K MODES CLUSTERING

~S S Roy,5th Sept,2025

What it is (and why it exists)

K-Modes is the categorical-data sibling of k-means.
Where k-means uses **means** and **squared Euclidean distance** (which break on strings/categories), k-modes uses:

- **Modes** (most frequent category per feature) instead of means
- **Simple matching dissimilarity** (count of mismatched categories) instead of squared distance

This makes it ideal for data sets like survey responses, SKUs with attributes, demographics, medical codes, etc.

Core definitions

1) Dissimilarity (distance)

For two categorical vectors $x = (x_1, \ldots, x_p)$ and $y = (y_1, \ldots, y_p)$, the usual distance is

$$d(x, y) = \sum_{j=1}^{p} 1\{x_j \neq y_j\}$$

(You can optionally **weight** features: $d_w(x, y) = \sum_j w_j 1\{x_j \neq y_j\}$.)

2) Cluster representative (mode)

Given a cluster C and feature j, the **mode** m_j is the category with highest frequency in that feature within C.
The **cluster center** is $m = (m_1, \ldots, m_p)$.

3) Objective function (what k-modes minimizes)

$$\text{Cost}(C_1, \dots, C_k) = \sum_{i=1}^{n} d(x_i, \text{mode of its cluster})$$

i.e., the **total number of within-cluster mismatches**.

The algorithm (Huang, 1998) — step-by-step

1. **Initialize** k modes (seeds):
   - **Random**: pick k rows at random.
   - **Huang**: for each feature, sample categories proportionally to frequency to build diverse, realistic seeds.
   - **Cao**: density-based seeding; spreads modes towards dense regions.
2. **Assign** each record to the nearest mode by simple matching dissimilarity.
3. **Update** modes: in each cluster, set each feature to its most frequent category (break ties consistently, e.g., highest frequency across other features or lexicographic).
4. **Repeat** assign ↔ update until modes don't change (or cost stops improving).

- **Per-iteration cost**: $O(nkp)$ with n rows, k clusters, p features.

A worked example (complete table + full first iteration)

We'll cluster $k = 2$ on a tiny retail-attributes dataset with four categorical columns:

- **Size** ∈ {S, M, L}
- **Fit** ∈ {Slim, Regular}
- **Color** ∈ {Red, Blue, Green}
- **Fabric** ∈ {Cotton, Polyester}

Example data table

ID	Size	Fit	Color	Fabric
1	S	Slim	Red	Cotton
2	S	Slim	Blue	Cotton
3	M	Regular	Blue	Cotton

ID	Size	Fit	Color	Fabric
4	L	Regular	Blue	Polyester
5	L	Regular	Green	Polyester
6	M	Slim	Red	Cotton
7	M	Regular	Blue	Polyester
8	L	Regular	Blue	Cotton

Initialization (random pick):

$Mode_1$ ← row **1** = (S, Slim, Red, Cotton)

$Mode_2$ ← row **4** = (L, Regular, Blue, Polyester)

First assignment step (simple-matching distance)

For each row, count feature mismatches vs each mode (0 = perfect match, 4 = all different):

ID	to $Mode_1$ (S,Slim,Red,Cotton)	to $Mode_2$ (L,Regular,Blue,Poly)	Assign
1	0	4	C_1
2	1 (Color)	3 (Size, Fit, Fabric)	C_1
3	3 (Size, Fit, Color)	2 (Size, Fabric)	C_2
4	4	0	C_2
5	4	1 (Color)	C_2
6	1 (Size)	4	C_1
7	4	1 (Size)	C_2
8	3 (Size, Fit, Color)	1 (Fabric)	C_2

Clusters after assignment:

- **C_1** = {1, 2, 6}
- **C_2** = {3, 4, 5, 7, 8}

Update modes

- For $\mathbf{C_1}$ (rows 1,2,6):
  - Size: S(2), M(1) → **S**
  - Fit: Slim(3) → **Slim**
  - Color: Red(2), Blue(1) → **Red**
  - Fabric: Cotton(3) → **Cotton**

    Mode_1' = (S, Slim, Red, Cotton) (unchanged)
- For $\mathbf{C_2}$ (rows 3,4,5,7,8):
  - Size: L(3), M(2) → **L**
  - Fit: Regular(5) → **Regular**
  - Color: Blue(4), Green(1) → **Blue**
  - Fabric: Polyester(3), Cotton(2) → **Polyester**

    Mode_2' = (L, Regular, Blue, Polyester) (unchanged)

Since modes didn't change, the algorithm **converges** after one full iteration here.

Final cost (sum of distances to final modes)

- C_1 distances: 0 (ID1) + 1 (ID2) + 1 (ID6) = **2**
- C_2 distances: 2 (ID3) + 0 (ID4) + 1 (ID5) + 1 (ID7) + 1 (ID8) = **5**

  Total cost = 7.

How to choose k

There's no single "right" k, but common practices:

1. **Elbow on cost**: run k-modes for $k = 1, 2, \ldots$ and plot total cost; pick the elbow.
2. **Holdout validation**: train on a subset, compute assignment cost on a held-out set; pick k that minimizes held-out cost.
3. **Stability**: run multiple seeds; pick k with high stability (e.g., high Adjusted Rand Index across runs).
4. **Silhouette-like scores for categorical data** (using Hamming/simple-matching) — useful but more nuanced than numeric silhouette.

Practical details & best practices

- **Initialization matters**: use Huang or Cao seeds and **multiple restarts** to avoid poor local minima.

- **Ties when updating modes**: break deterministically (e.g., pick the category that yields lower total cost, or a fixed order).
- **Feature scaling & weights**: high-cardinality or business-critical fields can be up- or down-weighted.
- **Missing values**: treat missing as a special category (e.g., "NA") or impute; be consistent at train & inference time.
- **Unseen categories at inference**: map to "other"/"rare" or nearest known category by domain rules.
- **Encoding**: **Do not** one-hot encode before k-modes; pass raw categorical labels (strings or ints).
- **Mixed data**: For numeric + categorical, use **k-prototypes** (extends k-modes + k-means).
- **Complexity**: Each iteration is $O(nkp)$; k-modes is typically fast for tidy categorical tables.

Minimal, readable Python

A. From-scratch, didactic k-modes (for learning)

```python
from collections import Counter
import random

def hamming(x, y):
    return sum(a != b for a, b in zip(x, y))

def mode_of_cluster(rows):
    # rows: list of tuples/arrays of categorical values with same length
    p = len(rows[0])
    mode = []
    for j in range(p):
        counts = Counter(r[j] for r in rows)
        # break ties by (freq desc, value asc) for determinism
        mode.append(sorted(counts.items(), key=lambda t: (-t[1], str(t[0])))[0][0])
    return tuple(mode)

def kmodes(X, k, max_iter=100, n_init=5, seed=None):
    rng = random.Random(seed)
```

```
    best = None
    for _ in range(n_init):
        # Huang-like start: pick k distinct rows as initial modes
        modes = [tuple(row) for row in rng.sample(X, k)]
        for _ in range(max_iter):
            # assign
            assigns = [[] for _ in range(k)]
            for row in X:
                dists = [hamming(row, m) for m in modes]
                j = min(range(k), key=lambda idx: (dists[idx], idx))  # tie-break by index
                assigns[j].append(row)
            # update
            new_modes = []
            for j in range(k):
                new_modes.append(mode_of_cluster(assigns[j]) if assigns[j] else modes[j])
            if new_modes == modes:
                break
            modes = new_modes
        # compute cost
        cost = sum(min(hamming(row, m) for m in modes) for row in X)
        if best is None or cost < best[0]:
            best = (cost, modes, assigns)
    cost, modes, assigns = best
    labels = []
    for row in X:
        dists = [hamming(row, m) for m in modes]
        labels.append(min(range(k), key=lambda idx: (dists[idx], idx)))
    return {"modes": modes, "labels": labels, "cost": cost}
```

Try it on the example:

```python
X = [
 ("S","Slim","Red","Cotton"),
 ("S","Slim","Blue","Cotton"),
 ("M","Regular","Blue","Cotton"),
 ("L","Regular","Blue","Polyester"),
 ("L","Regular","Green","Polyester"),
 ("M","Slim","Red","Cotton"),
 ("M","Regular","Blue","Polyester"),
 ("L","Regular","Blue","Cotton"),
]
```

```python
res = kmodes(X, k=2, n_init=10, seed=42)
print("Modes:", res["modes"])
print("Cost:", res["cost"])
print("Labels:", res["labels"])
```

B. Production-ready library (quick & robust)

Install once: `pip install kmodes`

```python
from kmodes.kmodes import KModes

km = KModes(n_clusters=2, init='Cao', n_init=10, max_iter=100, random_state=42)
labels = km.fit_predict([
 ["S","Slim","Red","Cotton"],
 ["S","Slim","Blue","Cotton"],
 ["M","Regular","Blue","Cotton"],
 ["L","Regular","Blue","Polyester"],
 ["L","Regular","Green","Polyester"],
 ["M","Slim","Red","Cotton"],
 ["M","Regular","Blue","Polyester"],
 ["L","Regular","Blue","Cotton"],
])
print("Cluster modes:", km.cluster_centroids_)  # modes per feature
print("Labels:", labels)
print("Cost:", km.cost_)
```

- `init='Huang'` or `init='Cao'` are strong defaults.
- Use `n_init>1` to reduce sensitivity to starting seeds.
- New points can be assigned via `kmodes.predict(new_samples)`.

Interpreting results

- **Modes tell the story**: each mode is a “typical profile” for the cluster (e.g., **(L, Regular, Blue, Polyester)** = stock keeping segment).
- **Within-cluster mismatch** is the natural fit score: lower is better.
- **Per-feature mismatch rates** help diagnose which attributes drive separation.

Common pitfalls (and quick fixes)

- **Many rare categories** → collapse to "Other" or group by domain ontology.
- **Dominant features** → apply **feature weights** (e.g., weight by inverse cardinality or business importance).
- **Unbalanced clusters** → consider different k, better initialization (Cao), or feature weights.
- **Mixed numeric/categorical** → use **k-prototypes** instead of forcing everything to categories.
- **Inconsistent preprocessing** → ensure the **same** cleaning, category mapping, and missing-value strategy for train & inference.

Imp points

- ☐ Categorical data only? (If mixed, use k-prototypes.)
- ☐ Clean categories, handle missing, map rare to "Other".
- ☐ Pick k via elbow/validation/stability.
- ☐ Use Huang/Cao init and multiple restarts.
- ☐ Inspect modes to explain clusters; report total cost and per-feature mismatch rates.

Further reading (for depth)

- Z. Huang (1998). "Extensions to the k-means algorithm for clustering large data sets with categorical values." *Data Mining and Knowledge Discovery*.
- Z. Huang (1997). "Clustering large data sets with mixed numeric and categorical values." *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining* (k-prototypes).

Full worked out example of K Modes clustering for multiple iterations

---

K-Modes Worked Example with Multiple Iterations

We'll use the same **toy dataset**:

ID	Size	Fit	Color	Fabric
1	S	Slim	Red	Cotton
2	S	Slim	Blue	Cotton
3	M	Regular	Blue	Cotton
4	L	Regular	Blue	Polyester
5	L	Regular	Green	Polyester
6	M	Slim	Red	Cotton
7	M	Regular	Blue	Polyester
8	L	Regular	Blue	Cotton

We'll set **k = 2 clusters**.

Step 1. Initialization (choose 2 random seeds)

Let's pick:

- **$Mode_1$ (initial)** ← Row 2 = (S, Slim, Blue, Cotton)
- **$Mode_2$ (initial)** ← Row 5 = (L, Regular, Green, Polyester)

These are less "clean," so convergence takes longer.

Step 2. First Assignment

Compute mismatches for each row vs each mode:

ID	Row (Size, Fit, Color, Fabric)	d to $Mode_1$ (S,Slim,Blue,Cotton)	d to $Mode_2$ (L,Regular,Green,Poly)	Assign
1	(S,Slim,Red,Cotton)	1 (Color)	4	C_1
2	(S,Slim,Blue,Cotton)	0	4	C_1
3	(M,Reg,Blue,Cotton)	2 (Size,Fit)	2 (Size,Fabric)	C_1 (tie→C_1)
4	(L,Reg,Blue,Poly)	2 (Size,Fabric)	1 (Color)	C_2
5	(L,Reg,Green,Poly)	3	0	C_2
6	(M,Slim,Red,Cotton)	2 (Size,Color)	4	C_1
7	(M,Reg,Blue,Poly)	2 (Size,Fabric)	1 (Color)	C_2
8	(L,Reg,Blue,Cotton)	2 (Size,Fabric)	1 (Color)	C_2

Clusters after Iteration 1:

- C_1 = {1,2,3,6}
- C_2 = {4,5,7,8}

Step 3. Update Modes

- **$Mode_1$ (C_1 rows 1,2,3,6):**
  - Size: S(2), M(2) → tie → pick **S** (tie-break fixed order)
  - Fit: Slim(3), Reg(1) → **Slim**
  - Color: Red(2), Blue(2) → tie → pick **Blue**
  - Fabric: Cotton(4) → **Cotton**

    → $Mode_1'$ = (S, Slim, Blue, Cotton)
- **$Mode_2$ (C_2 rows 4,5,7,8):**
  - Size: L(3), M(1) → **L**
  - Fit: Reg(4) → **Regular**

- Color: Blue(3), Green(1) → **Blue**
- Fabric: Polyester(2), Cotton(2) → tie → pick **Polyester**
  → $Mode_2'$ = (L, Regular, Blue, Polyester)

Step 4. Second Assignment

Now compare all rows again vs updated modes:

ID	Row	**d to $Mode_1'$ (S,Slim,Blue,Cotton)**	**d to $Mode_2'$ (L,Reg,Blue,Poly)**	Assign
1	(S,Slim,Red,Cotton)	1 (Color)	3 (Size,Fit,Fabric)	C_1
2	(S,Slim,Blue,Cotton)	0	3 (Size,Fit,Fabric)	C_1
3	(M,Reg,Blue,Cotton)	2 (Size,Fit)	1 (Fabric)	C_2
4	(L,Reg,Blue,Poly)	3 (Size,Fit,Fabric)	0	C_2
5	(L,Reg,Green,Poly)	4	1 (Color)	C_2
6	(M,Slim,Red,Cotton)	2 (Size,Color)	3 (Size,Fabric,Color)	C_1
7	(M,Reg,Blue,Poly)	3 (Size,Fit,Fabric)	1 (Size)	C_2
8	(L,Reg,Blue,Cotton)	2 (Size,Fit)	1 (Fabric)	C_2

Clusters after Iteration 2:

- C_1 = {1,2,6}
- C_2 = {3,4,5,7,8}

Step 5. Update Modes Again

- **$Mode_1$ (rows 1,2,6):**
  - Size: S(2), M(1) → **S**
  - Fit: Slim(3) → **Slim**
  - Color: Red(2), Blue(1) → **Red**

- Fabric: Cotton(3) → **Cotton**

  → $Mode_1''$ = (S, Slim, Red, Cotton)

- **$Mode_2$ (rows 3,4,5,7,8):**
  - Size: L(3), M(2) → **L**
  - Fit: Reg(5) → **Regular**
  - Color: Blue(4), Green(1) → **Blue**
  - Fabric: Polyester(3), Cotton(2) → **Polyester**

    → $Mode_2''$ = (L, Regular, Blue, Polyester)

Step 6. Third Assignment

Check again with new $Mode_1''$ and $Mode_2''$:

ID	Row	d to $Mode_1''$ (S,Slim,Red,Cotton)	d to $Mode_2''$ (L,Reg,Blue,Poly)	Assign
1	(S,Slim,Red,Cotton)	0	4	C_1
2	(S,Slim,Blue,Cotton)	1 (Color)	3 (Size,Fit,Fabric)	C_1
3	(M,Reg,Blue,Cotton)	3 (Size,Fit,Color)	1 (Fabric)	C_2
4	(L,Reg,Blue,Poly)	4	0	C_2
5	(L,Reg,Green,Poly)	3	1 (Color)	C_2
6	(M,Slim,Red,Cotton)	1 (Size)	4	C_1
7	(M,Reg,Blue,Poly)	4	1 (Size)	C_2
8	(L,Reg,Blue,Cotton)	3 (Size,Fit,Color)	1 (Fabric)	C_2

Clusters after Iteration 3:

- C_1 = {1,2,6}
- C_2 = {3,4,5,7,8}

Step 7. Update Modes

- **$Mode_1$ stays (S,Slim,Red,Cotton)**
- **$Mode_2$ stays (L,Regular,Blue,Polyester)**

→ **No change** → algorithm **converged** after **3 iterations**.

Final Results

- **Final modes (cluster representatives):**
  - Cluster 1 → (S, Slim, Red, Cotton)
  - Cluster 2 → (L, Regular, Blue, Polyester)
- **Cluster memberships:**
  - C_1 = {1,2,6}
  - C_2 = {3,4,5,7,8}
- **Total cost:**
  - C_1 mismatches = 0 + 1 + 1 = **2**
  - C_2 mismatches = 1 + 0 + 1 + 1 + 1 = **4**
  - **Total = 6 mismatches**