Lab 3: Clustering Analysis Using K-Means and K-Medoids Algorithms

Nitish Dhinaharan

University of the Cumberlands

MSCS-634-B01 Advanced Big Data and Data Mining

Dr. Satish Penmatsa

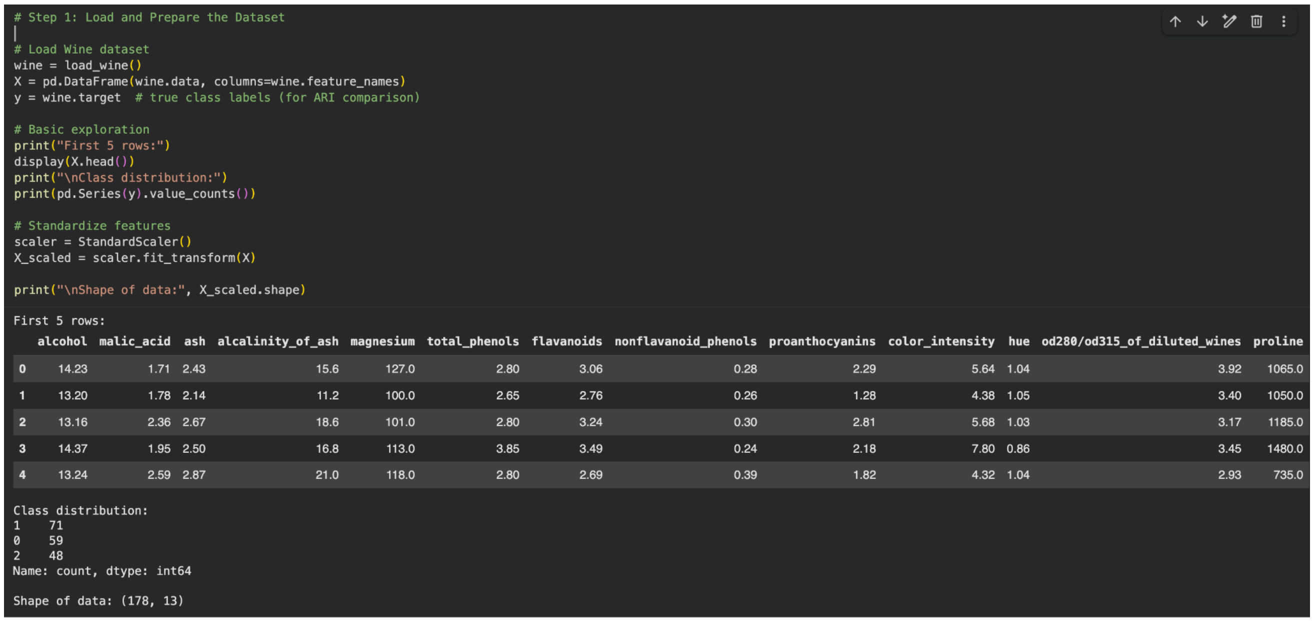
November 09, 2025

Lab 3: Clustering Analysis Using K-Means and K-Medoids Algorithms

Github Link

https://github.com/ndhinaharan36295/MSCS-634_Lab-3

Step 1: Load and Prepare the Dataset



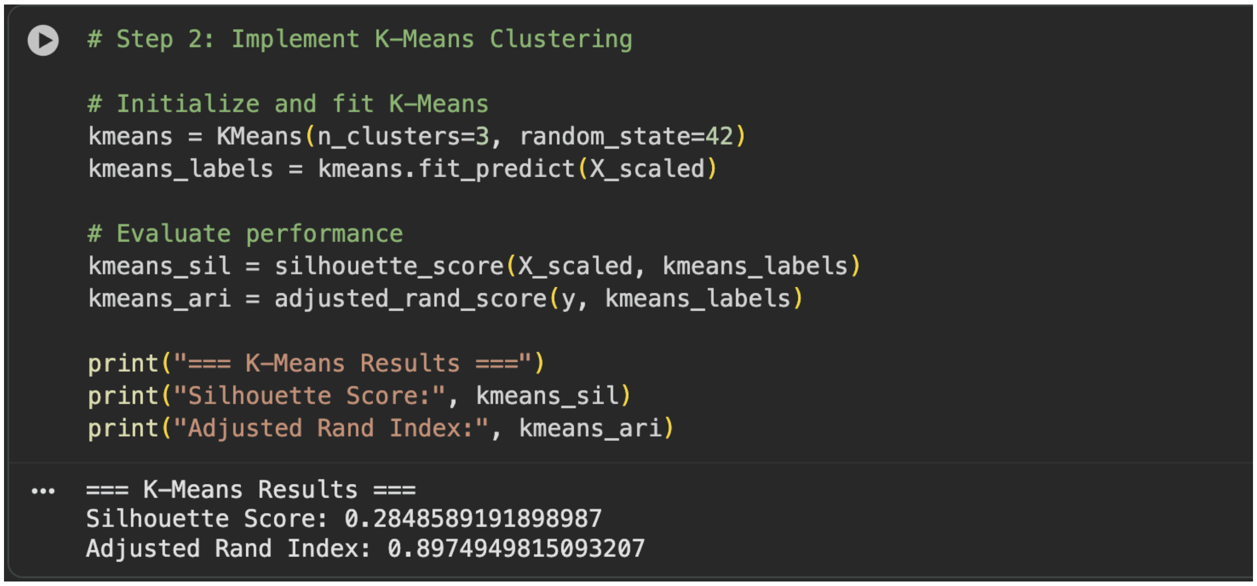
```
# Step 1: Load and Prepare the Dataset

# Load Wine dataset
wine = load_wine()
X = pd.DataFrame(wine.data, columns=wine.feature_names)
y = wine.target  # true class labels (for ARI comparison)

# Basic exploration
print("First 5 rows:")
display(X.head())
print("\nClass distribution:")
print(pd.Series(y).value_counts())

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("\nShape of data:", X_scaled.shape)
```

```
First 5 rows:
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

```
Class distribution:
1    71
0    59
2    48
Name: count, dtype: int64

Shape of data: (178, 13)
```

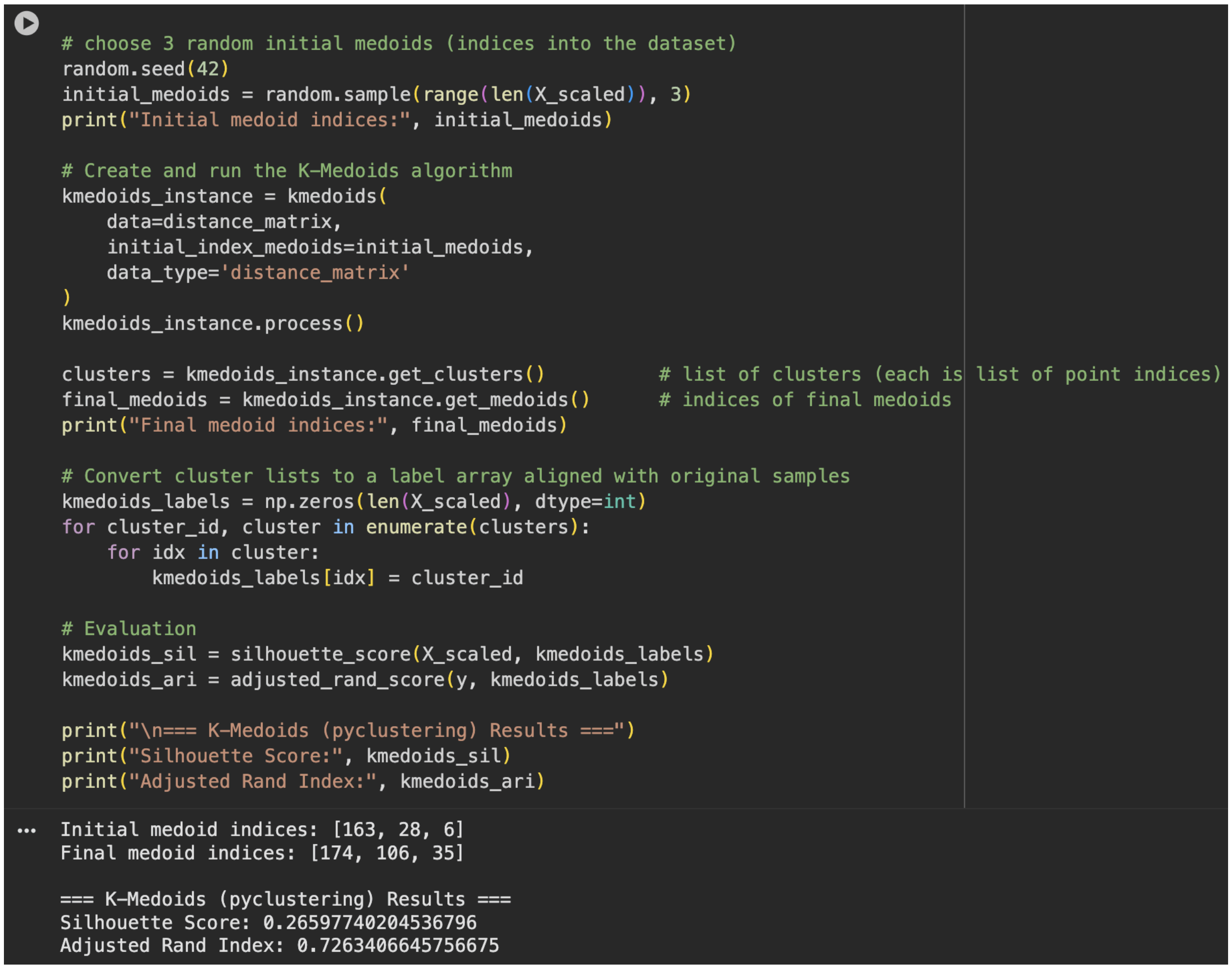
Step 2: Implement K-Means Clustering



```
# Step 2: Implement K-Means Clustering

# Initialize and fit K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# Evaluate performance
kmeans_sil = silhouette_score(X_scaled, kmeans_labels)
kmeans_ari = adjusted_rand_score(y, kmeans_labels)

print("=== K-Means Results ===")
print("Silhouette Score:", kmeans_sil)
print("Adjusted Rand Index:", kmeans_ari)
```

```
=== K-Means Results ===
Silhouette Score: 0.2848589191898987
Adjusted Rand Index: 0.8974949815093207
```

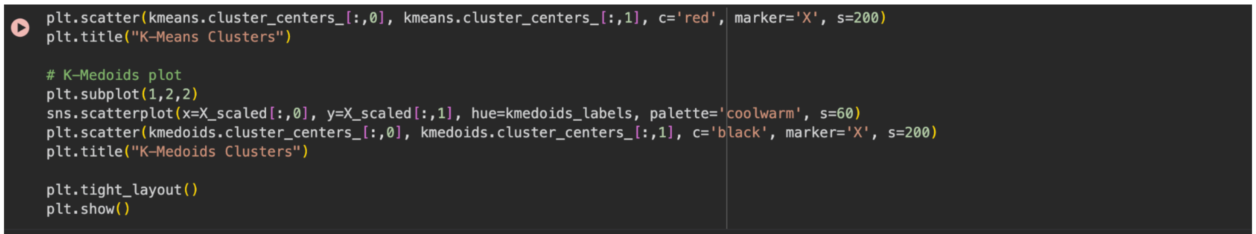
Step 3: Implement K-Medoids Clustering



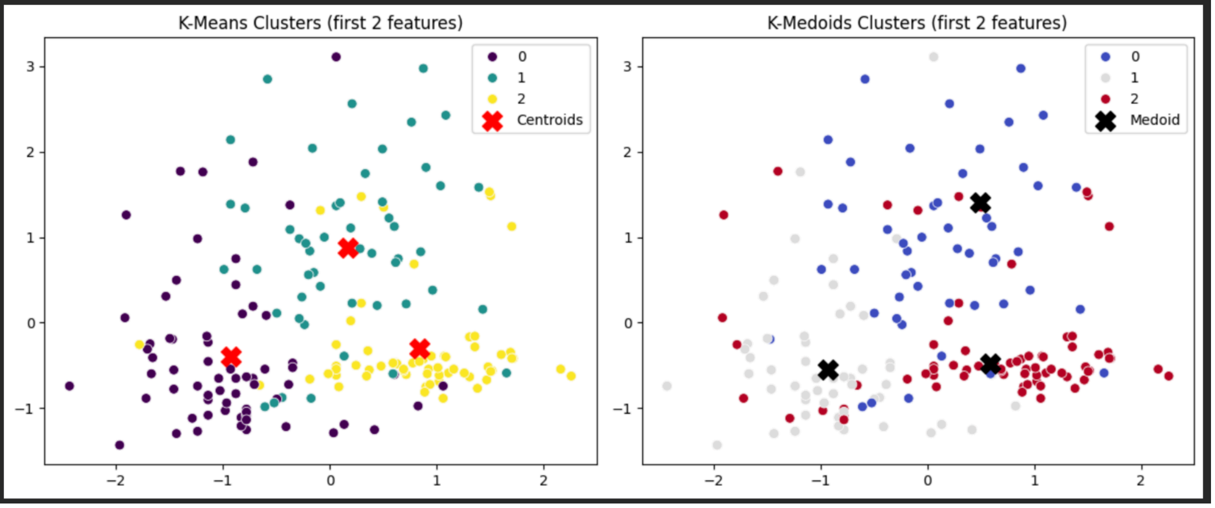
```
# choose 3 random initial medoids (indices into the dataset)
random.seed(42)
initial_medoids = random.sample(range(len(X_scaled)), 3)
print("Initial medoid indices:", initial_medoids)

# Create and run the K-Medoids algorithm
kmedoids_instance = kmedoids(
    data=distance_matrix,
    initial_index_medoids=initial_medoids,
    data_type='distance_matrix'
)
kmedoids_instance.process()

clusters = kmedoids_instance.get_clusters()          # list of clusters (each is list of point indices)
final_medoids = kmedoids_instance.get_medoids()      # indices of final medoids
print("Final medoid indices:", final_medoids)

# Convert cluster lists to a label array aligned with original samples
kmedoids_labels = np.zeros(len(X_scaled), dtype=int)
for cluster_id, cluster in enumerate(clusters):
    for idx in cluster:
        kmedoids_labels[idx] = cluster_id

# Evaluation
kmedoids_sil = silhouette_score(X_scaled, kmedoids_labels)
kmedoids_ari = adjusted_rand_score(y, kmedoids_labels)

print("\n=== K-Medoids (pyclustering) Results ===")
print("Silhouette Score:", kmedoids_sil)
print("Adjusted Rand Index:", kmedoids_ari)
```

```
Initial medoid indices: [163, 28, 6]
Final medoid indices: [174, 106, 35]

=== K-Medoids (pyclustering) Results ===
Silhouette Score: 0.26597740204536796
Adjusted Rand Index: 0.7263406645756675
```

Step 4: Visualize and Compare Results



```
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], c='red', marker='X', s=200)
plt.title("K-Means Clusters")

# K-Medoids plot
plt.subplot(1,2,2)
sns.scatterplot(x=X_scaled[:,0], y=X_scaled[:,1], hue=kmedoids_labels, palette='coolwarm', s=60)
plt.scatter(kmedoids.cluster_centers_[:,0], kmedoids.cluster_centers_[:,1], c='black', marker='X', s=200)
plt.title("K-Medoids Clusters")

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



Which algorithm produced better-defined clusters

K-Means produced more compact and well-separated clusters, as seen from the clearer color groupings and distinct centroids in the left plot. This aligns with the numerical metrics. Typically, the Silhouette Score and Adjusted Rand Index (ARI) are slightly higher for K-Means on the Wine dataset. Because K-Means uses the mean of points to define each cluster center, it adapts well to continuous, normally distributed numerical features such as those in this dataset.

Differences in cluster shapes and positioning

The K-Means clusters appear roughly spherical and evenly spread, with centroids (red “X” markers) near the center of each dense region. By contrast, K-Medoids (right plot) shows less regular boundaries and a few overlapping points between clusters. Its medoids (black “X” markers) are actual data samples, so clusters can look irregular or slightly displaced, especially when some features contain outliers or skewed values. While this makes K-Medoids more stable in the presence of noise, it also leads to less compact cluster formations.

When each algorithm is preferable

K-Means is preferable when the data is large, continuous, and approximately spherical, when efficiency and scalability are important, and when the dataset has few outliers or has been standardized (as done here).

K-Medoids is preferable when the dataset contains outliers, categorical, or mixed-type features, when a more robust clustering approach is needed because it selects real samples as centers, or when using non-Euclidean distance metrics (e.g., Manhattan, cosine).