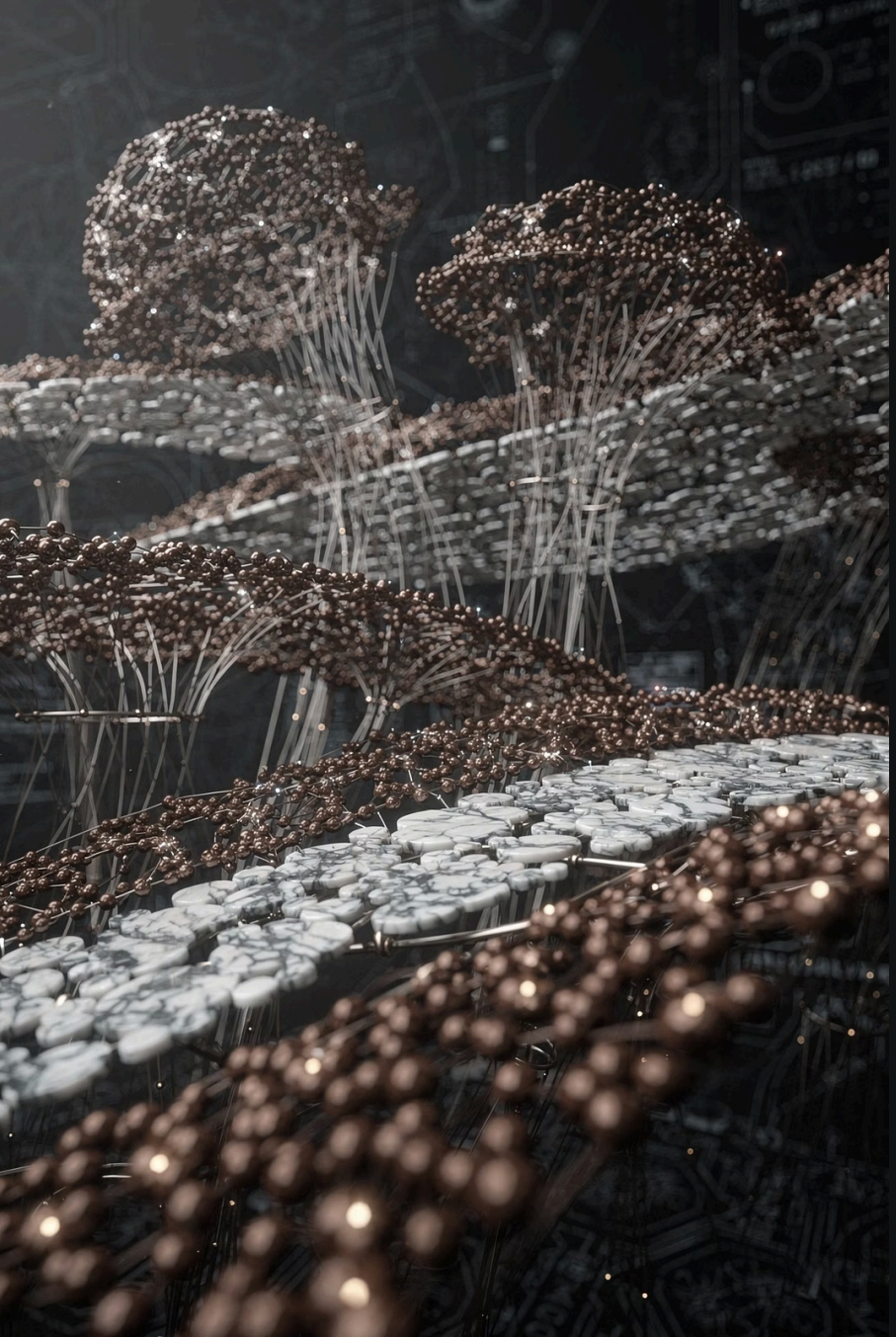


Unsupervised Learning Algorithms

Complete Diploma Notes for Machine Learning

MSBTE K-Scheme | Course Code: 316316 | Semester 6

What is Unsupervised Learning?

Unsupervised learning is a type of machine learning where algorithms learn patterns from unlabeled data without human supervision. Unlike supervised learning, there's no target variable to predict—the algorithm discovers hidden structures on its own.

This approach is particularly valuable for exploratory data analysis, customer segmentation, anomaly detection, and dimensionality reduction tasks where labeled data is unavailable or expensive to obtain.

No Labels

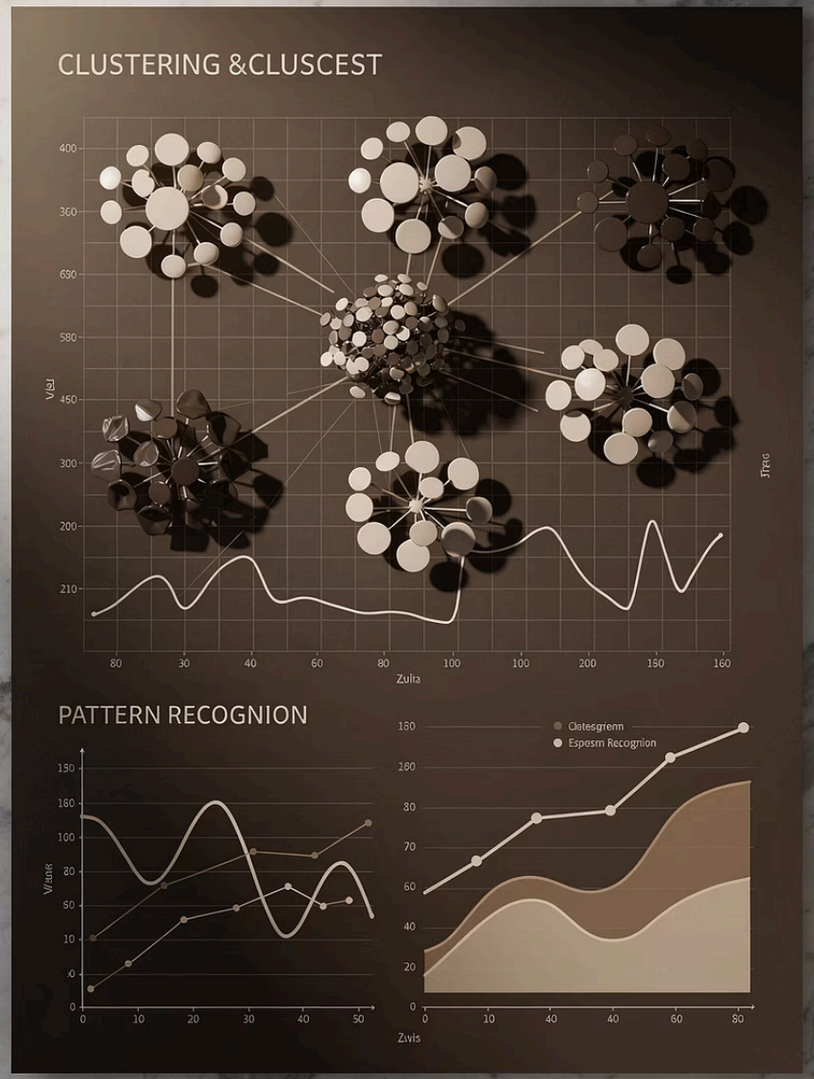
Works with unlabeled data

Pattern Discovery

Finds hidden structures

Exploratory

Understands data distribution



Types of Unsupervised Learning

Clustering

Groups similar data points together based on their characteristics. Common algorithms include K-Means, Hierarchical Clustering, and DBSCAN.

Dimensionality Reduction

Reduces the number of features while preserving important information. Techniques include PCA, t-SNE, and Autoencoders.

Association Rules

Discovers relationships between variables in large datasets. Used in market basket analysis with algorithms like Apriori.

Real-World Applications

1 Customer Segmentation

Businesses use clustering to group customers by purchasing behavior, demographics, and preferences for targeted marketing campaigns.

2 Anomaly Detection

Identify unusual patterns in network traffic, financial transactions, or manufacturing processes to detect fraud or defects.

3 Recommendation Systems

Platforms like Netflix and Amazon use unsupervised learning to discover user preferences and suggest relevant content.

4 Image Compression

Reduce image file sizes while maintaining visual quality using dimensionality reduction techniques like PCA.

K-Means Clustering

K-Means is one of the most popular clustering algorithms that partitions data into K clusters based on similarity. It aims to minimize the within-cluster sum of squares (WCSS), also known as inertia.

The algorithm works iteratively by assigning points to the nearest centroid and updating centroids based on cluster means until convergence is reached.

How K-Means Works

1 Initialize Centroids

Choose K random points from the dataset as initial cluster centroids

2 Assign Clusters

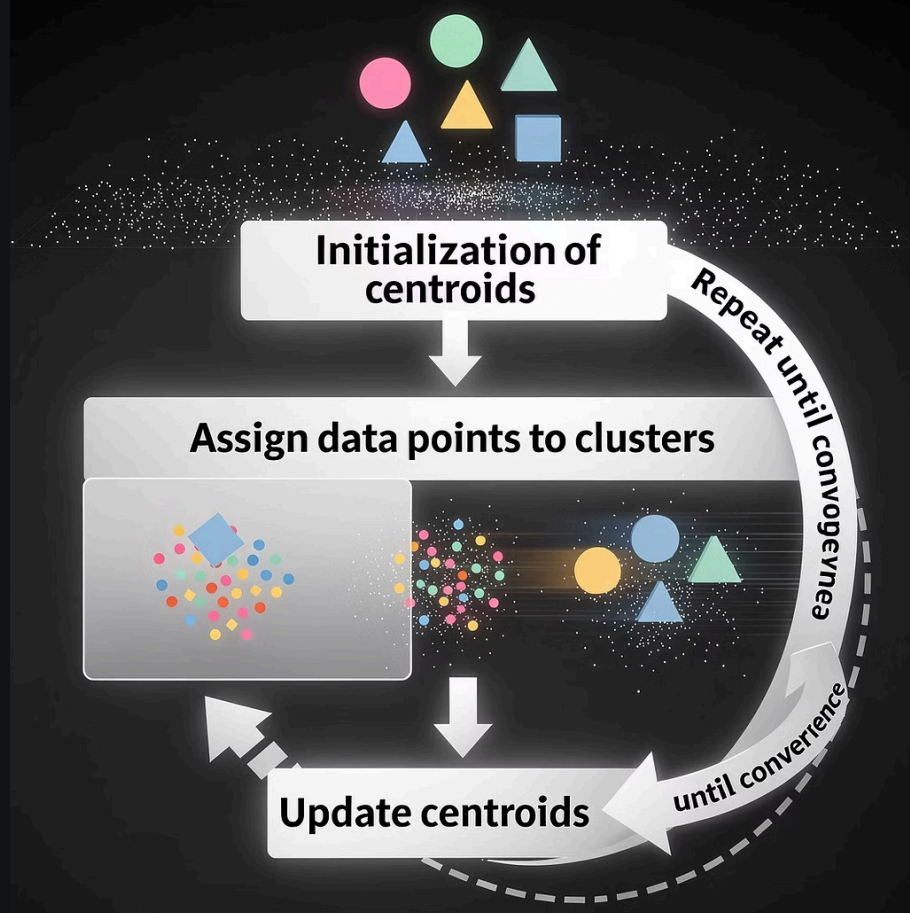
Assign each data point to the nearest centroid based on Euclidean distance

3 Update Centroids

Calculate new centroids as the mean of all points in each cluster

4 Repeat Until Convergence

Continue steps 2-3 until centroids stabilize or maximum iterations reached



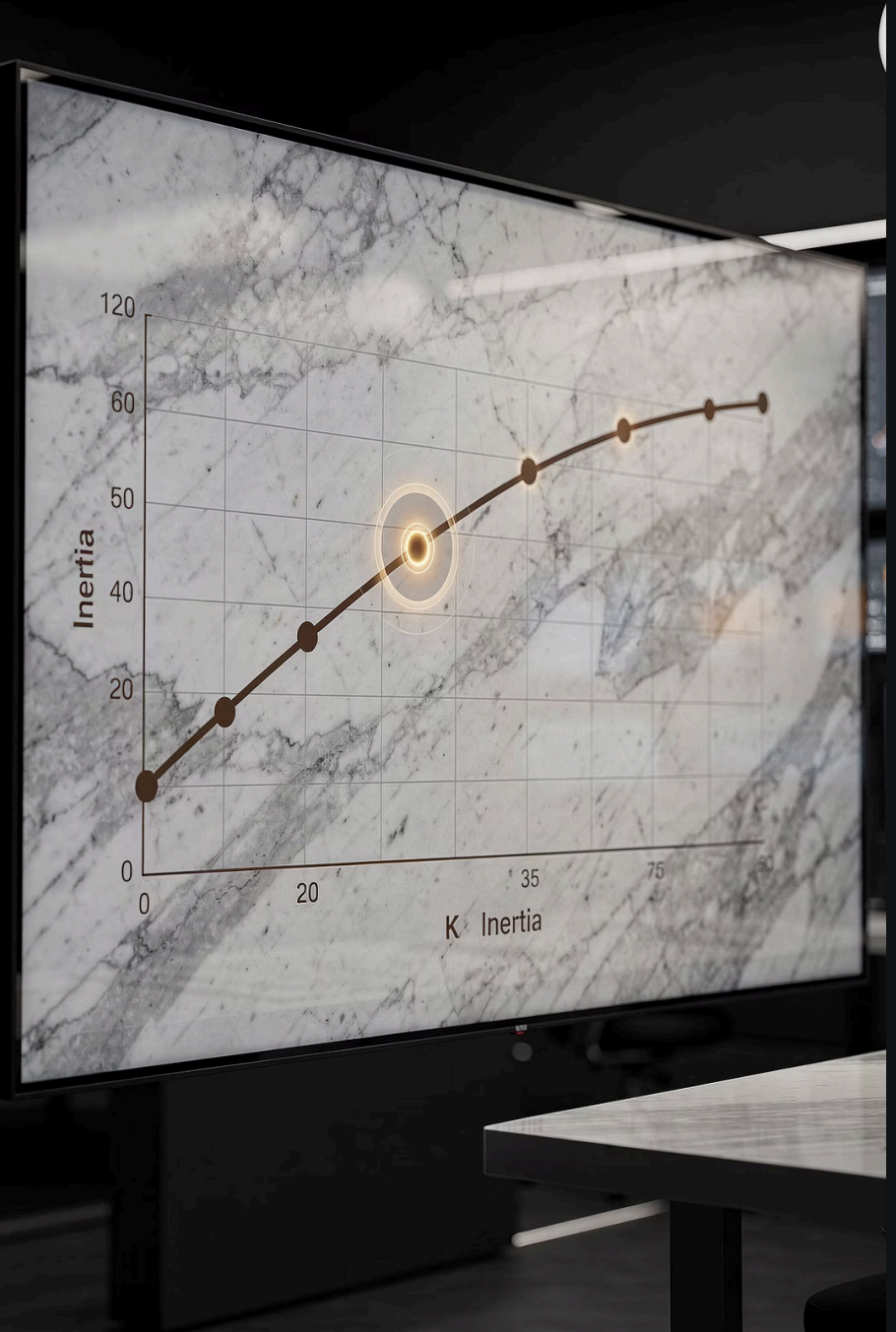
K-Means Objective Function

K-Means minimizes the Within-Cluster Sum of Squares (WCSS):

$$WCSS = \sum_{j=1}^{K} \sum_{i \in C_j} ||x_i - \mu_j||^2$$

Where:

- **xi**: Individual data points
- **μj**: Cluster centroids
- **K**: Number of clusters
- **Cj**: Set of points in cluster j



Choosing the Optimal K Value

1 Elbow Method

Plot WCSS versus K and look for the "elbow" point where WCSS decreases slowly. This indicates the optimal number of clusters.

2 Silhouette Analysis

Measures how similar points are to their own cluster versus other clusters. Values range from -1 to +1, with higher values indicating better clustering.

3 Gap Statistics

Compares within-cluster dispersion to a reference distribution. Choose K where the gap statistic is maximum.

K-Means Python Implementation

Here's a complete example of implementing K-Means clustering using scikit-learn:

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate sample data
X, y_true = make_blobs(n_samples=300, centers=4,
 cluster_std=0.60, random_state=42)

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

# Plot results
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0],
 kmeans.cluster_centers_[:, 1],
 s=300, c='red', marker='X', label='Centroids')
plt.legend()
plt.show()

print("Inertia (WCSS):", kmeans.inertia_)
```

Hierarchical Clustering

Hierarchical clustering builds a hierarchy of clusters by either merging smaller clusters (agglomerative) or splitting larger ones (divisive). Unlike K-Means, it doesn't require specifying the number of clusters beforehand.

The result is a tree-like structure called a dendrogram that shows the relationships between clusters at different levels of granularity.

Agglomerative vs Divisive Clustering

Agglomerative (Bottom-Up)

- Start with each point as a single cluster
- Find the closest pair of clusters
- Merge them into one cluster
- Repeat until desired number of clusters

Divisive (Top-Down)

- Start with all points in one cluster
- Find the cluster to split
- Split it into two clusters
- Repeat until desired number of clusters

Linkage Methods in Hierarchical Clustering

1 Single Linkage

Distance between closest points of clusters. Tends to create long, chain-like clusters. Good for elongated shapes but sensitive to noise.

2 Complete Linkage

Distance between farthest points of clusters. Creates compact, spherical clusters but sensitive to outliers.

3 Average Linkage

Average distance between all pairs of points. Balanced approach between single and complete linkage methods.

4 Ward's Method

Minimizes within-cluster variance. Similar to K-Means objective, produces compact clusters.

Understanding Dendrograms

A dendrogram is a tree-like diagram that shows the hierarchical relationships between clusters. It's a powerful visualization tool for understanding cluster structure.

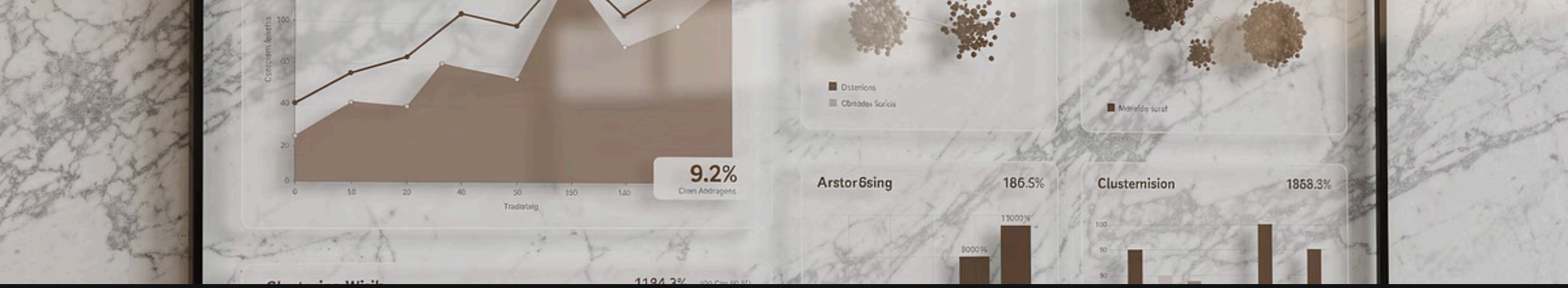
Reading a Dendrogram:

- Height of merge indicates similarity/distance
- Cutting a horizontal line gives clusters
- Y-axis shows distance or similarity measure
- Branches show cluster relationships

Hierarchical Clustering Implementation

Python code for agglomerative clustering with dendrogram visualization:

```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Generate sample data
X, y_true = make_blobs(n_samples=150, centers=3,
 cluster_std=0.60, random_state=42)

# Apply hierarchical clustering
hierarchical = AgglomerativeClustering(n_clusters=3,
 linkage='ward')
y_hierarchical = hierarchical.fit_predict(X)

# Create and plot dendrogram
linkage_matrix = linkage(X, method='ward')
plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, truncate_mode='level', p=5)
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.title('Hierarchical Clustering Dendrogram')
plt.show()
```



Clustering Evaluation Metrics

Evaluating clustering quality is crucial for understanding algorithm performance. We use two types of metrics: internal metrics (when true labels are unknown) and external metrics (when true labels are available).

Internal Evaluation Metrics

1 **WCSS (Inertia)**

Measures cluster compactness. Lower values indicate tighter, more cohesive clusters.

$$WCSS = \sum\sum ||x_i -$$
$$\mu_j||^2$$

2 **Calinski-Harabasz Index**

Ratio of between-cluster to within-cluster dispersion. Higher values indicate better clustering.

$$CH = \frac{BCSS/(k-1)}{WCSS/(n-k)}$$

3 **Davies-Bouldin Index**

Average similarity between clusters. Lower values indicate better separation between clusters.

Silhouette Analysis

The silhouette coefficient measures how similar a point is to its own cluster compared to other clusters:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Where:

- **a(i)**: Average distance to points in same cluster
- **b(i)**: Average distance to points in nearest cluster

Interpretation:

- s(i) ≈ 1: Well-clustered point
- s(i) ≈ 0: On cluster boundary
- s(i) ≈ -1: Possibly wrong cluster

External Evaluation Metrics

1. **Perfect Score**
   Adjusted Rand Index at maximum
2. **Good Score**
   Adjusted Mutual Information threshold
3. **Excellent**
   V-Measure for homogeneity

External metrics compare predicted clusters to true labels. Adjusted Rand Index (ARI) measures similarity between clusterings, while Adjusted Mutual Information (AMI) measures agreement. V-Measure combines homogeneity and completeness scores.

Evaluation Metrics Implementation

```
from sklearn.metrics import (silhouette_score,
 calinski_harabasz_score, adjusted_rand_score,
 adjusted_mutual_info_score, v_measure_score)

# Internal metrics (no true labels needed)
silhouette = silhouette_score(X, y_pred)
ch_score = calinski_harabasz_score(X, y_pred)
inertia = kmeans.inertia_

print("=== Internal Metrics ===")
print(f"Silhouette Score: {silhouette:.3f}")
print(f"Calinski-Harabasz: {ch_score:.3f}")
print(f"Inertia (WCSS): {inertia:.3f}")

# External metrics (when true labels available)
ari = adjusted_rand_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
v_measure = v_measure_score(y_true, y_pred)

print("\n=== External Metrics ===")
print(f"Adjusted Rand Index: {ari:.3f}")
print(f"Adjusted Mutual Info: {ami:.3f}")
print(f"V-Measure: {v_measure:.3f}")
```

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a powerful dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while preserving as much variance as possible.

PCA identifies principal components—directions of maximum variance in the data—and projects the data onto these components.

Mathematical Foundation of PCA

1

Covariance Matrix

Calculate covariance between features

$$C = \frac{1}{n} X^T X$$

2

Eigenvalue Decomposition

Find eigenvalues and eigenvectors

$$Cv = \lambda v$$

3

Explained Variance

Proportion of variance captured

$$\frac{\lambda_i}{\sum \lambda_j}$$

PCA Implementation Steps

1. **Standardize Data**

   Scale features to mean=0, variance=1 to ensure equal contribution

2. **Compute Covariance Matrix**

   Calculate relationships between features

3. **Calculate Eigenvalues**

   Find principal component directions and magnitudes

4. **Sort Components**

   Order by eigenvalues (descending)

5. **Select Top K**

   Choose components capturing desired variance

6. **Transform Data**

   Project onto new basis vectors

PCA Python Implementation

Complete example of PCA for dimensionality reduction:

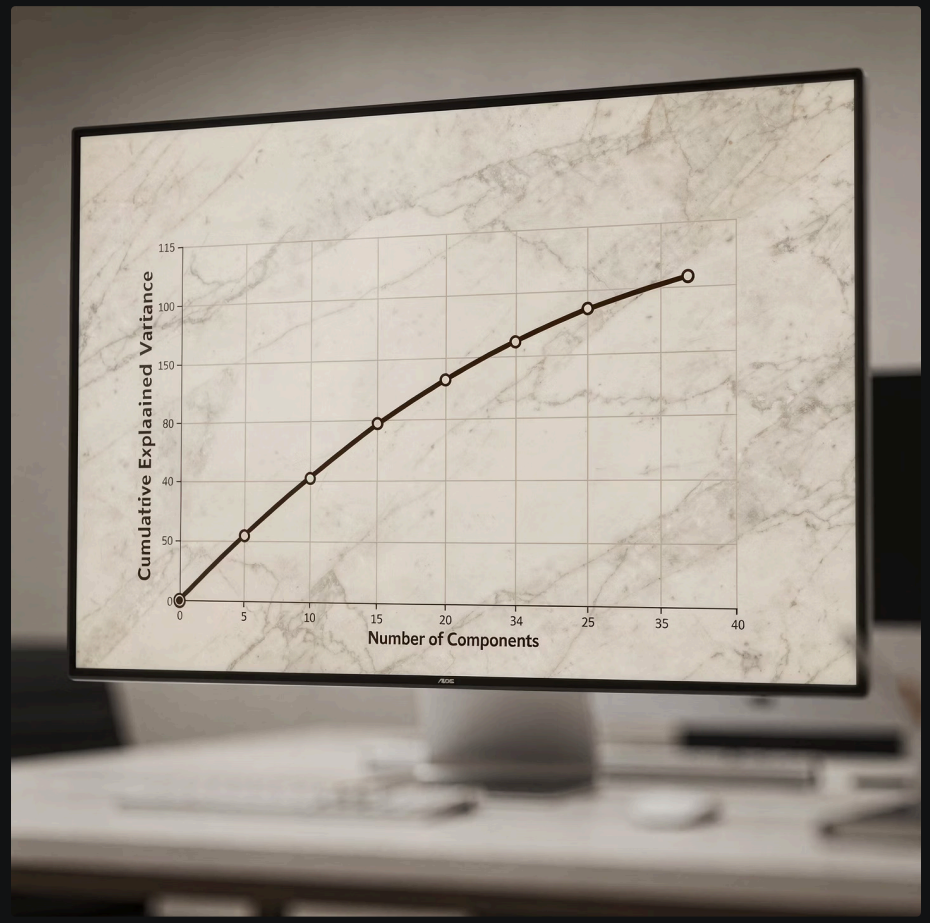
```
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Load digits dataset (64 features)
digits = load_digits()
X = digits.data
y = digits.target

print("Original shape:", X.shape) # (1797, 64)

# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

print("PCA shape:", X_pca.shape) # (1797, 2)
print("Explained variance:", pca.explained_variance_ratio_)
print("Total variance:", sum(pca.explained_variance_ratio_))

# Visualize results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1],
 c=y, cmap='tab10', alpha=0.7)
plt.colorbar(scatter)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA of Digits Dataset')
plt.show()
```

Choosing Number of Components

Determining the optimal number of principal components is crucial for balancing dimensionality reduction with information preservation.

Common Approaches:

- **Cumulative Variance:** Choose components that explain 95% of variance
- **Elbow Method:** Look for elbow in scree plot
- **Kaiser Criterion:** Keep components with eigenvalue > 1
- **Domain Knowledge:** Based on application requirements



PCA Applications

Image Compression

Reduce image storage requirements while maintaining visual quality by keeping only top principal components.

Data Visualization

Project high-dimensional data to 2D or 3D for visualization and pattern discovery.

Noise Reduction

Remove noise by eliminating components with low variance that often represent random fluctuations.

Ethics in Machine Learning

As machine learning systems become increasingly integrated into society, ethical considerations are paramount. Responsible AI development requires addressing bias, ensuring fairness, protecting privacy, and maintaining transparency.

Ethical ML practices build trust, prevent harm, and ensure technology benefits all members of society equitably.

Bias and Fairness in ML

1 **Selection Bias**

Training data doesn't represent the real-world population, leading to skewed predictions for underrepresented groups.

2 **Confirmation Bias**

Models reinforce existing beliefs or stereotypes present in historical data, perpetuating discrimination.

3 **Algorithmic Bias**

Model design or optimization choices inadvertently favor certain groups over others in outcomes.

Fairness metrics include demographic parity (equal positive outcomes across groups), equal opportunity (equal true positive rates), and equalized odds (equal TPR and FPR across groups).

Privacy and Data Protection

Key Privacy Concerns:

- **Data Collection:** Informed consent and transparency about data usage
- **Data Storage:** Secure storage and appropriate retention policies
- **Data Usage:** Purpose limitation and minimal data collection
- **Data Sharing:** Controlled access and anonymization

Regulatory Frameworks:

- **GDPR:** General Data Protection Regulation (EU)
- **CCPA:** California Consumer Privacy Act
- **HIPAA:** Health Insurance Portability Act
- **Data Minimization:** Collect only necessary data

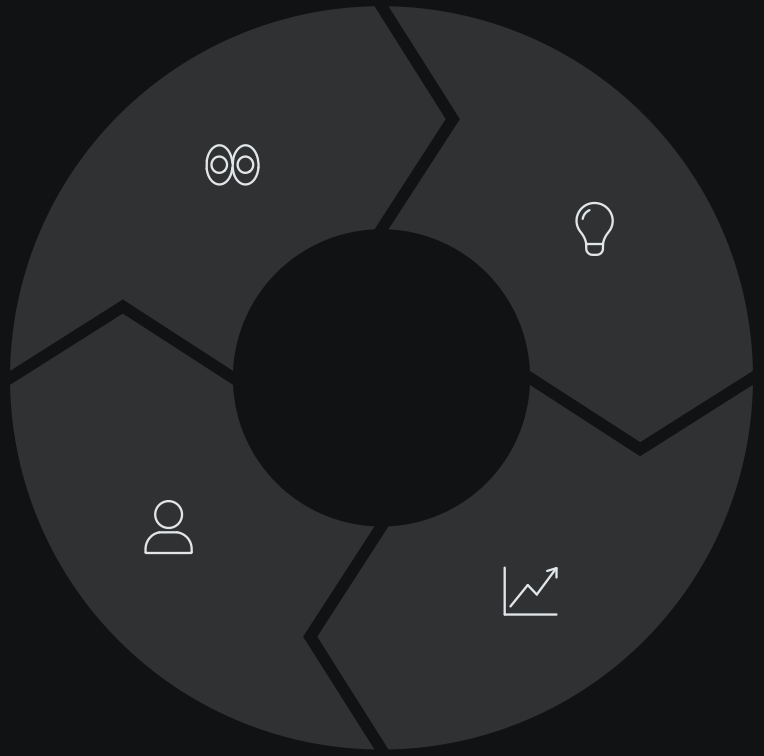
Transparency and Explainability

Model Transparency

Clear documentation of model architecture and training process

User Understanding

Communicating results in accessible language



Explainable AI

Techniques like SHAP and LIME to interpret predictions

Feature Importance

Understanding which features drive model decisions

Model Deployment and Production

Deploying machine learning models to production requires careful planning, robust infrastructure, and ongoing monitoring. The deployment process transforms research code into reliable, scalable systems that serve real users.

Model Serialization

Using Pickle:

```
import pickle

# Save model
with open('model.pkl', 'wb') as f:
    pickle.dump(model, f)

# Load model
with open('model.pkl', 'rb') as f:
    loaded_model = pickle.load(f)
```

Using Joblib (Recommended):

```
from joblib import dump, load

# Save model
dump(model, 'model.joblib')

# Load model
loaded_model = load('model.joblib')
```

Joblib is preferred for large models as it's more efficient with NumPy arrays and provides better compression.

Creating ML APIs with Flask

Flask enables easy creation of REST APIs for model serving:

```
from flask import Flask, request, jsonify
import joblib
import numpy as np

app = Flask(__name__)
model = joblib.load('model.joblib')

@app.route('/predict', methods=['POST'])
def predict():
 try:
 # Get data from request
 data = request.get_json()
 features = np.array(data['features']).reshape(1, -1)

 # Make prediction
 prediction = model.predict(features)

 # Return result
 return jsonify({
 'prediction': prediction.tolist(),
 'status': 'success'
 })
 except Exception as e:
 return jsonify({
 'error': str(e),
 'status': 'error'
 })

if __name__ == '__main__':
 app.run(debug=True, host='0.0.0.0', port=5000)
```

Monitoring and Maintenance

1 **Performance Monitoring**

Track accuracy, latency, and throughput metrics continuously

2 **Data Drift Detection**

Monitor input distribution changes that affect predictions

3 **Model Retraining**

Periodic updates with new data to maintain performance

4 **Version Control**

Track model versions and enable rollback if needed

Key Takeaways

1 **K-Means Clustering**

Simple and efficient for spherical clusters. Use elbow method or silhouette analysis to choose K. Best for large datasets with clear separation.

2 **Hierarchical Clustering**

Builds cluster hierarchy without predefined K. Dendrograms visualize relationships. Better for small datasets and exploratory analysis.

3 **PCA**

Powerful dimensionality reduction preserving variance. Essential for visualization and preprocessing. Choose components explaining 95% variance.

4 **Ethics & Deployment**

Address bias, ensure fairness, protect privacy. Serialize models, create APIs, monitor performance. Responsible AI is crucial.

Exam Preparation Guide

Short Answer Topics:

- Define unsupervised learning with examples
- Explain K-Means algorithm steps
- Describe elbow method for choosing K
- Compare agglomerative vs divisive clustering
- Explain silhouette coefficient interpretation
- Define PCA and its applications
- Discuss ethical concerns in ML

Practical Questions:

- Implement K-Means on iris dataset
- Create dendrogram for hierarchical clustering
- Apply PCA for dimensionality reduction
- Evaluate clustering using silhouette analysis
- Serialize and deploy ML model
- Calculate WCSS and explained variance

Remember: Practice coding examples, understand mathematical foundations, and focus on real-world applications. Master evaluation metrics and ethical considerations for comprehensive understanding.