Inf620 2025 Lecture Unsupervised Learning and Cluster: Kmeans, DBscan

Depto de Informática - UFV

Ricardo Ferreira
ricardo@ufv.br

2025

Introduction

- Class Material (click here for the Colab)
- **Review**: Supervised Learning
- **Problems**: Classification and Regression
- TODAY's Lesson: Unsupervised Learning
  - Kmeans
  - DBscan
  - Hierarchical

Supervised Learning

- Training data has known labels
- The goal is to learn a function that maps inputs to their corresponding labels
- Techniques:
  - Naive Bayes, KNN
  - Decision Trees, Random Forest, and Boosting
  - Linear Regression and Logistic Regression
  - Neural Networks
  - Support Vector Machines (SVM)
- Applications: Classification, Regression

KNN (K-Nearest Neighbors)

- Instance-based learning algorithm
- Classifies a new instance based on the k nearest neighbors
- Parameters:
  - Number of neighbors (k)
  - Distance metric (Euclidean, Manhattan, etc.)
- Advantages:
  - Simple to implement and interpret
  - Effective for classification and regression problems
- Disadvantages:
  - Sensitive to data dimensionality (Curse of Dimensionality)
  - Can be slow for large datasets

Naive Bayes

- Probabilistic learning algorithm
- Based on Bayes' Theorem to calculate the probability of a class given the features
- Assumes independence between features
- Parameters:
  - Type of probability distribution (Gaussian, Multinomial, etc.)
- Advantages:
  - Easy to implement and fast to train
  - Works well even with small data samples
  - Robust to noise in data
- Disadvantages:
  - Assuming feature independence may be an oversimplification
  - Sensitive to high-cardinality variables

Decision Tree

- Tree model-based learning algorithm
- Recursively splits the feature space into smaller regions
- Parameters:
  - Splitting criterion (Gini, Entropy, etc.)
  - Maximum tree depth
  - Minimum leaf node size
- Advantages:
  - Model interpretability
  - Can handle categorical and numerical variables
  - Efficient in terms of processing and memory
- Disadvantages:
  - Prone to overfitting
  - Sensitive to noise and outliers in data

Random Forest and Boosting

- Random Forest:
  - Ensemble of decision trees
  - Each tree is trained on a random subset of data and features
  - Combines predictions from multiple trees to achieve better results
  - Resilient to overfitting, efficient with high-dimensional data
- Boosting:
  - Builds an additive predictive model by sequentially adding weak models
  - Each weak model is trained to put more weight on misclassified examples
  - Popular algorithms: AdaBoost, Gradient Boosting, XGBoost
  - Achieves high accuracy but can be more susceptible to overfitting

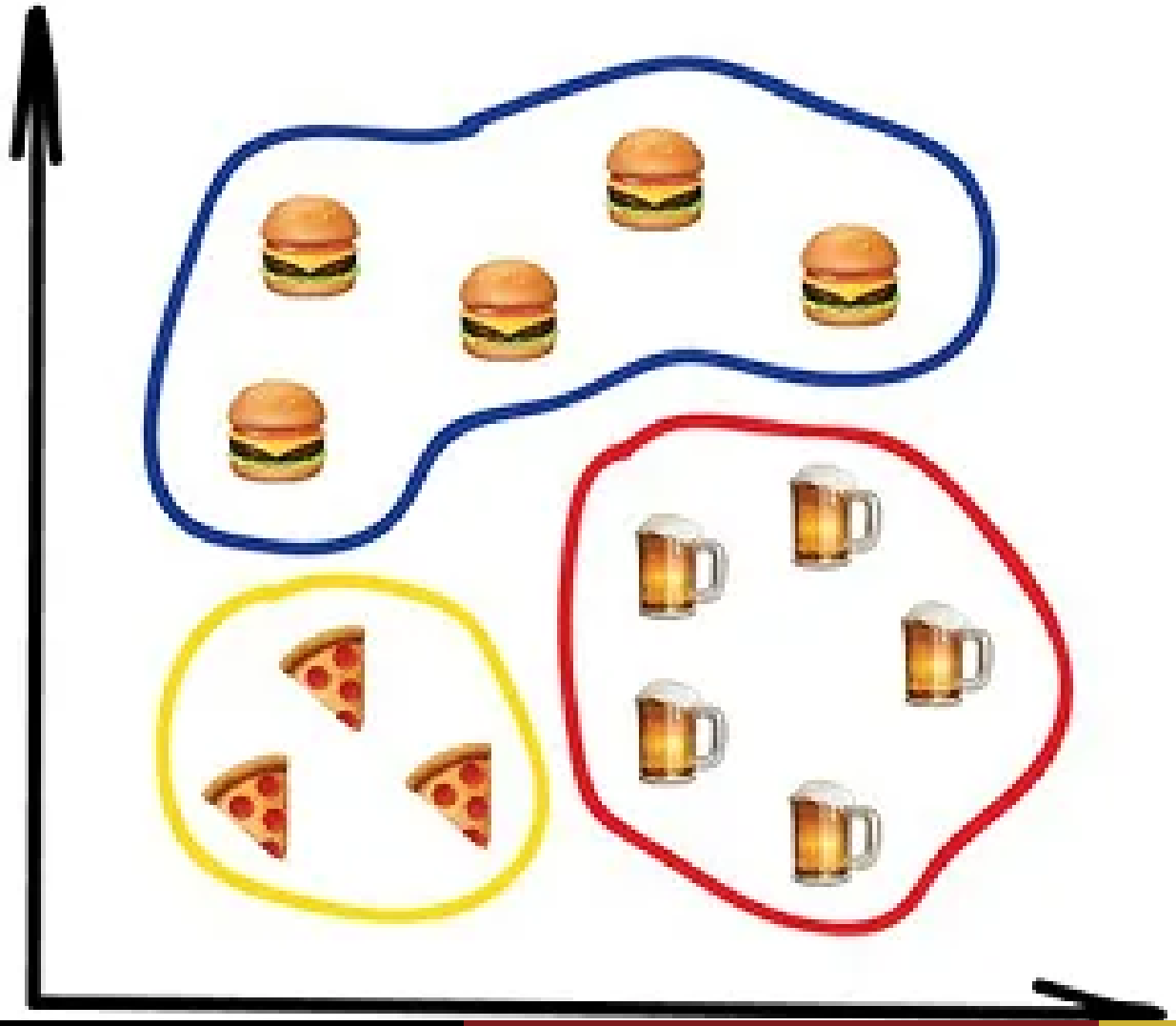
Unsupervised Learning

- Training data has no known labels
- The goal is to find patterns and inherent structures in the data
- Techniques:
  - Clustering
  - Principal Component Analysis (PCA)
  - Singular Value Decomposition (SVD)
  - Autoencoders
- Applications: Market Segmentation, Anomaly Detection, Dimensionality Reduction

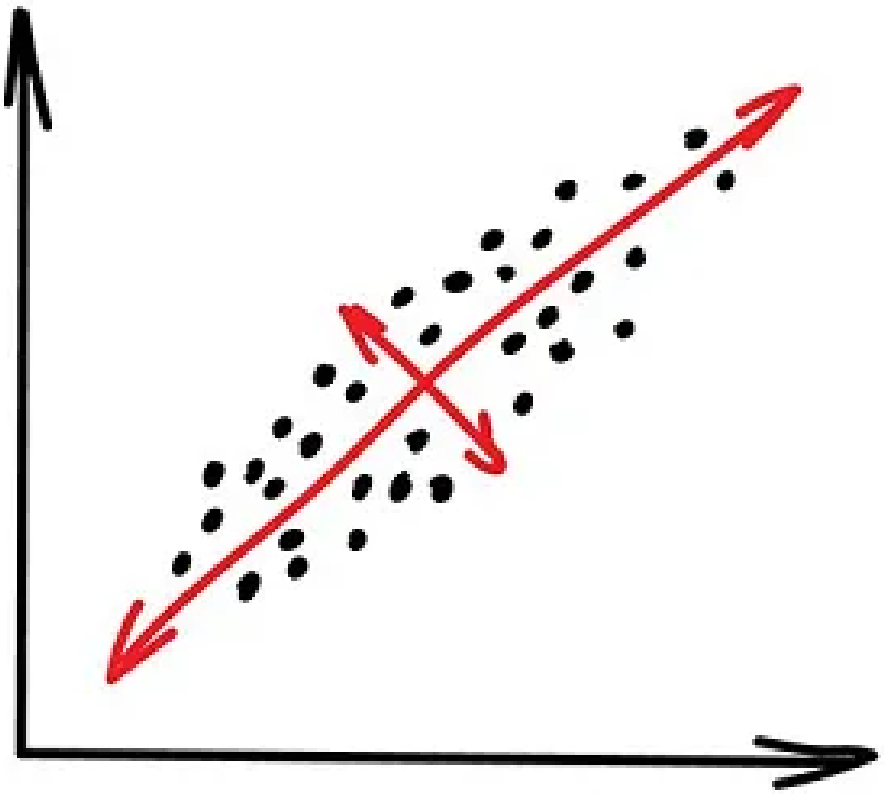
Comparison

	Supervised	Unsupervised
Labels	Known	Unknown
Objective	Learn mapping input-output	Find patterns and structures
Techniques	Regression Classification	Clustering, Dimensionality Reduction
Applications	Prediction	Segmentation, Anomaly Detection

Clustering



Dimensionality Reduction



Redução de dimensão

Click here for Blog access with more details

What is K-means?

- Unsupervised clustering algorithm
- Objective: group similar data
- Divides n observations into k groups
- Main characteristics:
  - Simple and efficient
  - Widely used
  - Centroid-based

What is K-means?

- Unsupervised clustering algorithm
- Objective: group similar data
- Divides n observations into k groups
- Main characteristics:
  - Simple and efficient
  - Widely used
  - Centroid-based

How does it work?

1. **Initialization**
   - Random choice of k centroids
2. **Assignment**
   - Each point is associated with the nearest centroid
3. **Update**
   - Recalculation of centroids
   - Mean of points in each cluster
4. Repeat steps 2 and 3 until convergence

How does it work?

1. **Initialization**
   - Random choice of k centroids
2. **Assignment**
   - Each point is associated with the nearest centroid
3. **Update**
   - Recalculation of centroids
   - Mean of points in each cluster
4. Repeat steps 2 and 3 until convergence

How does it work?

1. **Initialization**
   - Random choice of k centroids
2. **Assignment**
   - Each point is associated with the nearest centroid
3. **Update**
   - Recalculation of centroids
   - Mean of points in each cluster
4. Repeat steps 2 and 3 until convergence

How does it work?

1. **Initialization**
   - Random choice of k centroids
2. **Assignment**
   - Each point is associated with the nearest centroid
3. **Update**
   - Recalculation of centroids
   - Mean of points in each cluster
4. Repeat steps 2 and 3 until convergence

Advantages and Limitations

Advantages

- Simple
- Fast
- Scalable

Limitations

- Requires defining k
- Sensitive to outliers
- Assumes spherical clusters

Implementation in Python

Basic Code

```
from sklearn.cluster import KMeans

# Create model
kmeans = KMeans(n_clusters=k)

# Train
kmeans.fit(X)

# Predict
labels = kmeans.predict(X)
```

Practical Applications

- **Customer Segmentation**
  - Consumer groups
  - Purchase patterns
- **Image Analysis**
  - Compression
  - Segmentation
- **Text Analysis**
  - Document clustering
  - Categorization

Practical Applications

- **Customer Segmentation**
  - Consumer groups
  - Purchase patterns
- **Image Analysis**
  - Compression
  - Segmentation
- **Text Analysis**
  - Document clustering
  - Categorization

Practical Applications

- **Customer Segmentation**
  - Consumer groups
  - Purchase patterns
- **Image Analysis**
  - Compression
  - Segmentation
- **Text Analysis**
  - Document clustering
  - Categorization

Evolution of Machine Learning Algorithms

1750 Naive Bayes

1943 Neural Network: Threshold Logic

1963 Support Vector Machine

1986 Neural Networks: Backpropagation

1987 Convolutional Neural Networks

2009 Deep Learning: ImageNet

2012 AlexNet

2016 Inception, ResNet

Fundamental Algorithms

1901 **PCA**
Principal Component Analysis - Karl Pearson

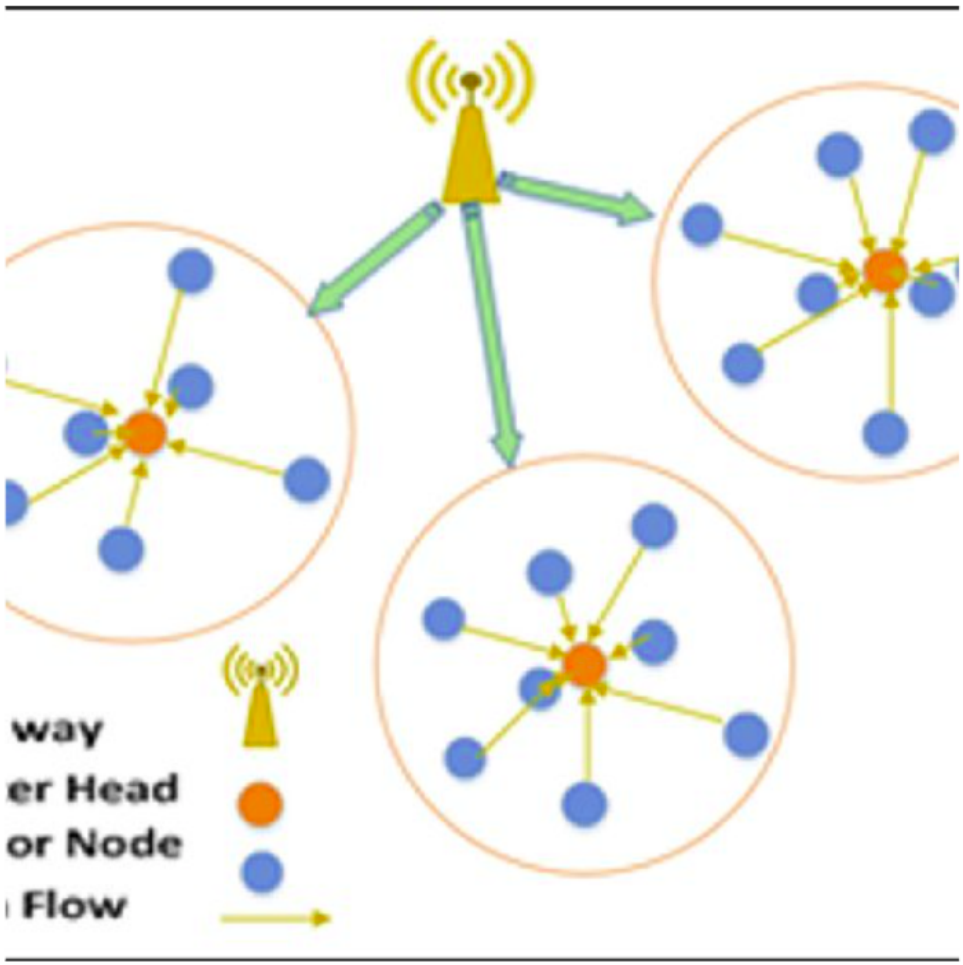
1957 K-means
Lloyd, Forgy

1996 DBSCAN
Density-Based Spatial Clustering

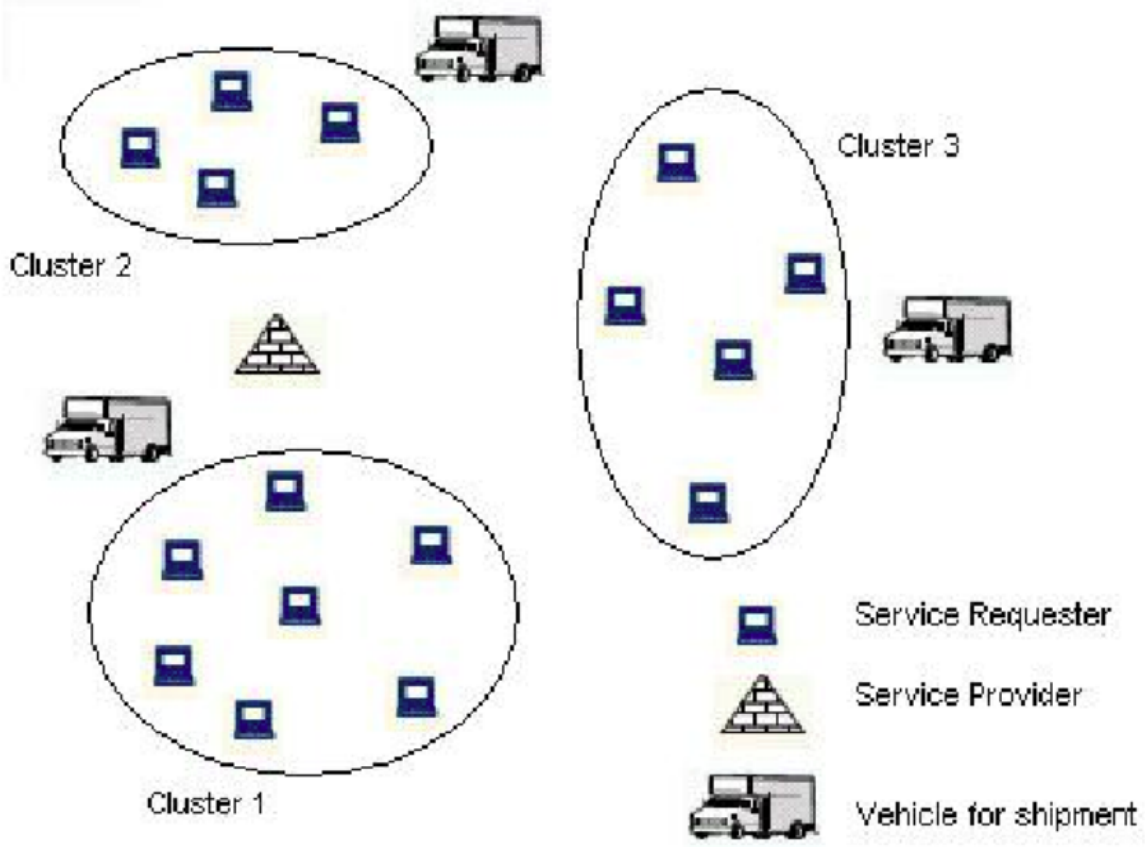
2007 K-means++
David Arthur and Sergei Vassilvitskii

Clustering and Dimensionality Reduction

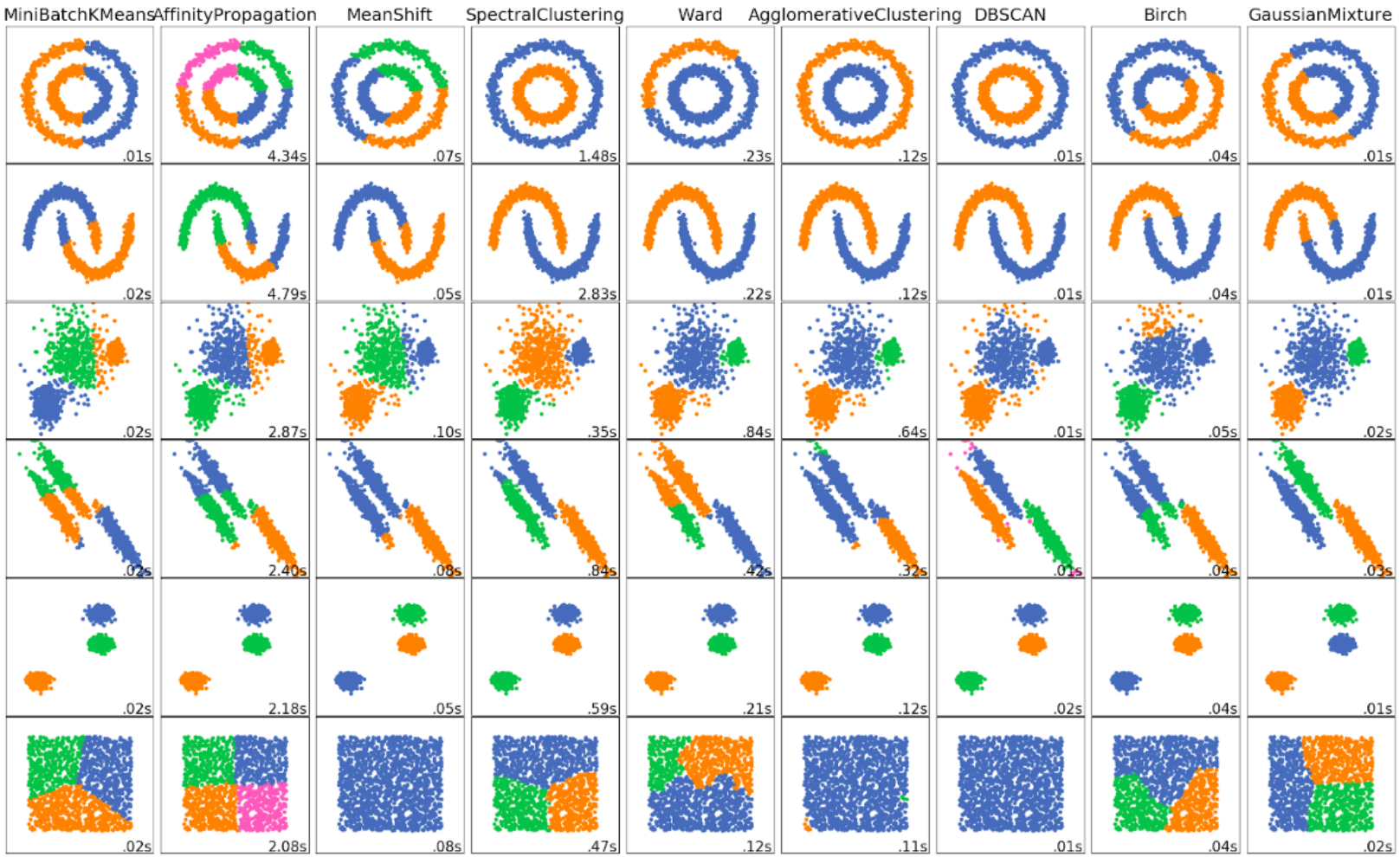
Applications



Applications



Several Methods



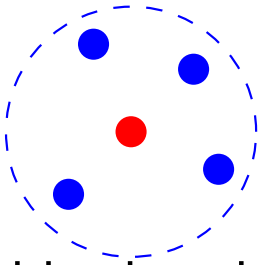
The 5 Clustering Algorithms Data Scientists Need to Know

Several Methods

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

DBSCAN: Fundamental Concepts

- **Density-Based Spatial Clustering of Applications with Noise**
- Developed in 1996 (Ester, Kriegel, Sander, Xu)
- **Main parameters:**
  - (eps): neighborhood radius
  - minPts: minimum number of points



Neighborhood with minPts = 4

- **Advantages:**
  - No need to predefine the number of clusters
  - Detects clusters of arbitrary shapes
  - Robust to outliers

DBSCAN: Types of Points

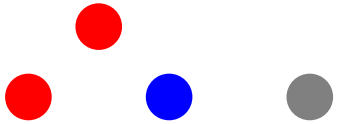
Point Classification:

1. **Core Points**
   - Have minPts in their -neighborhood
   - Form the "core"of the cluster
2. **Border Points**
   - In the neighborhood of a core point
   - Fewer than minPts neighbors
3. **Noise Points**
   - Neither core nor border points
   - Considered outliers

Core Point

Border Point

Noise Point



DBSCAN: Process and Applications

Clustering Process:

1. Identify core points
2. Connect density-reachable core points
3. Associate border points
4. Identify noise points

Complexity:

- O(n log n) with spatial index structures
- O(n^2) without optimizations

Practical Applications:

- Image segmentation
- Social network analysis
- Anomaly detection
- Recommendation systems
- Traffic analysis

Limitations:

- Sensitive to and minPts parameters
- Difficulty with clusters of very different densities

Random Forest Algorithm

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

AdaBoost Algorithm

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
ada = AdaBoostClassifier(n_estimators=100, random_state=42)
ada.fit(X_train, y_train)
y_pred = ada.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Gradient Boost Algorithm

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
gb = GradientBoostingClassifier(n_estimators=100, random_state=42
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

XGBoost Algorithm

```python
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
xgb = XGBClassifier(n_estimators=100, random_state=42)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

DBSCAN Algorithm

```python
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
import numpy as np

X, _ = make_moons(n_samples=200, noise=0.05, random_state=42)
X = StandardScaler().fit_transform(X)
dbscan = DBSCAN(eps=0.3, min_samples=5)
clusters = dbscan.fit_predict(X)

n_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise = list(clusters).count(-1)
print(f'Estimated number of clusters: {n_clusters}')
print(f'Number of noise points: {n_noise}')
print(f'Unique labels: {np.unique(clusters)}')
```

K-means Algorithm

```python
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score

X, true_labels = make_blobs(n_samples=300, centers=4, cluster_std=
kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(X)
inertia = kmeans.inertia_
silhouette = silhouette_score(X, clusters)

print(f'Inertia (sum of squares): {inertia:.2f}')
print(f'Silhouette score: {silhouette:.2f}')
print(f'Centroids:\n{kmeans.cluster_centers_}')
```

Hierarchical Clustering: Main Techniques

Approaches:

1. **Agglomerative (Bottom-up)**
   - Starts with n clusters
   - Merges the closest clusters
   - Most common in practice
2. **Divisive (Top-down)**
   - Starts with 1 cluster
   - Recursively splits
   - More computationally expensive

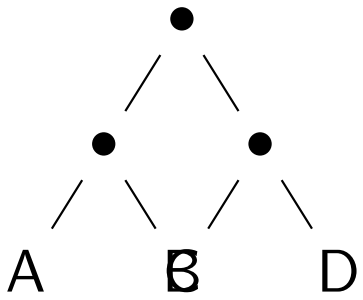
Linkage Metrics:

- **Single Linkage:** Minimum distance between points
- **Complete Linkage:** Maximum distance between points
- **Average Linkage:** Average of distances
- **Ward:** Minimizes variance within clusters

Hierarchical Clustering: Main Techniques

Approaches:

1. **Agglomerative (Bottom-up)**
   - Starts with n clusters
   - Merges the closest clusters
   - Most common in practice
2. **Divisive (Top-down)**
   - Starts with 1 cluster
   - Recursively splits
   - More computationally expensive



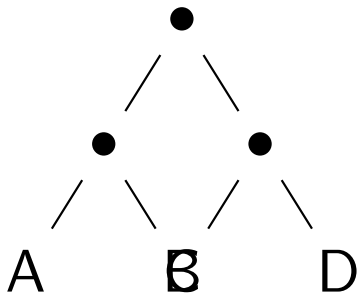
Linkage Metrics:

- **Single Linkage:** Minimum distance between points
- **Complete Linkage:** Maximum distance between points
- **Average Linkage:** Average of distances
- **Ward:** Minimizes variance within clusters

Hierarchical Clustering: Main Techniques

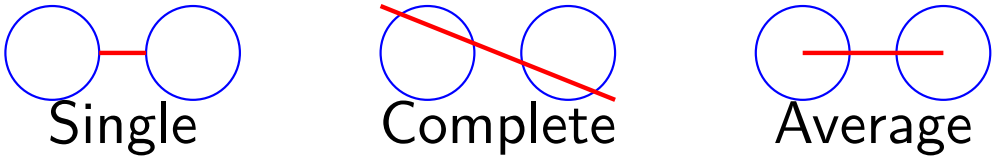
Approaches:

1. **Agglomerative (Bottom-up)**
   - Starts with n clusters
   - Merges the closest clusters
   - Most common in practice
2. **Divisive (Top-down)**
   - Starts with 1 cluster
   - Recursively splits
   - More computationally expensive



Linkage Metrics:

- **Single Linkage:** Minimum distance between points
- **Complete Linkage:** Maximum distance between points
- **Average Linkage:** Average of distances
- **Ward:** Minimizes variance within clusters



Hierarchical Clustering Implementation

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
import scipy.cluster.hierarchy as shc
import matplotlib.pyplot as plt

X, _ = make_blobs(n_samples=100, centers=3, random_state=42)
clustering = AgglomerativeClustering(n_clusters=3, linkage='ward'
clusters = clustering.fit_predict(X)

print(f'Number of clusters: {len(set(clusters))}')
print(f'Size of clusters: {[list(clusters).count(i) for i in rang
```

Hierarchical Clustering - YouTube

Very simple example, 12 minutes

K-means and Hierarchical, 17 minutes

Hierarchical Clustering and Heatmaps, Stat Quest, genetics example

K-means: Key Takeaways

Key Points:

- Most popular clustering algorithm
- Centroid-based
- Complexity: $O(t \times k \times n)$
- **When to use:**
  - Clusters of similar size
  - Spherical/convex shapes
  - Known number of clusters
  - Large volumes of data

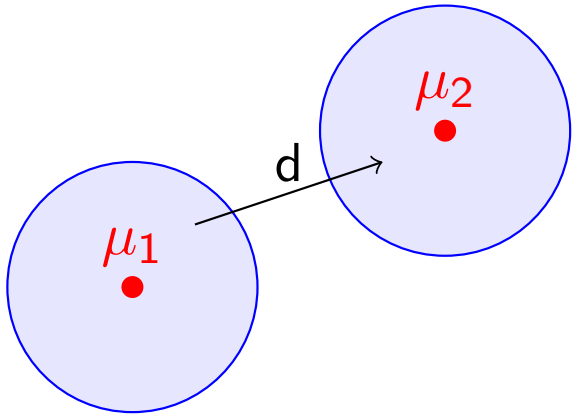
Hyperparameters:

- n_clusters (k)
- init
- max_iter

K-means: Key Takeaways

Key Points:

- Most popular clustering algorithm
- Centroid-based
- Complexity: $O(t \times k \times n)$
- **When to use:**
  - Clusters of similar size
  - Spherical/convex shapes
  - Known number of clusters
  - Large volumes of data



Hyperparameters:

- n_clusters (k)
- init
- max_iter

DBSCAN: Core Insights

Core Concepts:

1. **Types of Points:**
   - Core points
   - Border points
   - Noise points
2. **Unique Advantages:**
   - Detects outliers
   - Arbitrary-shaped clusters
   - Does not require predefined k

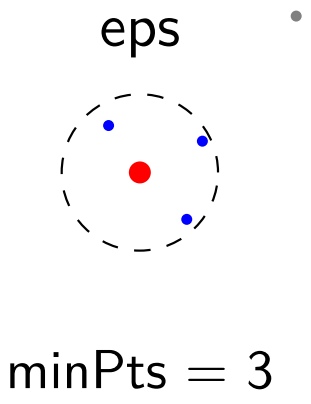
Critical Parameters:

- eps (ϵ)
- min_samples

DBSCAN: Core Insights

Core Concepts:

1. **Types of Points:**
   - Core points
   - Border points
   - Noise points
2. **Unique Advantages:**
   - Detects outliers
   - Arbitrary-shaped clusters
   - Does not require predefined k



Critical Parameters:

- eps (ϵ)
- min_samples

Unsupervised Learning: Overview

Main Applications:

- Customer segmentation
- Anomaly detection
- Dimensionality reduction
- Exploratory analysis

Algorithm Selection:

- K-means: well-separated, defined clusters
- DBSCAN: noisy data and irregular shapes
- Hierarchical: when hierarchical structure is needed

Final Considerations:

- Validation
- Visualization
- Interpretability

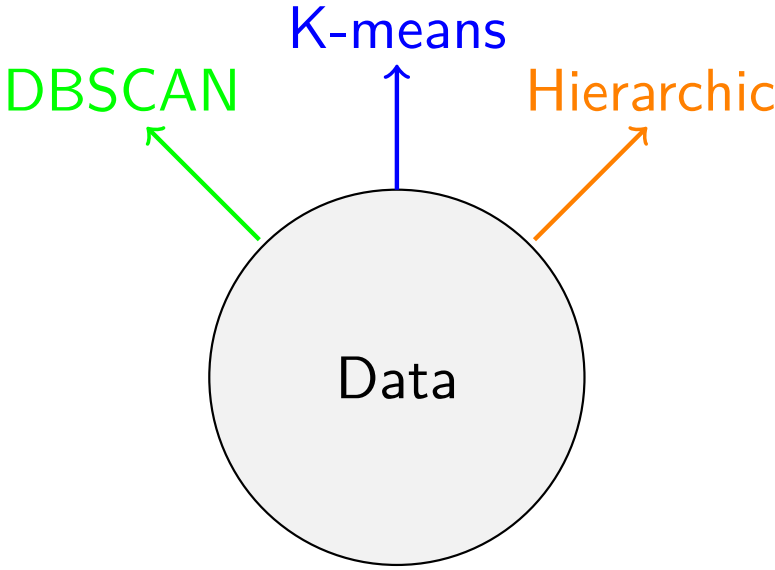
Unsupervised Learning: Overview

Main Applications:

- Customer segmentation
- Anomaly detection
- Dimensionality reduction
- Exploratory analysis

Algorithm Selection:

- K-means: well-separated, defined clusters
- DBSCAN: noisy data and irregular shapes
- Hierarchical: when hierarchical structure is needed



Final Considerations:

- Validation
- Visualization
- Interpretability

What is t-SNE?

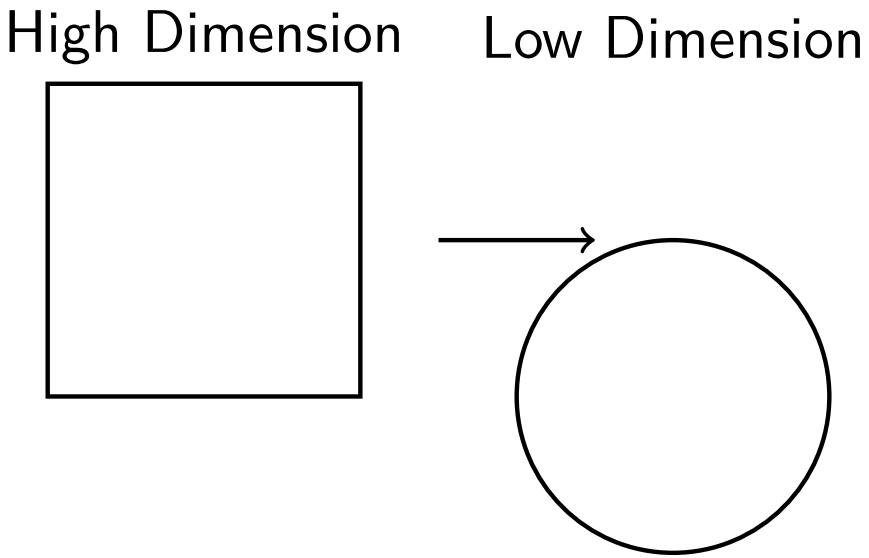
- **t-SNE** (t-Distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique
- **Main goal:** Visualize high-dimensional data in 2D or 3D
- **Key characteristics:**
  - Preserves local data structures
  - Reveals natural clusters
  - Maintains neighborhood relationships

What is t-SNE?

- **t-SNE** (t-Distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique
- **Main goal:** Visualize high-dimensional data in 2D or 3D
- **Key characteristics:**
  - Preserves local data structures
  - Reveals natural clusters
  - Maintains neighborhood relationships

What is t-SNE?

- **t-SNE** (t-Distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique
- **Main goal:** Visualize high-dimensional data in 2D or 3D
- **Key characteristics:**
  - Preserves local data structures
  - Reveals natural clusters
  - Maintains neighborhood relationships



How Does t-SNE Work?

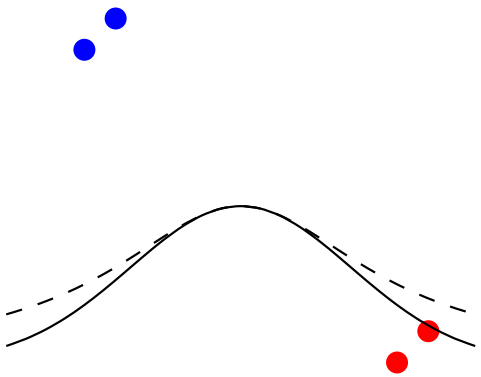
Two-step process:

1. **Similarities in original space**
   - Uses Gaussian distribution
   - Calculates neighborhood probabilities
2. **Similarities in reduced space**
   - Uses t-Student distribution
   - Minimizes KL divergence

How Does t-SNE Work?

Two-step process:

1. **Similarities in original space**
   - Uses Gaussian distribution
   - Calculates neighborhood probabilities
2. **Similarities in reduced space**
   - Uses t-Student distribution
   - Minimizes KL divergence



Advantages and Applications

Advantages:

- Better cluster separation
- Handles outliers well
- Preserves non-linear structures

Applications:

- Visualization of genomic data
- Image analysis
- Natural language processing
- Deep learning

Advantages and Applications

Advantages:

- Better cluster separation
- Handles outliers well
- Preserves non-linear structures

Applications:

- Visualization of genomic data
- Image analysis
- Natural language processing
- Deep learning