X_test_scaled, fault_test)

# PCA-reduced classifier
X_pca = pca.fit_transform(X_scaled)
clf_pca = LogisticRegression()
clf_pca.fit(X_pca, fault_labels)
acc_pca = clf_pca.score(X_test_pca, fault_test)

print(f"Full: {acc_full:.3f} | PCA: {acc_pca:.3f}")
```

Goal: Minimal accuracy loss with major dimension reduction

Information Loss Analysis

Reconstruction Error: Measure information lost in dimensionality reduction

```
# Reconstruct from reduced representation
X_reconstructed = pca.inverse_transform(X_pca)

# Overall error
recon_error = np.mean((X_scaled - X_reconstructed)**2)
print(f"MSE: {recon_error:.4f}")

# Per-feature error
for i, feature in enumerate(features):
    feature_error = np.mean((X_scaled[:, i] -
                             X_reconstructed[:, i])**2)
    print(f"{feature}: {feature_error:.4f}")
```

Decision:

- Accept error if **critical features** preserved
- Monitor reconstruction error for important features
- Higher tolerance for redundant features

PCA for Noise Filtering

Insight: Noise projects onto low-variance components

Denoising Procedure:

1. Apply PCA to noisy data
2. Keep only high-variance components
3. Reconstruct from reduced representation

Signal is in high-variance components, noise in low-variance

```
# Denoise by keeping top k components
pca_denoise = PCA(n_components=3)
X_denoised = pca_denoise.inverse_transform(
    pca_denoise.fit_transform(X_noisy)
)
```

Application: Clean noisy sensor data

Integration: PCA + Clustering

Why Combine PCA and Clustering?

Motivation:

- High-dimensional data is computationally expensive
- Correlated features confuse clustering algorithms
- Visualization impossible in >3 dimensions

Solution Pipeline:

1. Apply PCA: d dimensions $\rightarrow$ k dimensions
2. Apply clustering on reduced space
3. Interpret results in original space

Benefits: Faster, more stable, visualizable

Implementation Pipeline

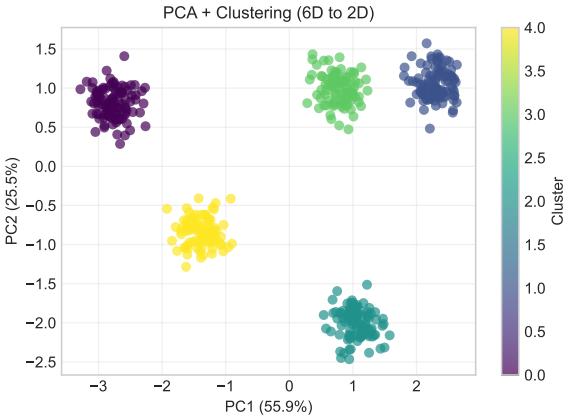
```python
# Step 1: PCA
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

# Step 2: Cluster
kmeans = KMeans(n_clusters=7, random_state=42)
clusters = kmeans.fit_predict(X_pca)

# Step 3: Validate
from sklearn.metrics import silhouette_score
print(f"Silhouette: {silhouette_score(X_pca, clusters):.3f}")
```

Result: 6D $\rightarrow$ 3D, then K-means clustering

Visualization in PC Space



Pipeline:

1. Start: 6D data
2. PCA: Reduce to 2D
3. K-means: Cluster in 2D
4. Visualize: See all clusters

Advantage: Visualize high-dimensional clustering!

Application: View classes in 2D PC space

PCA + Clustering: Code

```
# Pipeline: PCA then clustering
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Cluster in reduced space
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(X_pca)

# Visualize
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters)
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%})')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%})')
```

Advantage: Visualize high-dimensional clustering in 2D!

Parameter Sensitivity

How do choices affect results?

```
for n_comp in [2, 3, 4, 5]:
    X_pca = PCA(n_components=n_comp).fit_transform(X_scaled)
    clusters = KMeans(n_clusters=7).fit_predict(X_pca)
    ari = adjusted_rand_score(true_labels, clusters)
    print(f"Components: {n_comp}, ARI: {ari:.3f}")
```

Observation: Performance plateaus after 3-4 components

Clustering Algorithm Comparison

```
algorithms = {
    'K-means': KMeans(n_clusters=7),
    'DBSCAN': DBSCAN(eps=0.5, min_samples=10),
    'GMM': GaussianMixture(n_components=7)
}
for name, model in algorithms.items():
    clusters = model.fit_predict(X_pca)
    print(f"{name}: ARI={adjusted_rand_score(y, clusters):.3f}")
```

Result: K-means often best for structured data classification

Real-World Case Study

Scenario: Large dataset with 10,000 samples, 12 features

Goal: Identify operational patterns for optimization

Approach:

1. **Data collection**: 10,000 samples $\times$ 12 features $\times$ 7200 timesteps $=$ 864M data points
2. **PCA**: Reduce 12D $\rightarrow$ 4D (preserving 96% variance)
3. **Clustering**: K-means with $k = 5$ operational modes
4. **Analysis**: Characterize each mode by context variables
5. **Optimization**: Recommend best practices from efficient clusters

Result: 2-3% efficiency improvement $=$ significant cost savings

Supervised vs Unsupervised

The Complete ML Taxonomy

Method	Type	Input	Output	Example
Regression	Supervised	X, y continuous	$\hat{y} \in \mathbb{R}$	Predict drag coefficient
Classification	Supervised	X, y categorical	$\hat{y} \in \{0, 1, \ldots, K-1\}$	Identify fault type
Clustering	Unsupervised	X only	Cluster assignments	Discover patterns
PCA	Unsupervised	X only	Reduced X'	Reduce features

Decision Flowchart

Labeled Data? → Supervised

- Continuous output → Regression
- Categorical output → Classification

No Labels? → Unsupervised

- Find groups → Clustering
- Reduce dimensions → PCA

Hybrid: Use clustering to create labels, then train classifier

When to Use Each

Regression: Continuous predictions
Classification: Categorical decisions

Clustering: Discover patterns
PCA: Reduce dimensions

Key Question: Do you have labels?

- Yes $\rightarrow$ Supervised (Regression/Classification)
- No $\rightarrow$ Unsupervised (Clustering/PCA)

Wrap-Up

Key Takeaways

Classification vs Clustering:

- Classification uses labels; Clustering discovers labels
- Both partition data, different paradigms

Clustering Algorithms:

- K-means: Fast, requires k, spherical clusters
- DBSCAN: Finds outliers, arbitrary shapes, needs ε
- Hierarchical: Explores all k, dendrogram
- GMM: Probabilistic, soft assignments

Validation:

- Elbow method, silhouette score (internal)
- ARI, NMI (external, when labels available)
- Always check domain interpretability

Key Takeaways (continued)

PCA:

- Projects to directions of maximum variance
- Connection to modal analysis (eigenvectors!)
- Choose k to preserve 95% variance

Integration:

- PCA before clustering: faster, more stable
- Safety-aware reduction: preserve classification performance
- Visualization: cluster in 3D PC space

Application Context:

- Unsupervised learning discovers unknown failure modes
- PCA reduces feature redundancy
- Always validate physical interpretability

Homework 5 Preview

Assignment: Data Clustering & PCA Analysis

Part 1: Clustering (35 points):

- Implement K-means with elbow method selection
- Apply DBSCAN for anomaly detection
- Hierarchical clustering with dendrogram
- Validate against ground truth labels

Part 2: PCA (35 points):

- Full PCA with scree plot analysis
- Reduce 6D $\rightarrow$ 3D feature space
- Classification-aware validation (preserve performance)
- Interpret principal components physically

Homework 5 Preview (continued)

Part 3: Integration (30 points):

- PCA + Clustering pipeline
- Compare performance: full vs reduced space
- Visualize 3D clustering results
- Technical writeup with domain interpretation

Bonus (10 points):

- Gaussian Mixture Model implementation
- Advanced visualization (interactive 3D plots)
- Novel application

Due: Next week

Dataset: Synthetic multi-sensor data (provided)

Next Week: Week 6

Coming Up: Advanced Dimensionality Reduction

Topics:

- Proper Orthogonal Decomposition (POD)
- Connection between PCA and POD
- t-SNE for visualization
- Autoencoders (neural network approach)
- Applications to CFD data compression

Bridge to Deep Learning: Autoencoders lead to Week 7 (Neural Networks)

Questions & Discussion

Open Floor:

- Clustering vs classification confusion?
- PCA mathematics clarification?
- Applications?
- Homework questions?

Office Hours: See syllabus

Resources:

- Bishop PRML Chapter 9 (Mixture Models)
- Bishop PRML Chapter 12 (PCA)
- Class notes on Canvas

Thank You!

AERO 689: Machine Learning for Aerospace Engineers

Dr. Raktim Bhattacharya
Texas A&M University

Next Lecture: Week 6 - Advanced Dimensionality Reduction