High-Dimensional Data: Classification and Clustering

Techniques for Categorizing High-Dimensional Data — Part 2 of 2

Prof. Michael Mascagni

Department of Computer Science
Department of Mathematics
Department of Scientific Computing
Graduate Program in Molecular Biophysics
Florida State University, Tallahassee, FL 32306 **USA**
E-mail: mascagni@fsu.edu
URL: http://www.cs.fsu.edu/∼mascagni

Outline

Review and Roadmap

k-Nearest Neighbors (k-NN)

Support Vector Machines (SVM)

Random Forests

k-Means and Gaussian Mixture Models

DBSCAN and Spectral Clustering

Evaluation Metrics

End-to-End Pipeline and Summary

Part 1 Recap and Part 2 Roadmap

Part 1 covered (dimensionality reduction):

- Curse of dimensionality — why raw high-d space is problematic
- **PCA / SVD** — linear, variance-maximizing projections
- **LDA** — supervised, class-separating projection
- **t-SNE / UMAP** — nonlinear manifold visualization
- **Autoencoders** — learned nonlinear bottleneck representations

Part 2 covers (categorization algorithms):

1. **Supervised classification:** k-NN, SVM with kernels, Random Forests
2. **Unsupervised clustering:** k-Means, DBSCAN, Gaussian Mixture Models
3. **Graph-based:** Spectral Clustering
4. **Evaluation:** Metrics for both supervised and unsupervised settings
5. **Pipeline:** Combining reduction + classification end-to-end

Unifying Theme

All methods must confront high dimensionality explicitly — either via assumptions, kernels, ensemble averaging, or density estimation.

k-NN: Algorithm and High-Dimensional Behavior

Algorithm

Given a query $\mathbf{x}^*$, find its k nearest neighbors in the training set (by Euclidean distance) and assign the majority class label.

Decision rule:

$$\hat{y}(\mathbf{x}^*) = \arg\max_c \sum_{i \in \mathcal{N}_k(\mathbf{x}^*)} \mathbf{1}[y_i = c]$$

Theoretical guarantee (Cover & Hart, 1967): As $n \to \infty$, 1-NN error rate $\leq 2\varepsilon^*$ where ε^* is the Bayes error rate.

Curse of dimensionality impact on k-NN:

- Distance concentration: $\frac{d_{\max} - d_{\min}}{d_{\min}} \to 0$
- To maintain k neighbors within radius r, need $n \propto r^{-d}$ samples — exponential in d
- **Practical rule:** Apply PCA or feature selection before k-NN when $d > 20$

k-NN: Distance Metrics for High-Dimensional Data

Minkowski distance (L_p norm):

$$d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$$

Metric	p	**High-d behavior**	**Use case**
Euclidean	2	Concentrates; all pts equidistant	Dense, continuous data
Manhattan	1	More robust to concentration	Sparse, mixed data
Chebyshev	inf	Dominated by worst feature	Rarely preferred
Cosine	–	Angle-based; scale-invariant	Text, embeddings
Mahalanobis	–	Decorrelates features	Correlated features

Mahalanobis distance:

$$d_M(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})}$$

Equivalent to Euclidean distance after PCA whitening.

k-NN: Python Example with Dimension Comparison

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler

X, y = load_digits(return_X_y=True)  # 64 dimensions
X = StandardScaler().fit_transform(X)

dims  = [2, 5, 10, 20, 30, 40, 64]
accs  = []
```

k-NN: Python Example with Dimension Comparison (Cont.)

```python
for d in dims:
    X_d = PCA(n_components=d).fit_transform(X)
    knn = KNeighborsClassifier(n_neighbors=5)
    score = cross_val_score(knn, X_d, y, cv=5).mean()
    accs.append(score)

plt.plot(dims, accs, 'o-', color='steelblue')
plt.xlabel('PCA Components Retained')
plt.ylabel('5-Fold CV Accuracy')
plt.title('k-NN Accuracy vs. Dimensionality (Digits)')
plt.grid(True); plt.tight_layout(); plt.show()
```

```
d= 2: 0.543    d= 5: 0.887    d=10: 0.950
d=20: 0.971    d=30: 0.975    d=40: 0.974    d=64: 0.969
# Peak accuracy near d=30; raw 64-d slightly worse (noise dims)
```

SVM: Maximum Margin Hyperplane

Goal

Find the hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ that **maximizes the margin** between two classes.

Hard-margin SVM (linearly separable):

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \; \forall i$$

Margin $= 2/\|\mathbf{w}\|$. Decision depends only on **support vectors** (points on margin boundary).

Soft-margin SVM (non-separable, parameter C):

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \quad \text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \;\; \xi_i \geq 0$$

C controls bias-variance tradeoff: large C = low bias, high variance.

SVM: The Kernel Trick

Key Insight: The SVM dual depends only on inner products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$, so we can replace them with a **kernel function** $K(\mathbf{x}_i, \mathbf{x}_j)$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

where ϕ maps to a (possibly infinite-dimensional) feature space **without explicit computation**.

Kernel	Formula	Use Case
Linear	$\mathbf{x}_i^\top \mathbf{x}_j$	Linearly separable, high-d text
Polynomial	$(\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)^p$	Interaction features
RBF (Gaussian)	$\exp(-\gamma \lVert \mathbf{x}_i - \mathbf{x}_j \rVert^2)$	Most common; general purpose
Sigmoid	$\tanh(\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)$	Neural-network-like

SVM in High Dimensions

Linear SVM is extremely effective for $d \gg n$ (e.g., text classification) because regularization via margin prevents overfitting. RBF kernel degrades in very high d — prefer linear or reduce first.

SVM: Python Example with Grid Search

```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

X, y = load_digits(return_X_y=True)

# Pipeline: scale -> PCA -> SVM
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('pca',    PCA(n_components=30)),
    ('svm',    SVC(kernel='rbf', C=10, gamma='scale'))
])
```

SVM: Python Example with Grid Search (Cont.)

```python
# Grid search over C and gamma
param_grid = {
    'svm__C':      [0.1, 1, 10, 100],
    'svm__gamma': ['scale', 'auto', 0.01, 0.001]
}
gs = GridSearchCV(pipe, param_grid, cv=5, n_jobs=-1)
gs.fit(X, y)

print(f"Best params : {gs.best_params_}")
print(f"Best CV acc : {gs.best_score_:.4f}")

# Final cross-validation
scores = cross_val_score(gs.best_estimator_, X, y, cv=5)
print(f"5-fold CV   : {scores.mean():.4f} +/- {scores.std():.4f}")
```

```
Best params : {'svm__C': 10, 'svm__gamma': 'scale'}
Best CV acc : 0.9872
5-fold CV   : 0.9878 +/- 0.0042
```

Random Forests: Ensemble of Decision Trees

Definition

A **Random Forest** is an ensemble of T decision trees, each trained on a bootstrap sample with a random subset of features at each split. Final prediction is by majority vote (classification) or averaging (regression).

Two sources of randomness:

1. **Bootstrap aggregating (bagging):** Each tree trained on n samples drawn with replacement from the training set
2. **Random feature subsets:** At each split, consider only $m = \lfloor \sqrt{d} \rfloor$ randomly chosen features

Bias-variance decomposition:

$$\text{Error} = \underbrace{\bar{\rho} \cdot \bar{s}^2}_{\text{correlation between trees}} + \frac{1 - \bar{\rho}}{T} \cdot \bar{s}^2$$

where $\bar{\rho}$ is average inter-tree correlation and $\bar{s}^2$ is average per-tree variance. More trees reduce variance; random features reduce $\bar{\rho}$.

Random Forests: Feature Importance

Mean Decrease in Impurity (MDI):

$$\text{Importance}(j) = \frac{1}{T} \sum_{t=1}^{T} \sum_{\text{nodes } v \text{ split on } j} \Delta\text{Gini}(v)$$

Measures total reduction in Gini impurity due to feature j across all trees.

Permutation Importance (more reliable):

1. Record baseline accuracy on out-of-bag (OOB) samples
2. Permute feature j values randomly; re-evaluate accuracy
3. Importance = drop in accuracy due to permutation

Why Random Forests excel in high dimensions:

- Random feature selection prevents a few strong features from dominating
- Built-in OOB estimate avoids expensive cross-validation
- Handles irrelevant features gracefully (they rarely appear at splits)
- No distance computations — immune to distance concentration

Random Forests: Python Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import cross_val_score
from sklearn.inspection import permutation_importance

X, y = load_digits(return_X_y=True)  # 64 features

rf = RandomForestClassifier(
    n_estimators=200,
    max_features='sqrt',     # sqrt(64) = 8 features/split
    oob_score=True,
    random_state=42,
    n_jobs=-1
)
rf.fit(X, y)
```

Random Forests: Python Example (Cont.)

```python
print(f"OOB Score       : {rf.oob_score_:.4f}")
scores = cross_val_score(rf, X, y, cv=5)
print(f"5-fold CV Acc   : {scores.mean():.4f} +/- {scores.std():.4f}")

# Plot MDI feature importance (64 pixel importances)
imp = rf.feature_importances_.reshape(8, 8)
plt.imshow(imp, cmap='hot', interpolation='nearest')
plt.colorbar(label='MDI Importance')
plt.title('Random Forest: Pixel Importance Heatmap')
plt.tight_layout(); plt.show()
```

```
OOB Score       : 0.9716
5-fold CV Acc   : 0.9716 +/- 0.0083
# Importance map highlights central pixels (most informative for digits)
```

k-Means Clustering

Goal

Partition n points into k clusters minimizing total within-cluster variance (inertia):

$$\mathcal{L} = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} \|\mathbf{x} - \boldsymbol{\mu}_j\|^2$$

Lloyd's Algorithm (iterative):

1. Initialize k centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ (random or k-means++)
2. **Assignment step:** Assign each point to its nearest centroid
3. **Update step:** Recompute centroids as cluster means
4. Repeat until convergence (centroid change $< \varepsilon$)

High-dimensional caveats:

- Distance concentration degrades cluster quality
- **Always reduce dimensionality first (PCA/UMAP) before k-Means**
- k-Means++ initialization reduces sensitivity to starting point
- Assumes spherical, equal-sized clusters

Gaussian Mixture Models (GMM)

Model

Data is generated by a mixture of k Gaussians:

$$p(\mathbf{x}) = \sum_{j=1}^{k} \pi_j \mathcal{N}(\mathbf{x};\, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

with mixing weights $\pi_j \geq 0$, $\sum_j \pi_j = 1$.

EM Algorithm for GMM:

- **E-step:** Compute posterior responsibilities $r_{ij} = P(z_i = j \mid \mathbf{x}_i)$
- **M-step:** Update π_j, $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ using weighted MLE

Gaussian Mixture Models (GMM) vs. k-Means

GMM vs. k-Means:

Property	**k-Means**	**GMM**
Cluster shape	Spherical only	Ellipsoidal (full $\boldsymbol{\Sigma}$)
Assignment	Hard (0/1)	Soft (probability)
Parameters	$k \cdot d$	$k(d + d^2/2 + 1)$
High-d scaling	Moderate	**Poor** (covariance blows up)

k-Means and GMM: Python Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.metrics import adjusted_rand_score

X, y = load_digits(return_X_y=True)

# Reduce to 30 dims first (critical for clustering quality)
X_pca = PCA(n_components=30).fit_transform(X)

# k-Means
km = KMeans(n_clusters=10, n_init=20,
            random_state=42)
km_labels = km.fit_predict(X_pca)
print(f"k-Means ARI    : {adjusted_rand_score(y, km_labels):.4f}")
```

k-Means and GMM: Python Example (Cont.)

```python
# GMM (diagonal covariance for high-d stability)
gmm = GaussianMixture(n_components=10,
                      covariance_type='diag',
                      n_init=5, random_state=42)
gmm_labels = gmm.fit_predict(X_pca)
print(f"GMM ARI       : {adjusted_rand_score(y, gmm_labels):.4f}")
print(f"GMM BIC       : {gmm.bic(X_pca):.1f}")

# Elbow method for k selection
inertias = [KMeans(n_clusters=k, n_init=10,
            random_state=42).fit(X_pca).inertia_
            for k in range(2, 15)]
```

```
k-Means ARI   : 0.7241
GMM ARI       : 0.7588   (diagonal cov handles high-d better)
GMM BIC       : 892341.3
```

DBSCAN: Density-Based Clustering

Definition (Ester et al., 1996)

DBSCAN groups together points that are **densely packed**, marking low-density regions as noise.

Key Concepts:

- ε-neighborhood: $N_\varepsilon(\mathbf{x}) = \{\mathbf{y} : d(\mathbf{x}, \mathbf{y}) \leq \varepsilon\}$
- **Core point:** $|N_\varepsilon(\mathbf{x})| \geq \text{MinPts}$
- **Border point:** In ε-neighborhood of a core point, but not itself a core point
- **Noise point:** Neither core nor border

DBSCAN: Density-Based Clustering (Cont.)

Property	**k-Means**	**DBSCAN**
Number of clusters	Must specify k	Discovered automatically
Cluster shape	Convex only	Arbitrary shapes
Noise handling	None	Explicit outlier class
High-d behavior	Moderate	**Poor** (ε meaningless)
Complexity	$O(nk)$	$O(n \log n)$ with index

Spectral Clustering

Core Idea

Represent data as a **weighted graph** $G = (V, W)$, then cluster by partitioning the graph using the eigenvectors of the **normalized graph Laplacian**.

Algorithm:

1. Build affinity matrix: $W_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$
2. Compute degree matrix $D_{ii} = \sum_j W_{ij}$
3. Normalized Laplacian: $\mathbf{L} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}$
4. Compute smallest k eigenvectors of $\mathbf{L}$: matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$
5. Row-normalize $\mathbf{U}$, then apply k-Means to rows

Why it works: The graph structure captures local similarity; eigenvectors reveal the cluster structure even for non-convex, non-spherical shapes. **Complexity:** $O(n^2)$ for dense graph — not scalable to very large n; use approximate/sparse methods for $n > 10{,}000$.

DBSCAN and Spectral Clustering: Python Example

```python
import numpy as np
from sklearn.cluster import DBSCAN, SpectralClustering
from sklearn.datasets import make_moons
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Two-moons dataset: non-convex clusters
X, y = make_moons(n_samples=500, noise=0.05, random_state=42)
X = StandardScaler().fit_transform(X)

# DBSCAN
db = DBSCAN(eps=0.3, min_samples=5)
db_labels = db.fit_predict(X)
print(f"DBSCAN clusters : {len(set(db_labels)) - (1 if -1 in db_labels else 0)
    }")
print(f"DBSCAN ARI      : {adjusted_rand_score(y, db_labels):.4f}")
print(f"Noise points    : {np.sum(db_labels == -1)}")
```

DBSCAN and Spectral Clustering: Python Example (Cont.)

```
1
2  # Spectral Clustering
3  sc = SpectralClustering(n_clusters=2, affinity='rbf',
4                          gamma=1.0, random_state=42)
5  sc_labels = sc.fit_predict(X)
6  print(f"Spectral ARI    : {adjusted_rand_score(y, sc_labels):.4f}")
7
8  # k-Means fails here
9  from sklearn.cluster import KMeans
10 km_labels = KMeans(n_clusters=2, random_state=42).fit_predict(X)
11 print(f"k-Means ARI     : {adjusted_rand_score(y, km_labels):.4f}")
```

```
DBSCAN clusters : 2       DBSCAN ARI : 1.0000    Noise: 0
Spectral ARI    : 1.0000
k-Means ARI     : 0.4981  # k-Means completely fails on moons
```

Supervised Evaluation Metrics

Accuracy (overall):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision, Recall, F1 (per class c):

$$P_c = \frac{TP_c}{TP_c + FP_c}, \quad R_c = \frac{TP_c}{TP_c + FN_c}, \quad F1_c = \frac{2P_cR_c}{P_c + R_c}$$

Macro vs. Weighted averaging:

- Macro: unweighted mean over classes (sensitive to minority class)
- Weighted: weighted by class frequency (reflects overall performance)

Metric	High-Dim Issue	Robust?	Use when
Accuracy	Imbalanced classes bias result	No	Balanced datasets
F1 Macro	Ignores sample size per class	Moderate	All classes matter equally
AUC-ROC	Threshold-independent	Yes	Binary, ranked outputs
MCC	Accounts for all 4 confusion terms	Yes	Imbalanced datasets

Unsupervised Evaluation Metrics

External metrics (require ground-truth labels):
Adjusted Rand Index (ARI):

$$\text{ARI} = \frac{\text{RI} - \mathbb{E}[\text{RI}]}{\max(\text{RI}) - \mathbb{E}[\text{RI}]} \in [-1, 1]$$

1.0 = perfect; 0.0 = random; negative = worse than random.
Normalized Mutual Information (NMI):

$$\text{NMI}(Y, \hat{Y}) = \frac{2\, I(Y; \hat{Y})}{H(Y) + H(\hat{Y})} \in [0, 1]$$

Internal metrics (no ground truth needed):

- **Silhouette score:** $s(i) = \frac{b(i)-a(i)}{\max(a(i),b(i))} \in [-1, 1]$ where a = mean intra-cluster distance, b = mean nearest-cluster distance
- **Davies-Bouldin index:** Lower is better; ratio of intra-cluster spread to inter-cluster distance
- **Calinski-Harabasz index:** Higher is better; ratio of between-cluster to within-cluster dispersion

Evaluation Metrics: Python Example

```python
import numpy as np
from sklearn.datasets import load_digits
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import (adjusted_rand_score,
    normalized_mutual_info_score, silhouette_score,
    davies_bouldin_score, calinski_harabasz_score,
    classification_report)
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X, y = load_digits(return_X_y=True)
X_s = StandardScaler().fit_transform(X)
X_r = PCA(n_components=30).fit_transform(X_s)
```

Evaluation Metrics: Python Example (Cont.)

```python
# --- Supervised metrics ---
X_tr, X_te, y_tr, y_te = train_test_split(
    X_r, y, test_size=0.2, random_state=42)
svm = SVC(C=10, gamma='scale').fit(X_tr, y_tr)
print(classification_report(y_te, svm.predict(X_te)))

# --- Unsupervised metrics ---
km_labels = KMeans(n_clusters=10, n_init=20,
                   random_state=42).fit_predict(X_r)
print(f"ARI   : {adjusted_rand_score(y, km_labels):.4f}")
print(f"NMI   : {normalized_mutual_info_score(y, km_labels):.4f}")
print(f"Silh.: {silhouette_score(X_r, km_labels):.4f}")
print(f"DB    : {davies_bouldin_score(X_r, km_labels):.4f}")
```

```
ARI   : 0.7241     NMI   : 0.8003
Silh.: 0.1423     DB    : 2.0841   (lower is better)
```

Building an End-to-End Pipeline

Recommended workflow for high-dimensional categorization:

1. **Preprocess:** Standardize (zero mean, unit variance)
2. **Reduce:** PCA (retain 90–95% variance) or UMAP
3. **Select algorithm:**
   - Labels available? → SVM (RBF or Linear) or Random Forest
   - No labels, convex clusters? → k-Means + elbow/silhouette
   - No labels, arbitrary shapes? → DBSCAN or Spectral Clustering
   - Soft assignment needed? → GMM (diagonal covariance)
4. **Tune hyperparameters:** Grid search with cross-validation
5. **Evaluate:** ARI/NMI (unsupervised), F1/AUC (supervised)
6. **Visualize:** UMAP or t-SNE for sanity check

scikit-learn Pipeline Example

`Pipeline([('scaler', StandardScaler()), ('pca', PCA(30)), ('clf', SVC(C=10))])` wraps everything into a single, cross-validatable object.

Complete Method Summary

Method	Type	Labels?	High-d?	Cluster Shape	Best Use
k-NN	Classify	Yes	**Reduce first**	N/A	Small, dense datasets
SVM (Linear)	Classify	Yes	**Yes**	N/A	Text, $d \gg n$
SVM (RBF)	Classify	Yes	**Reduce first**	N/A	General nonlinear
Random Forest	Classify	Yes	**Yes**	N/A	Mixed features, importance
k-Means	Cluster	No	**Reduce first**	Spherical	Large n, known k
GMM	Cluster	No	**Diag cov**	Ellipsoidal	Soft assignments
DBSCAN	Cluster	No	**Reduce first**	Arbitrary	Outlier detection
Spectral	Cluster	No	**Small n**	Arbitrary	Non-convex, graph data

Scalability Summary:

Method	Train	Predict	Scales to
k-NN (exact)	$O(1)$	$O(nd)$	$\leq$ 100k
SVM	$O(n^2 d)$	$O(n_{\text{sv}} d)$	$\leq$ 100k
Random Forest	$O(Tnd \log n)$	$O(T \log n)$	Millions
k-Means	$O(nkd \cdot I)$	$O(kd)$	Millions
DBSCAN	$O(n \log n)$	N/A	Millions
Spectral	$O(n^2 d + n^3)$	N/A	$\leq$ 10k

Key Takeaways: Both Lectures

Part 1 (Dimensionality Reduction):

- High dimensions create sparsity, distance concentration, and overfitting
- **PCA/SVD**: linear, fast, interpretable baseline
- **LDA**: supervised separation, at most $C - 1$ components
- **t-SNE/UMAP**: nonlinear visualization; not for preprocessing
- **Autoencoders**: flexible, learned representations for large data

Part 2 (Classification and Clustering):

- **k-NN**: simple but requires dimensionality reduction first
- **SVM**: maximum-margin; kernel trick for nonlinear boundaries; linear SVM scales well
- **Random Forests**: robust, scalable, built-in feature importance
- **k-Means**: fast, scalable; spherical clusters only
- **GMM**: probabilistic; soft assignments; use diagonal covariance in high d
- **DBSCAN**: arbitrary shapes, automatic k; ε hard to tune in high d
- **Spectral**: graph-based; best for non-convex; limited to small n

Golden Rule

Always reduce dimensionality before clustering. Always validate with multiple metrics. Visualize with UMAP.

References and Further Reading

- Bishop, C.M., *Pattern Recognition and Machine Learning*, Springer, 2006. Chs. 7 (SVM), 9 (GMM/EM), 12 (PCA).
- Breiman, L., "Random Forests," *Machine Learning*, 45(1), 2001.
- Cover, T., Hart, P., "Nearest Neighbor Pattern Classification," *IEEE Trans. Inf. Theory*, 13(1), 1967.
- Ester, M., et al., "A Density-Based Algorithm for Discovering Clusters," *KDD*, 1996.
- Ng, A., Jordan, M., Weiss, Y., "On Spectral Clustering," *NeurIPS*, 2001.
- Rousseeuw, P.J., "Silhouettes: A Graphical Aid to the Interpretation of Cluster Analysis," *J. Comput. Appl. Math.*, 20, 1987.
- Vapnik, V., *The Nature of Statistical Learning Theory*, 2nd ed., Springer, 1999.
- scikit-learn documentation and user guide: `scikit-learn.org/stable/user_guide.html`

Questions?

Abbreviations and Acronyms (Part 1)

Classification Methods

Acronym	Expansion
k-NN	k-Nearest Neighbors
SVM	Support Vector Machine
RBF	Radial Basis Function
RF	Random Forest
LDA	Linear Discriminant Analysis
QP	Quadratic Programming
KKT	Karush–Kuhn–Tucker

Clustering Methods

Acronym	Expansion
GMM	Gaussian Mixture Model
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
EM	Expectation–Maximization
BIC/AIC	Bayesian/Akaike Information Criterion

Evaluation Metrics

Acronym	Expansion
ARI	Adjusted Rand Index
NMI	Normalized Mutual Information
DB	Davies–Bouldin Index
AUC	Area Under the Curve
ROC	Receiver Operating Characteristic
F1	F1 Score (harmonic mean)
CV	Cross-Validation
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative

Abbreviations and Acronyms (Part 2)

Dimensionality Reduction (Part 1 Reference)

Acronym	Expansion
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
EVR	Explained Variance Ratio
t-SNE	t-distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection

Statistics & Linear Algebra

Acronym	Expansion
KL	Kullback–Leibler (divergence)
KDE	Kernel Density Estimation
IID	Indep. and Identically Distributed
MDI	Mean Decrease in Impurity
OOB	Out-of-Bag

Abbreviations and Acronyms (Part 3)

Data Structures & Computing

Acronym	Expansion
kd-tree	k-Dimensional Tree
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit

Datasets & General

Acronym	Expansion
MNIST	Modified NIST digit dataset
NLP	Natural Language Processing
CS	Computer Science
FSU	Florida State University
2D/3D	Two-/Three-Dimensional
d	Number of original features
k	Number of clusters / components
n	Number of data samples

LDA Note

Here LDA = **Linear Discriminant Analysis**, not Latent Dirichlet Allocation.