L03: KNN & K-Means

Deep Dive: Mathematical Foundations and Implementation

Methods and Algorithms

MSc Data Science

Spring 2026

Outline

The Math Behind “Similar Things Behave Similarly”

Today’s Deep Dive

In the overview, we saw that KNN classifies by neighbors and K-Means discovers clusters. Now we go deeper:

- How do we **formalize** distance and prove KNN’s theoretical guarantees?
- How do we **prove** K-Means convergence and understand its connection to EM?
- How do we **validate** clusters statistically before trusting them?

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

XKCD #1838 by Randall Munroe (CC BY-NC 2.5) – The math behind the “pile of linear algebra”

Learning Objectives

By the end of this lecture, you will be able to:

1. **Analyze** the bias-variance tradeoff in KNN and derive its asymptotic error bounds
2. **Evaluate** clustering validity using statistical tests (Hopkins, Gap statistic, silhouette)
3. **Prove** K-Means convergence and analyze computational complexity
4. **Compare** distance metrics and assess their suitability for high-dimensional finance data

Finance Applications: Customer segmentation (RFM), fraud detection

---

Bloom's Level 4–5: Analyze, Evaluate, Prove, Compare

Why Is KNN Called a Lazy Learner?

Key Insight

- No explicit model training – store all training data
- Classification by majority vote of K nearest neighbors
- "Lazy" because all computation happens at prediction time

The Algorithm in Words

1. Store all training examples $(\mathbf{x}_i, y_i)$
2. For new query $\mathbf{x}$: find K nearest training points
3. Return majority class among those K neighbors

Instance-based learning: the training data IS the model – no parameters to estimate

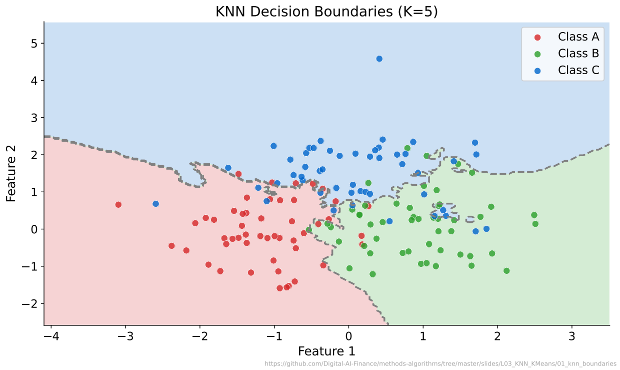
How Does the KNN Algorithm Work?

```
1: **Input**: Training set $\mathcal{D}$, query point $\mathbf{x}$, number of neighbors $K$
2: Compute $d(\mathbf{x}, \mathbf{x}_i)$ for all $\mathbf{x}_i \in \mathcal{D}$
3: Select $K$ points with smallest distances: $N_K(\mathbf{x})$
4: Count votes: $v_c = |\{i \in N_K(\mathbf{x}) : y_i = c\}|$ for each class $c$
5: if unique maximum class exists then
6:     $\hat{y} = \arg\max_c v_c$
7: else
8:     Tie-breaking: $\hat{y} = \arg\max_c \sum_{i:y_i=c} \frac{1}{d(\mathbf{x},\mathbf{x}_i)}$
9: end if
10: return $\hat{y}$
```

Complexity: $O(nd)$ per query (brute force over n samples, d features)

Tie-breaking via distance-weighted vote prevents non-determinism on decision boundaries

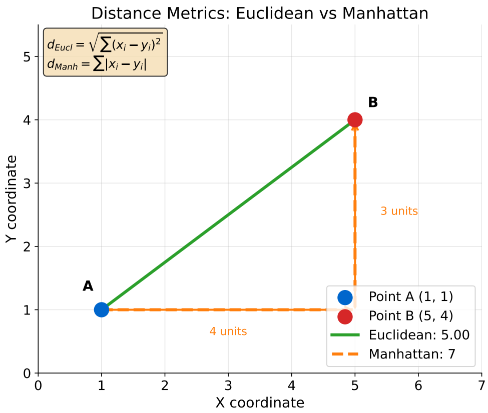
How Do KNN Boundaries Change with K?



- Small K creates jagged, complex boundaries that follow noise
- Large K creates smooth boundaries that may miss local patterns
- The optimal K balances flexibility with stability

Decision boundaries are non-parametric – they adapt to the local data structure

What Do Different Distance Metrics Look Like?



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L03_KNN_KMeans/02_distance_metrics

Key Insight: Different metrics define different "neighborhoods" – the choice of metric shapes the decision boundary and determines which points count as nearest neighbors.

Euclidean (circular), Manhattan (diamond), Chebyshev (square) unit balls in 2D

How Do We Generalize Distance?

Minkowski Distance

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

- $p = 1$: Manhattan (L1) – robust to outliers, sparse feature differences
- $p = 2$: Euclidean (L2) – default, geometrically intuitive
- $p = \infty$: Chebyshev – maximum absolute difference along any axis

Triangle Inequality: For $p \geq 1$, $d_p(\mathbf{x}, \mathbf{z}) \leq d_p(\mathbf{x}, \mathbf{y}) + d_p(\mathbf{y}, \mathbf{z})$

- Enables pruning in KD-Trees and Ball Trees
- For $p < 1$: violates triangle inequality (not a true metric)

Higher p amplifies large single-feature differences; $p = 2$ is the default for most applications

When Do We Need Non-Euclidean Distance?

Cosine Similarity (for text and embeddings)

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}, \quad \text{range: } [-1, 1]$$

- Measures angle, not magnitude – ideal for sparse, high-dimensional data
- Use for: document similarity, word embeddings, TF-IDF vectors

Mahalanobis Distance (accounts for correlation)

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

- Uses feature covariance Σ – unit-less, scale-invariant
- Detects outliers accounting for correlation structure

Choose metric based on data type: Euclidean for dense, Cosine for sparse/text, Mahalanobis for correlated features

How Does K Control the Bias-Variance Tradeoff?

The Fundamental Tradeoff

- $K = 1$: High variance, low bias – very flexible, memorizes noise
- $K = n$: High bias, low variance – always predicts majority class

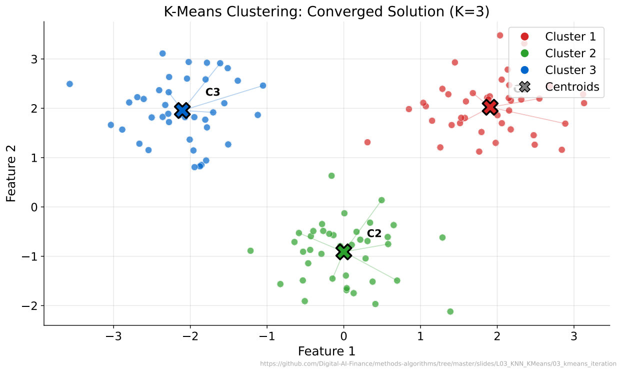
Practical Guidelines

- Start with $K = \sqrt{n}$ where n is training size
- Use odd K for binary classification (avoids ties)
- Always validate with cross-validation

Common Choices: $K \in \{3, 5, 7, 11\}$

Small K for complex patterns with clean data; larger K for noisy data requiring smoothing

How Do Centroids Move During Iteration?



- Each iteration: assign points to nearest centroid, then recompute centroids
- Centroids migrate toward cluster centers until stable
- Convergence typically occurs within 10–50 iterations

Visualizing iterations helps verify that K-Means is finding meaningful clusters

Why Weight Neighbors by Distance?

Problem: All K neighbors have equal influence – a distant neighbor counts as much as the closest one.

Inverse-Distance Weighting

$$\hat{y} = \arg\max_{c} \sum_{i \in N_K(\mathbf{x})} w_i \cdot \mathbf{1}[y_i = c] \quad \text{where } w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)}$$

- Closer neighbors receive exponentially higher weight
- Reduces sensitivity to the choice of K
- For $d = 0$ (exact match): return that point's class directly

In scikit-learn: `weights='uniform'` (default) vs `weights='distance'`

Distance weighting often improves performance, especially when K is not perfectly tuned

How Do We Select K Objectively?

GridSearchCV for Optimal K

- `param_grid = {'n_neighbors': range(1, 21, 2)}`
- `grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)`
- `grid.fit(X_train, y_train)`
- `best_k = grid.best_params_['n_neighbors']`

Validation Curve Interpretation

- Plot accuracy vs K for both training and validation sets
- Choose K where validation accuracy peaks (before overfitting gap widens)
- Large gap between train/val curves signals overfitting

Cross-validation provides objective, data-driven K selection – never choose K by "eyeballing"

Why Must We Scale Features for KNN?

Why Scaling Matters – Without scaling:

- Income: ranges 20,000–200,000
- Age: ranges 20–80
- Distance dominated by income (180,000$\times$ larger scale!)

Scaling Methods

- **Standardization**: $z = \frac{x-\mu}{\sigma}$ (mean=0, std=1)
- **Min-Max**: $x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$ (range [0,1])

Edge Cases

- If $\sigma = 0$ or $x_{\text{max}} = x_{\text{min}}$ (constant feature): drop feature or set to 0

Rule: ALWAYS scale features for distance-based methods. StandardScaler for Gaussian-like, MinMaxScaler for bounded.

Why Does KNN Struggle in High Dimensions?

The Problem (Beyer et al., 1999)
In high dimensions, for any query point **x**:

$$\lim_{d \to \infty} \frac{d_{\max} - d_{\min}}{d_{\min}} \to 0$$

- All points become approximately equidistant
- "Nearest neighbor" becomes meaningless
- Volume of unit hypersphere $\to 0$ as $d \to \infty$

Solutions

- **Dimensionality reduction**: PCA before KNN (see L05)
- **Feature selection**: keep only relevant features
- **Domain knowledge**: select meaningful features from subject matter

KNN works best with moderate dimensionality ($d < 15$–20, problem-dependent)

How Good Can KNN Theoretically Be?

Theorem: As $n \to \infty$, the 1-NN error rate satisfies:

$$R^* \leq R_{\text{1-NN}} \leq 2R^*(1 - R^*)$$

where R^* is the Bayes optimal error rate.

Key Results

- 1-NN error is at most $2\times$ Bayes error – remarkably strong for such a simple method
- KNN is **universally consistent**: $R_{K\text{-NN}} \to R^*$ exactly
- Requires: $K \to \infty$ and $K/n \to 0$ simultaneously

Practical Implication: KNN is a strong baseline – if it performs badly, the features may be poor rather than the algorithm.

Universal consistency holds under mild conditions; see appendix for full proof

How Do We Prove the 1-NN Error Bound?

Proof idea (1-NN case, binary classification):

1. As $n \to \infty$, the nearest neighbor $\mathbf{x}_{(1)} \to \mathbf{x}$ (converges to query)
2. Let $\eta(\mathbf{x}) = P(Y = 1|\mathbf{x})$. The 1-NN error at $\mathbf{x}$:

$$R_{\text{1-NN}}(\mathbf{x}) = \eta(\mathbf{x})(1 - \eta(\mathbf{x})) + (1 - \eta(\mathbf{x}))\eta(\mathbf{x}) = 2\eta(\mathbf{x})(1 - \eta(\mathbf{x}))$$

Since $R^*(\mathbf{x}) = \min(\eta, 1 - \eta)$ and $2\eta(1 - \eta) \leq 2\min(\eta, 1 - \eta)$:

$$R^* \leq R_{\text{1-NN}} \leq 2R^*(1 - R^*)$$

For K-NN with $K \to \infty$, $K/n \to 0$: majority vote over K neighbors converges to $\mathbf{1}[\eta(\mathbf{x}) > 0.5]$, so $R_{K\text{-NN}} \to R^*$ exactly.

The $2\times$ bound is tight: equality when $\eta(\mathbf{x}) = 0.5$ everywhere (maximum ambiguity)

How Does K Affect Bias and Variance?

For KNN regression, $\hat{f}_K(\mathbf{x}) = \frac{1}{K}\sum_{i \in N_K(\mathbf{x})} y_i$, the expected error decomposes as:

$$E[(\hat{f}_K(\mathbf{x}) - Y)^2] = \underbrace{\text{Bias}^2(\hat{f}_K(\mathbf{x}))}_{\text{increases with } K} + \underbrace{\text{Var}(\hat{f}_K(\mathbf{x}))}_{\text{decreases with } K} + \sigma^2$$

- **Variance**: $\text{Var}(\hat{f}_K) = \frac{\sigma^2}{K}$ (averaging K neighbors reduces variance)
- **Bias**: $\text{Bias}(\hat{f}_K) \approx f(\mathbf{x}) - \frac{1}{K}\sum_{i \in N_K} f(\mathbf{x}_i)$ (larger K = farther neighbors = more bias)
- **Optimal** K: balances this tradeoff; found via cross-validation

$K = 1$: **zero bias, variance** $= \sigma^2$. $K = n$: **high bias (global mean), variance** $= \sigma^2/n$.

What Is K-Means Trying to Optimize?

Goal: Partition n points into K clusters by minimizing within-cluster sum of squares (WCSS):

$$J = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

where $\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$ is the centroid of cluster C_k.

Key Properties

- Each point assigned to its nearest centroid
- Iterative refinement: assign $\rightarrow$ update $\rightarrow$ repeat
- Converges to local (not global) optimum

Finding the globally optimal K-Means solution is NP-hard (Aloise et al., 2009)

How Does Lloyd's Algorithm Work?

```
1: Input: Data $\mathbf{X}$, number of clusters $K$
2: Initialize $K$ centroids (K-Means++ or random)
3: repeat
4:     Assignment: $C_k = \{\mathbf{x}_i : \|\mathbf{x}_i - \boldsymbol{\mu}_k\| \leq \|\mathbf{x}_i - \boldsymbol{\mu}_j\| \; \forall j\}$
5:     Update: $\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$
6:     Handle empty: if $|C_k| = 0$, reinitialize $\boldsymbol{\mu}_k$ from farthest point
7: until centroid change $< \epsilon$ or max iterations reached
8: return cluster assignments, centroids
```

Convergence: Guaranteed – WCSS decreases monotonically each iteration. May converge to local optimum; run multiple restarts.

Each iteration costs $O(nKd)$ where n = samples, K = clusters, d = features

Why Does Smart Initialization Matter?

Problem: Random initialization is sensitive to starting positions and often converges to poor local optima.

K-Means++ Algorithm (Arthur & Vassilvitskii, 2007)

1. Choose first centroid uniformly at random from data
2. For each subsequent centroid: select point $\mathbf{x}$ with probability $\propto d(\mathbf{x})^2$ (squared distance to nearest existing centroid)
3. Repeat until K centroids chosen

Guarantee: Expected cost $\leq 8(\ln K + 2)\times$ optimal cost – an $O(\log K)$-competitive approximation.

K-Means++ is the default in scikit-learn (`init='k-means++'`); typically converges in fewer iterations

How Do We Measure Cluster Quality?

For each point i:

- $a(i)$: average distance to all other points in **same** cluster (cohesion)
- $b(i)$: average distance to points in **nearest other** cluster (separation)

Silhouette coefficient:

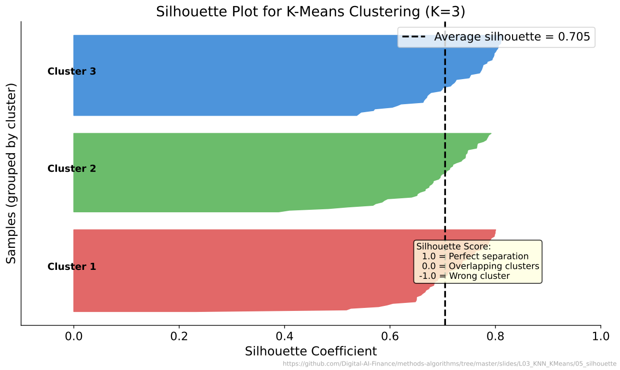
$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}, \quad s(i) \in [-1, 1]$$

Interpretation:

- $s \approx 1$: point is well-clustered (far from neighboring clusters)
- $s \approx 0$: point lies on the boundary between clusters
- $s < 0$: point is likely assigned to the wrong cluster

Average silhouette score across all points summarizes overall clustering quality. Singleton: $s(i) = 0$ by convention.

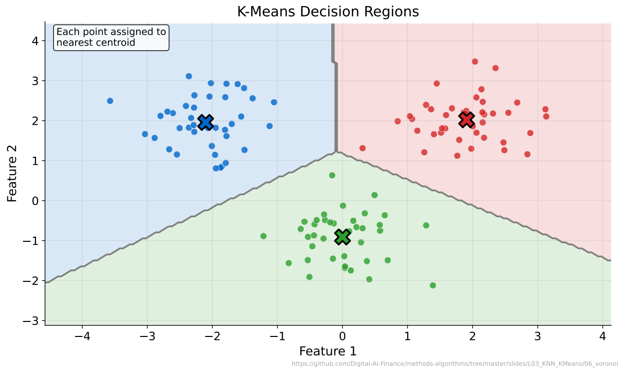
What Does a Silhouette Plot Reveal?



Reading the Plot: Each cluster should have similar width (balanced sizes) and scores above the average line. Thin or negative clusters indicate poor assignments.

Compare silhouette plots across different K values to find the optimal number of clusters

What Shape Are K-Means Clusters?



Voronoi Tessellation: K-Means partitions the feature space into convex polygonal regions. Every point within a region is closest to that region's centroid.

Voronoi cells are always convex – this is why K-Means cannot discover non-convex cluster shapes

When Does K-Means Fail?

What K-Means Assumes

- Clusters are **spherical** (isotropic variance in all directions)
- Clusters have **similar sizes** (roughly equal number of points)
- Clusters have **similar densities**

When K-Means Fails

- Non-convex shapes (e.g., crescents, rings) – use DBSCAN
- Very different cluster sizes or densities – use GMM
- Strong outliers – use K-Medoids (centroids must be data points)

Always visualize clusters in 2D (via PCA/t-SNE) to check if assumptions hold

Why Must K-Means Converge?

Theorem: Lloyd's algorithm converges in a finite number of iterations.

Proof (coordinate descent argument):

The WCSS objective $J = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$ satisfies:

1. **Assignment step** (fix $\boldsymbol{\mu}$, optimize C): Reassigning each point to its nearest centroid can only decrease $J \quad \Rightarrow J$ non-increasing
2. **Update step** (fix C, optimize $\boldsymbol{\mu}$): The mean minimizes within-cluster SSE $\quad \Rightarrow J$ non-increasing

Since $J \geq 0$ and non-increasing, and there are finitely many partitions of n points into K clusters: the algorithm must terminate. $\square$

NP-hardness: Global optimum is NP-hard; K-Means++ provides $O(\log K)$ approximation.

Block coordinate descent: alternating optimization of two blocks (C and $\boldsymbol{\mu}$) on a bounded objective

How Does K-Means Relate to EM?

Connection to Expectation-Maximization

- K-Means = "Hard EM" for Gaussian Mixture Models (GMM)
- Assumes: spherical Gaussians with equal variance $\sigma^2 \boldsymbol{I}$
- **E-step**: assign points to nearest centroid (hard assignment, $\gamma_{ik} \in \{0, 1\}$)
- **M-step**: update centroids as cluster means

Why This Matters

- Explains WHY convergence is guaranteed (EM always converges)
- Explains WHY K-Means assumes spherical clusters
- Opens door to **soft clustering** via full GMM (probabilistic assignments)

Soft EM: $\gamma_{ik} \in [0, 1]$ gives probability of membership – more flexible but computationally heavier

What Alternatives Exist to Standard K-Means?

Mini-Batch K-Means

- Uses random subsets (mini-batches) for centroid updates
- 10–100$\times$ faster for large datasets; slightly worse results

K-Medoids (PAM)

- Centroids must be actual data points (medoids)
- More robust to outliers; works with any distance metric

K-Modes / K-Prototypes

- K-Modes: for categorical data (uses mode instead of mean)
- K-Prototypes: mixed continuous and categorical features

Mini-Batch K-Means is recommended for $n > 10{,}000$ samples; K-Medoids for non-Euclidean distances

How Fast Are These Algorithms?

Method	**Training**	**Prediction**
KNN (brute force)	$O(1)$	$O(nd)$
KNN (KD-Tree)	$O(nd \log n)$	$O(d \log n)$
KNN (Ball Tree)	$O(nd \log n)$	$O(d \log n)$
K-Means (Lloyd's)	$O(nKdT)$	$O(Kd)$
Mini-Batch K-Means	$O(bKdT)$	$O(Kd)$

n = samples, d = features, K = neighbors/clusters, T = iterations, b = batch size.

- KD-Tree degrades to $O(nd)$ when $d > 15$–20
- For very large n: approximate nearest neighbors (LSH, FAISS)

scikit-learn `algorithm='auto'` **selects the best method based on** n **and** d

Should We Cluster This Data at All?

Before Clustering: Test whether data has cluster tendency at all.

$$H = \frac{\sum_{i=1}^{m} u_i^d}{\sum_{i=1}^{m} u_i^d + \sum_{i=1}^{m} w_i^d}$$

- u_i = distance from random point (uniform in data range) to nearest data point
- w_i = distance from randomly sampled data point to its nearest neighbor

Interpretation:

- $H \approx 0.5$: uniform distribution (no clusters) – do not cluster
- $H > 0.75$: significant clustering tendency – proceed with clustering

Always run Hopkins test before K-Means – clustering uniform data produces meaningless results

How Does the Gap Statistic Choose K?

Gap Statistic (Tibshirani et al., 2001)

$$\text{Gap}_n(k) = E_n^*[\log W_k] - \log W_k$$

where $W_k = \sum_{r=1}^{k} \frac{1}{2|C_r|} \sum_{i,j \in C_r} \|x_i - x_j\|^2$ and E_n^* is the expectation under a reference (uniform) distribution.

Selection Rule: Choose smallest K such that:

$$\text{Gap}(k) \geq \text{Gap}(k+1) - s_{k+1}$$

where s_{k+1} is the standard error from B bootstrap reference samples.

Advantage over Elbow: Provides a statistical criterion rather than subjective visual inspection.

Tibshirani et al. recommend $B = 20$–50 **bootstrap samples for stable estimates**

Which K Selection Method Should We Use?

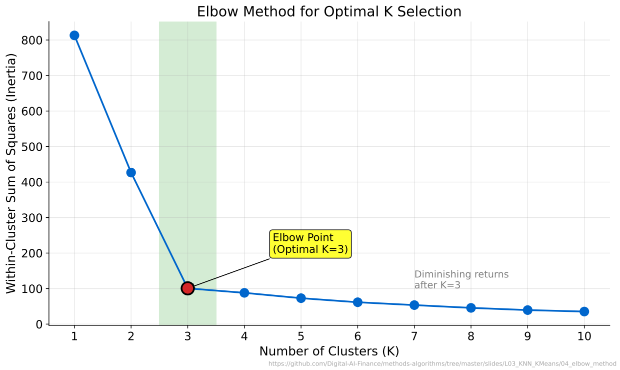
Method	Objectivity	Speed	Best For
Elbow (WCSS)	Low (subjective)	Fast	Quick exploration
Silhouette	High	Moderate	Cluster quality
Gap Statistic	High (statistical)	Slow	Rigorous analysis

Practical Recommendation:

- Start with elbow plot for quick overview
- Validate with silhouette score and silhouette plots
- Use Gap statistic when formal justification is needed (e.g., publications)

No single method is perfect – use multiple methods and check if they agree

Where Is the Elbow in the WCSS Curve?



- WCSS always decreases with more clusters – the question is *how fast*
- The “elbow” marks the point of diminishing returns
- Combine with silhouette analysis for a more objective decision

The elbow is often ambiguous – that is why we also use silhouette and Gap statistics

How Do KNN and K-Means Differ Fundamentally?

Aspect	**KNN**	**K-Means**
Task	Classification / Regression	Clustering
Learning Type	Supervised (needs labels)	Unsupervised (no labels)
K meaning	Number of neighbors	Number of clusters
Training	None (lazy learner)	Iterative optimization
Prediction	Compute distances to all	Assign to nearest centroid
Output	Class label / value	Cluster assignment
Scalability	$O(nd)$ per query	$O(nKdT)$ total

The "K" in KNN and K-Means mean completely different things – a common source of confusion!

How Do Banks Segment Customers with RFM?

RFM Analysis (Industry Standard)

- **R**ecency: Days since last transaction
- **F**requency: Number of transactions in period
- **M**onetary: Total / average transaction value

Pipeline: Standardize RFM $\rightarrow$ K-Means (K=4–6) $\rightarrow$ Profile segments **Typical Segments**:

- "Champions" (high R, F, M) $\rightarrow$ loyalty rewards
- "At Risk" (low R, declining F) $\rightarrow$ retention campaigns
- "New Customers" (high R, low F) $\rightarrow$ onboarding programs

Business Value: Customer Lifetime Value (CLV) prediction per segment

RFM segmentation is foundational for CRM, marketing analytics, and credit risk profiling

How Does KNN Detect Financial Fraud?

CRITICAL: Class Imbalance Problem

- Fraud is typically $<1\%$ of transactions (100:1 ratio)
- Naive KNN majority vote: ALWAYS predicts non-fraud (99% accuracy, zero fraud detected!)

Solutions for Imbalanced Data

- **SMOTE**: Synthetic Minority Oversampling TEchnique
- **Weighted KNN**: Higher weight for minority class neighbors
- **Anomaly score**: Use distance to K-th neighbor instead of majority vote

Evaluation: Use **Precision-Recall AUC**, NOT accuracy!

Always check class distribution before applying majority vote – accuracy is misleading for imbalanced data

What If Clusters Aren't Spherical?

DBSCAN (Density-Based Spatial Clustering):

- **Core point**: $\geq$ minPts neighbors within radius ε
- **Border point**: within ε of a core point but not itself core
- **Noise**: neither core nor border $\Rightarrow$ automatic outlier detection
- Does not require K; discovers non-spherical clusters

Hierarchical (Agglomerative) Clustering:

- Bottom-up: each point starts as its own cluster, iteratively merge closest pairs
- Linkage: single (min), complete (max), Ward (minimize variance)
- **Dendrogram**: cut at desired height to get K clusters

DBSCAN for irregular shapes with outliers; hierarchical when exploring multiple K values via dendrogram

Which Algorithm Should You Choose?

Use KNN When

- You have labeled training data (supervised)
- Local patterns matter (non-linear boundaries)
- Interpretability: “classified because similar to these examples”

Use K-Means When

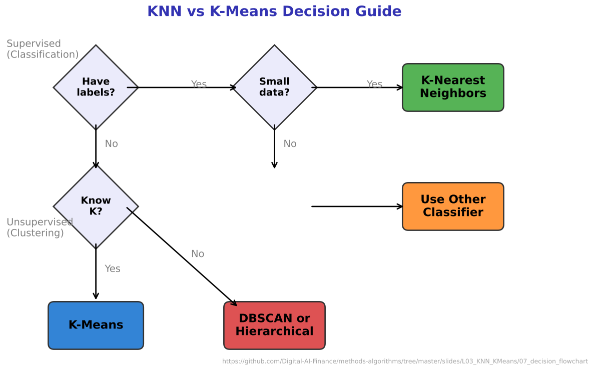
- No labels available (unsupervised)
- Looking for natural groupings in data
- Clusters are roughly spherical and similar in size

Use Alternatives When

- Non-spherical clusters $\rightarrow$ DBSCAN or spectral clustering
- Soft assignments needed $\rightarrow$ Gaussian Mixture Models
- Very large n $\rightarrow$ Mini-Batch K-Means or approximate NN

K-Means is often used as preprocessing (cluster features) before supervised learning

Decision Flowchart: KNN vs K-Means vs Alternatives


KNN vs K-Means Decision Guide
Supervised (Classification)
Have labels?
Yes
Small data?
Yes
K-Nearest Neighbors
No
No
Know K?
Unsupervised (Clustering)
Use Other Classifier
No
Yes
K-Means
DBSCAN or Hierarchical
https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L03_KNN_KMeans/07_decision_flowchart

- Labels available → supervised methods (KNN, logistic regression, random forests)
- No labels, spherical clusters → K-Means with K-Means++ initialization
- Non-spherical or unknown K → DBSCAN or hierarchical clustering

Start simple (K-Means) and add complexity only when validation metrics demand it

How Do We Implement KNN in Python?

Classification

- `from sklearn.neighbors import KNeighborsClassifier`
- `knn = KNeighborsClassifier(n_neighbors=5, weights='distance')`
- `knn.fit(X_train, y_train)`
- `y_pred = knn.predict(X_test)`

Key Parameters

- `n_neighbors`: K value (default 5)
- `weights`: `'uniform'` or `'distance'`
- `metric`: `'euclidean'`, `'manhattan'`, `'minkowski'`
- `algorithm`: `'auto'`, `'ball_tree'`, `'kd_tree'`, `'brute'`

Also available: `KNeighborsRegressor` **for regression;** `RadiusNeighborsClassifier` **for radius-based**

How Do We Implement K-Means in Python?

Basic Usage

- `from sklearn.cluster import KMeans`
- `km = KMeans(n_clusters=3, init='k-means++', random_state=42)`
- `labels = km.fit_predict(X)`
- `centroids = km.cluster_centers_`

Key Parameters

- `n_clusters`: K (required, no default)
- `init`: `'k-means++'` (default) or `'random'`
- `n_init`: number of restarts (default 10; keeps best)
- `max_iter`: max iterations per run (default 300)

`km.inertia_` **gives WCSS after fitting;** `km.n_iter_` **gives iterations until convergence**

Why Must Scaling Be Inside the Pipeline?

Why Pipelines? Prevent data leakage – scaling must be fit on training data only.

- `from sklearn.pipeline import Pipeline`
- `from sklearn.preprocessing import StandardScaler`
- `pipe = Pipeline([('scaler', StandardScaler()),`
- `                 ('knn', KNeighborsClassifier())])`
- `pipe.fit(X_train, y_train)`

GridSearchCV with Pipeline

- `params = {'knn__n_neighbors': [3,5,7,11],`
- `          'knn__weights': ['uniform','distance']}`
- `grid = GridSearchCV(pipe, params, cv=5, scoring='f1')`

Pipelines ensure scaling is part of cross-validation – fitting scaler on full data before CV causes leakage

Hands-On Exercise

Open the Colab Notebook

- **Exercise 1**: Implement weighted KNN with distance weighting; compare uniform vs distance on fraud data
- **Exercise 2**: Compare Gap statistic vs Elbow vs Silhouette for K selection on customer data
- **Exercise 3**: Apply SMOTE + KNN pipeline for imbalanced fraud detection; evaluate with PR AUC

Link: See course materials for Colab notebook

All exercises use real-world-inspired financial datasets with class imbalance and mixed scales

What Should You Remember?

K-Nearest Neighbors

- Instance-based lazy learner; Cover & Hart bound: $R_{\text{1-NN}} \leq 2R^*$
- Scale features, choose K via cross-validation, consider weighted voting

K-Means

- Iterative assign-update with guaranteed convergence (local optimum)
- K-Means++ for initialization; silhouette/Gap for K selection

Common Considerations

- Feature scaling is critical for both methods
- “K” means completely different things in each algorithm

Finance: RFM segmentation (K-Means), fraud detection with SMOTE (KNN)

Both are foundational algorithms: simple, interpretable, widely used across industries

Until Next Time...

"Even K-Means would struggle to cluster the ways students misuse K-Means."

With KNN and K-Means, you can now classify the known and discover the unknown.

Next Session: L04 – Random Forests (from distance-based to tree-based methods)

XKCD #2731 callback – clustering is easy, knowing when to cluster is the hard part

Appendix: Advanced Topics and Proofs

Distance Metric Properties: Triangle Inequality

Statement: For any metric d and points $\mathbf{x}, \mathbf{y}, \mathbf{z}$:

$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$$

Proof for Euclidean (via Cauchy-Schwarz):

1. $\|\mathbf{x} - \mathbf{z}\| = \|(\mathbf{x} - \mathbf{y}) + (\mathbf{y} - \mathbf{z})\|$
2. $\leq \|\mathbf{x} - \mathbf{y}\| + \|\mathbf{y} - \mathbf{z}\|$ (by Minkowski inequality, itself a consequence of Cauchy-Schwarz)

Why It Enables KD-Tree Pruning:

- If current best distance is d^* and node center is $\mathbf{c}$:
- Prune subtree if $d(\mathbf{x}, \mathbf{c}) - r > d^*$ (where $r =$ node radius)
- Triangle inequality guarantees no closer point can exist in that subtree

Without triangle inequality ($p < 1$), spatial indexing structures cannot prune and brute force is required

K-Means Convergence: Formal Proof

Objective: $J(C, \boldsymbol{\mu}) = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$
Block Coordinate Descent:

1. **Fix $\boldsymbol{\mu}$, optimize C**: For each $\mathbf{x}_i$, $\frac{\partial J}{\partial C}$ is minimized by $C_k^* = \{\mathbf{x}_i : k = \arg\min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|\} \quad \Rightarrow J^{(t+1)} \leq J^{(t)}$
2. **Fix C, optimize $\boldsymbol{\mu}$**: $\frac{\partial J}{\partial \boldsymbol{\mu}_k} = -2\sum_{\mathbf{x}_i \in C_k}(\mathbf{x}_i - \boldsymbol{\mu}_k) = 0$ gives $\boldsymbol{\mu}_k^* = \frac{1}{|C_k|}\sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i \quad \Rightarrow J^{(t+1)} \leq J^{(t)}$

Termination: $J \geq 0$ and strictly non-increasing. The number of distinct partitions of n points into K groups is $\leq K^n$ (finite). Each partition visited at most once $\Rightarrow$ convergence in $\leq K^n$ steps (in practice, much fewer).

Worst-case K^n iterations is exponential but never observed in practice; typical convergence in 10–50 iterations

Empty Cluster Handling Strategies

Problem: During K-Means iteration, a cluster may lose all its points.
Strategy 1: Farthest Point Reinitialization

- Find the point farthest from its assigned centroid
- Use it as the new centroid for the empty cluster
- Rationale: poorly fit points are good candidates for new clusters

Strategy 2: Split Largest Cluster

- Find cluster with highest WCSS, split it into two
- Maintains total number of clusters at K

Strategy 3: Random Reinitialization

- Randomly select a new data point as centroid
- Simplest but least principled approach

scikit-learn uses a reassignment strategy; K-Means++ initialization makes empty clusters rare

Cover & Hart: Full Proof Details

Setup: Binary classification, $\eta(\mathbf{x}) = P(Y = 1|\mathbf{x})$, Bayes rule: $g^*(\mathbf{x}) = \mathbf{1}[\eta(\mathbf{x}) > 0.5]$.
1-NN Error Rate Derivation:

- As $n \to \infty$, nearest neighbor $\mathbf{x}_{(1)} \to \mathbf{x}$, so $\eta(\mathbf{x}_{(1)}) \to \eta(\mathbf{x})$
- $P(\text{error}|\mathbf{x}) = P(Y \neq Y_{(1)}|\mathbf{x})$
- $= \eta(\mathbf{x})(1 - \eta(\mathbf{x})) + (1 - \eta(\mathbf{x}))\eta(\mathbf{x}) = 2\eta(\mathbf{x})(1 - \eta(\mathbf{x}))$

Bounding:

$$R^*(\mathbf{x}) = \min(\eta, 1 - \eta) \leq 2\eta(1 - \eta) \leq 2\min(\eta, 1 - \eta) = 2R^*(\mathbf{x})$$

Integrating over $\mathbf{x}$: $R^* \leq R_{\text{1-NN}} \leq 2R^*(1 - R^*)$

K-**NN Extension**: With $K \to \infty$, $K/n \to 0$, majority vote converges to $\mathbf{1}[\eta > 0.5]$ by law of large numbers
$\Rightarrow R_{K\text{-NN}} \to R^*$.

The tighter bound $2R^*(1 - R^*) \leq 2R^*$ shows 1-NN is at most 2$\times$ suboptimal

KNN Computational Complexity

Algorithm	**Build**	**Query (avg)**	**Query (worst)**
Brute Force	$O(1)$	$O(nd)$	$O(nd)$
KD-Tree	$O(nd \log n)$	$O(d \log n)$	$O(nd)$
Ball Tree	$O(nd \log n)$	$O(d \log n)$	$O(nd)$
LSH (approx.)	$O(nd)$	$O(d)$	$O(nd)$

When Each Breaks Down:

- KD-Tree: degrades to brute force for $d > 15$–20 (curse of dimensionality)
- Ball Tree: better than KD-Tree in moderate-high d but still degrades
- LSH: approximate; tunable accuracy-speed tradeoff via hash functions

Locality-Sensitive Hashing (LSH) is used by FAISS, Annoy, and other large-scale NN libraries

Gap Statistic: Mathematical Details

Reference Distribution: Generate B datasets from uniform distribution over bounding box of data (or from PCA-aligned box for better null model).

Bootstrap Procedure:

1. For each $k = 1, \ldots, K_{\max}$: compute $\log W_k$ on real data
2. For $b = 1, \ldots, B$: generate reference data, compute $\log W_k^{*(b)}$
3. $\text{Gap}(k) = \frac{1}{B} \sum_{b=1}^{B} \log W_k^{*(b)} - \log W_k$
4. $s_k = \text{sd}(\log W_k^{*(b)}) \cdot \sqrt{1 + 1/B}$

Formal Selection Rule: Choose smallest k such that:

$$\text{Gap}(k) \geq \text{Gap}(k+1) - s_{k+1}$$

Intuition: Gap measures how much better the clustering is compared to clustering uniform (structureless) data.

Tibshirani et al. (2001) recommend $B = 20$–50; the $\sqrt{1 + 1/B}$ correction accounts for simulation error

References

Textbooks

- James et al. (2021). *ISLR*, Ch. 2 (KNN), Ch. 12 (Clustering)
- Hastie et al. (2009). *ESL*, Ch. 13 (Prototypes/NN), Ch. 14 (Unsupervised)

Key Papers

- Arthur & Vassilvitskii (2007). K-Means++: The Advantages of Careful Seeding
- Cover & Hart (1967). Nearest Neighbor Pattern Classification
- Tibshirani et al. (2001). Estimating the Number of Clusters via the Gap Statistic
- Beyer et al. (1999). When Is "Nearest Neighbor" Meaningful?

Next Lecture: L04 – Random Forests (from distance-based to tree-based ensemble methods)

All papers are available via university library access or open-access preprints