Assignment 4 BUDA 451

Collin Edwards

2025-04-24

Question 1: Similarity and Distance

(a) Hamming Distance and Jaccard Similarity

```
import numpy as np

# define the two binary vectors
x = np.array([0,1,0,1,0,1,0,0,0,1])
y = np.array([0,1,0,0,0,1,1,0,0,0])

# (a) Hamming distance = number of positions where they differ
hamming = np.sum(x != y)

# (a) Jaccard similarity = |intersection| / |union|
intersection = np.sum((x == 1) & (y == 1))
union        = np.sum((x == 1) | (y == 1))
jaccard = intersection / union if union else 0

print(f"Hamming distance: {hamming}")
print(f"Jaccard similarity: {jaccard:.4f}")
```

```
Hamming distance: 3
Jaccard similarity: 0.4000
```

(1b) Choice of Similarity Measure

For very sparse purchase-history vectors at Amazon, we prefer measures that *ignore double-zeros*, so that unpurchased items don't dominate the similarity.
- **Jaccard coefficient** (and similarly **Cosine similarity**) focus only on co-purchases (1's), whereas Hamming or Simple Matching count zeros and can be misleading.

Question 2: k-Means Clustering

(a) k-Means Algorithm Description

k-Means is an iterative algorithm for partitioning data into k clusters. It minimizes the within-cluster variance by: 1. Initialize K centroids randomly. 2. Assign each point to nearest centroid by Euclidean distance. 3. Update each centroid to the mean of its assigned points. 4. Repeat steps 2-3 until centroids stabilize.

(b) Identifying the k-Means Output=

k-means produces compact, roughly spherical clusters (Voronoi partitions).
- **Dataset A (two clusters):**
- **A2** is k-means (clusters are round and balanced).
- A1 shows a non-convex shape, which k-means cannot capture.
- **Dataset B (two clusters):**
- **B1** is k-means (clusters look round).
- B2 has elongated shapes, not typical for k-means.
- **Dataset C (three clusters):**
- **C1** is k-means (three round, well-separated groups).
- C2 splits one natural group, which k-means avoids when clusters are distinct.

Question 3: Hierarchical Clustering

We have clusters A and B, each with four 2-D points. The Euclidean distance is

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}.$$

- **Complete Link (farthest pair):**

$$(\max\_a \in A, b \in Bd(a, b) \approx 2.1090)$$

- **Single Link (closest pair):**

$$(\min\_a \in A, b \in Bd(a, b) \approx 0.9220)$$

- **Average Link (mean of all 16 distances):**

$$\frac{1}{16}\sum_{a\in A, b\in B} d(a,b) \approx 1.4127)$$

- **Centroid Link (distance between centroids):**

$$d(\frac{1}{4}\sum_{a\in A} a, \frac{1}{4}\sum_{b\in B} b) \approx 1.4142)$$

Robustness to Noise: - Single link can drop too low if one pair is accidentally very close.

- Complete link can spike if one pair is an outlier.

- **Average link** smooths out extremes and is the most robust.

Question 4: Programming (USArrests)

We load and normalize the USArrests data, perform K-means, use the Elbow method, and build dendrograms for three linkage methods.

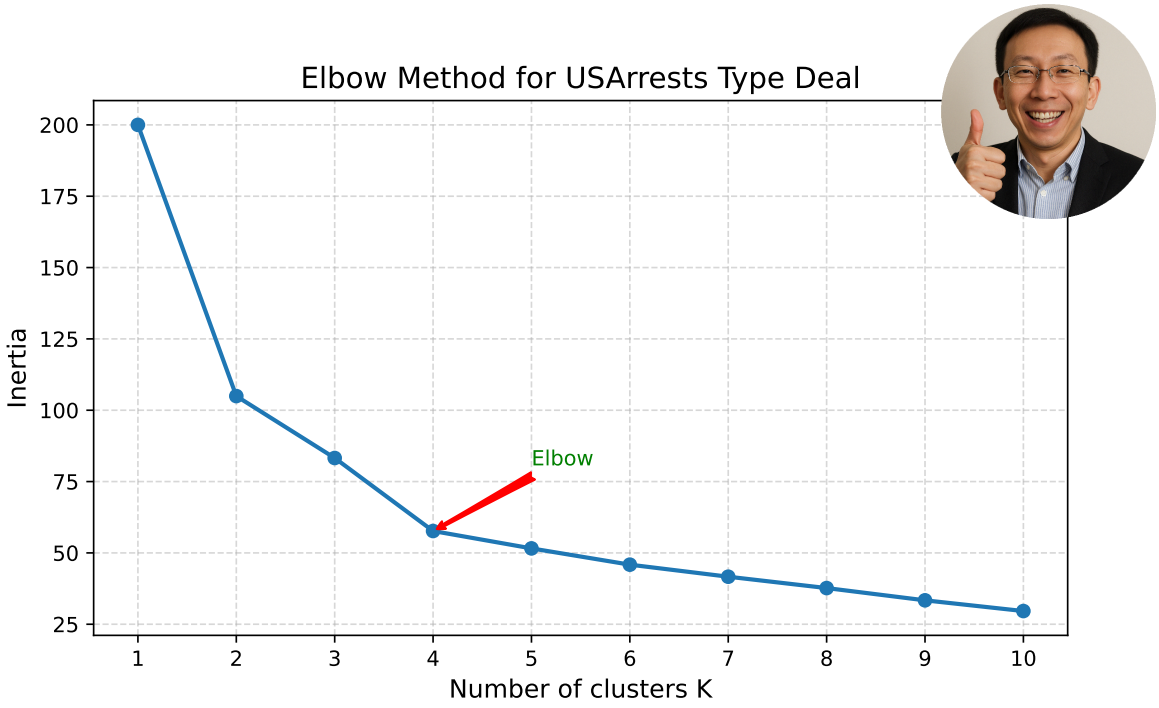
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage, dendrogram



# 1) Load the data
url = "https://raw.githubusercontent.com/binbenliu/Teaching/main/data/USArrests.txt"
df = pd.read_csv(url, index_col=0)

# 2) Normalize the features
features = ["Murder", "Assault", "UrbanPop", "Rape"]
X = StandardScaler().fit_transform(df[features])
```

```
# 3) K-means with K=3 (just to show we can)
k3 = KMeans(n_clusters=3, random_state=42).fit(X)

# instead of print(...), do
centers = pd.DataFrame(
    k3.cluster_centers_,
    columns=features
)
centers

# 4) Compute inertias for K=1..10
inertias = [KMeans(n_clusters=k, random_state=42).fit(X).inertia_
            for k in range(1, 11)]

# 5) Plot the Elbow curve and inset a small logo
fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(range(1, 11), inertias, 'o-', linewidth=2, markersize=6)
ax.set_xticks(range(1, 11))
ax.set_xlabel("Number of clusters K", fontsize=12)
ax.set_ylabel("Inertia", fontsize=12)
ax.set_title("Elbow Method for USArrests Type Deal", fontsize=14)
ax.grid(True, linestyle='--', alpha=0.5)

# annotate the "elbow"
opt_k = 4
ax.annotate("Elbow",
            xy=(opt_k, inertias[opt_k-1]),
            xytext=(opt_k+1, inertias[opt_k-1] + 23),
            arrowprops=dict(arrowstyle='fancy', color='red'),
            color='green')

#    lines below are inserting the logo as a perfect circle

# 1) load the logo
img = mpimg.imread("images/logo1.png")

# 2) ensure RGBA
h, w = img.shape[:2]
if img.shape[2] == 3:
    rgba = np.zeros((h, w, 4), dtype=img.dtype)
    rgba[..., :3] = img
    rgba[...,  3] = 255
```

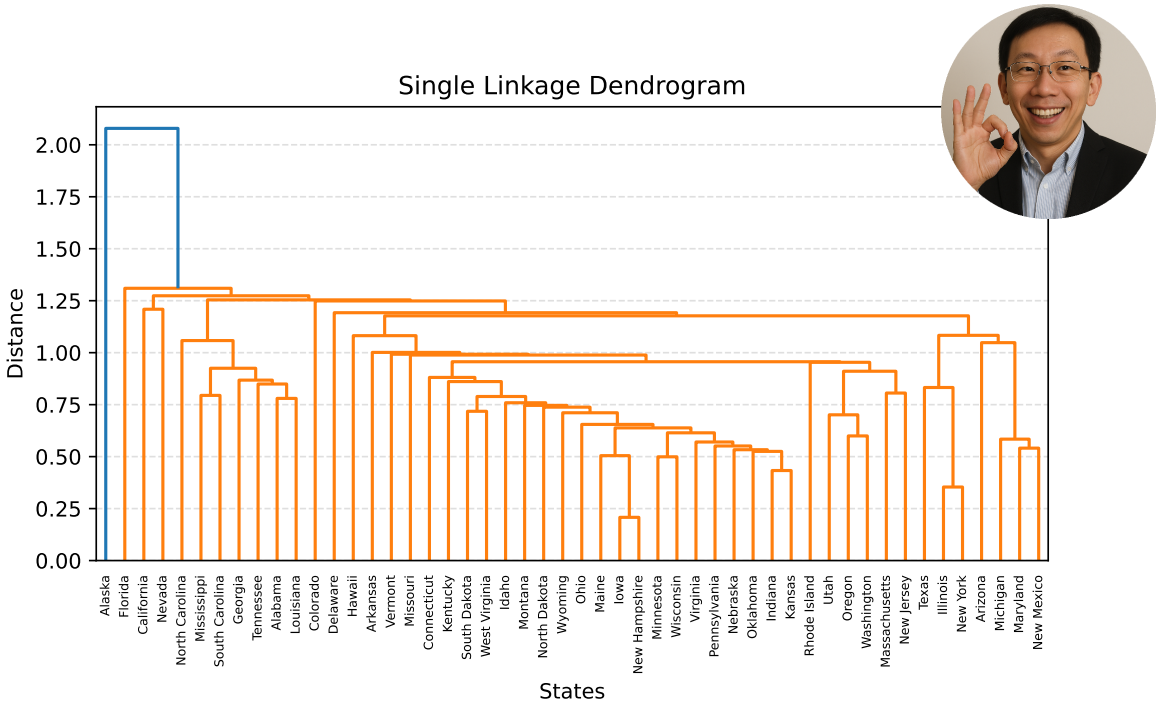
```
else:
    rgba = img.copy()

# 3) build a circular mask
yy, xx = np.ogrid[:h, :w]
cy, cx = h/2, w/2
r = min(h, w) / 2
mask = (yy - cy)**2 + (xx - cx)**2 <= r**2

# 4) apply mask to alpha channel
rgba[..., 3] = rgba[..., 3] * mask

# 5) create and add the clipped image
imagebox = OffsetImage(rgba, zoom=0.10)
ab = AnnotationBbox(
    imagebox,
    (0.98, 0.98),
    xycoords='axes fraction',
    frameon=False
)
ax.add_artist(ab)

plt.tight_layout()
plt.show()

# 6) Agglomerative clustering dendrograms

# 1) Read the PNG
logo = mpimg.imread("images/logo6.png")
h, w = logo.shape[:2]

# 2) Ensure RGBA
if logo.shape[2] == 3:
    rgba = np.zeros((h, w, 4), dtype=logo.dtype)
    rgba[...,:3] = logo
    rgba[..., 3]  = 255
else:
    rgba = logo.copy()

# 3) Build a circular mask
yy, xx = np.ogrid[:h, :w]
cy, cx = h/2, w/2
```

```
r = min(h, w)/2
mask = (yy - cy)**2 + (xx - cx)**2 <= r**2
rgba[..., 3] *= mask  # mask alpha channel

# -------------------------

for method in ["single", "complete", "average"]:
    Z = linkage(X, method=method)
    fig, ax = plt.subplots(figsize=(8, 5))
    dendrogram(Z, labels=df.index,
               leaf_rotation=90, leaf_font_size=6,
               ax=ax)
    ax.set_title(f"{method.title()} Linkage Dendrogram", fontsize=12)
    ax.set_xlabel("States", fontsize=10)
    ax.set_ylabel("Distance", fontsize=10)
    ax.grid(axis='y', linestyle='--', alpha=0.4)

    # - insert circular logo into this axes -
    imagebox = OffsetImage(rgba, zoom=0.10)
    ab = AnnotationBbox(
        imagebox,
        (1, 0.99),            # top-right in axes fraction coords
        xycoords='axes fraction',
        frameon=False
    )
    ax.add_artist(ab)

    plt.tight_layout()
    plt.show()
```

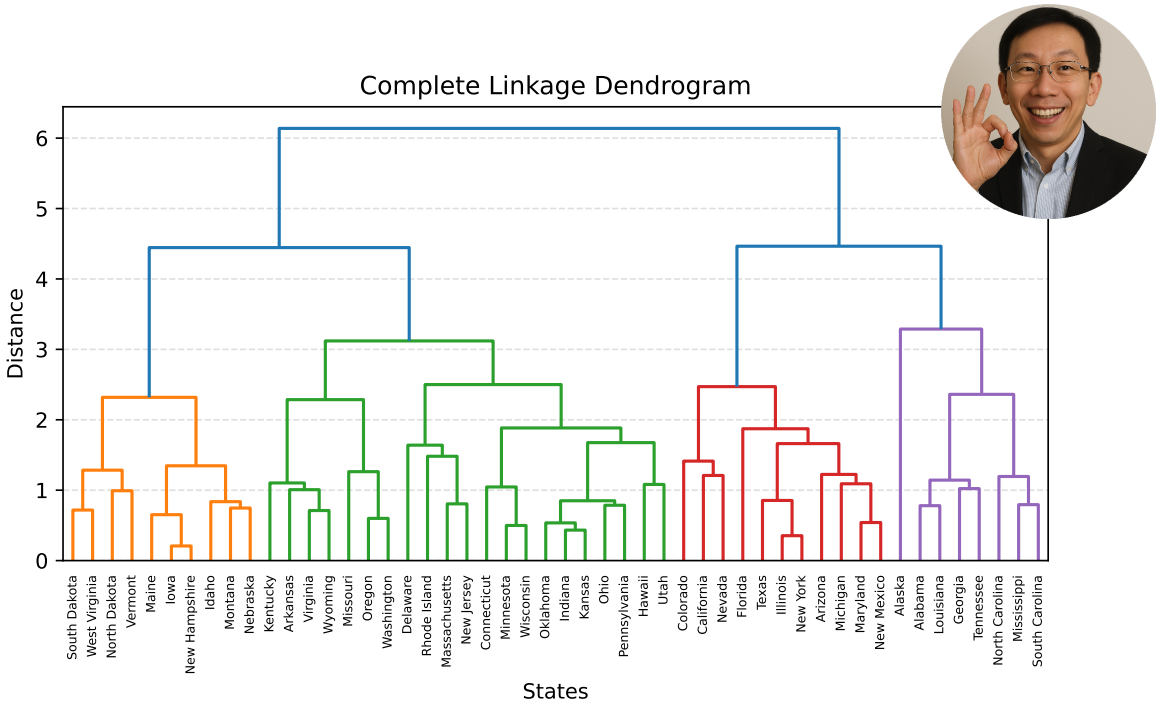
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255
```



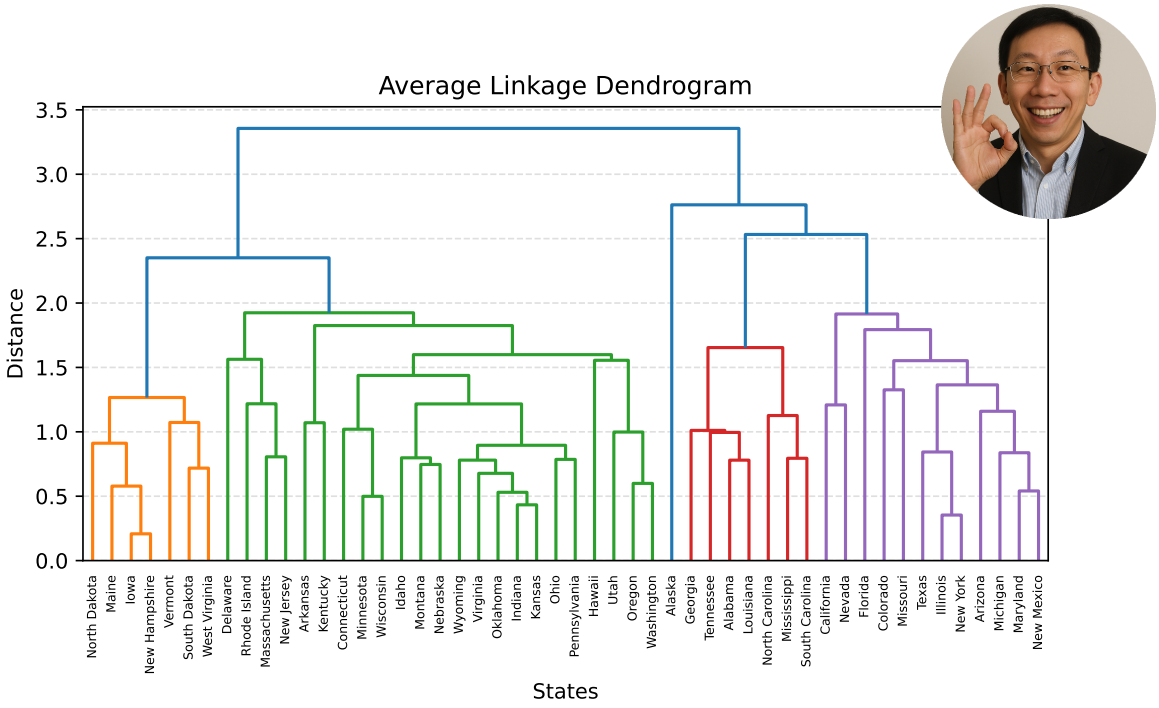

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255
```



Illustrated above are the dendrograms for the three linkage methods: single, complete, and average. Each dendrogram shows how the states cluster based on their arrest rates, with the circular logo inset in the top right corner. It's important to note that the choice of linkage method can significantly affect the clustering results, as seen in the different shapes and heights of the branches in each dendrogram.