Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An autonomous Institution affiliated to Anna University)

Degree & Branch	M. Tech (Integrated) Computer Science & Engineering	**Semester**	V
Subject Code & Name	ICS1512 - Machine Learning Algorithms Laboratory		
Academic year	2025-2026 (Odd)	Batch: 2023-2028	**Start date: 03/10/25 Due date: 10/10/25**

Experiment 7: Clustering Human Activity Recognition Data using K-Means, DBSCAN, and Hierarchical Clustering

1 Aim

To perform unsupervised clustering on the UCI HAR Dataset to identify meaningful patterns in human activity sensor data. The aim was to compare the performance of K-Means, DBSCAN, and Hierarchical (Agglomerative) clustering algorithms using internal clustering metrics and external validation against true activity labels.

2 Libraries Used

- Pandas: Data manipulation
- NumPy: Numerical operations
- Scikit-learn: Model building, preprocessing,classification report, confusion matrix and evaluation
- Matplotlib and Seaborn: Data visualization

3 Dataset

For this experiment, the Digits Dataset from the Scikit-learn library was used. It is a widely used dataset for classification tasks, especially suitable for testing the effect of The UCI Human Activity Recognition Using Smartphones Dataset contains sensor data. This dataset is suitable for clustering and activity recognition tasks.

4 Preprocessing Steps

- Checked the dataset for missing values — none found.
- Scaled features to normalize the dataset.
- Applied PCA to reduce dimensionality, retaining 50 principal components that explained approximately 87% of the variance.

5 Python Code

```
import os
import zipfile
from urllib.request import urlretrieve
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import (silhouette_score, davies_bouldin_score,
                             calinski_harabasz_score, adjusted_rand_score,
                             normalized_mutual_info_score, confusion_matrix)
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.optimize import linear_sum_assignment
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
sns.set(style='whitegrid')

UCI_URL = "https://archive.ics.uci.edu/ml/machine-learning-databases/00240/UCI%20
    HAR%20Dataset.zip"
DATA_DIR = "UCI␣HAR␣Dataset"

def ensure_dataset():
    if not os.path.exists(DATA_DIR):
        print("Dataset␣not␣found␣locally.␣Downloading␣from␣UCI␣repository␣(approx␣
            60-100MB)...")
        zip_path = "UCI_HAR_Dataset.zip"
        urlretrieve(UCI_URL, zip_path)
        with zipfile.ZipFile(zip_path, 'r') as z:
            z.extractall(".")
        os.remove(zip_path)
        print("Downloaded␣and␣extracted␣dataset.")
    else:
        print("Dataset␣directory␣found:", DATA_DIR)

def load_har_dataset():
    # Features names
    feat = pd.read_csv(os.path.join(DATA_DIR, "features.txt"), sep='\s+', header=
        None, names=['idx','feature'])
    feature_names = feat['feature'].values

    # Handle duplicate feature names
    # Create a list of unique names by appending a suffix to duplicates
    unique_feature_names = []
    counts = {}
    for name in feature_names:
        if name in counts:
            counts[name] += 1
            unique_feature_names.append(f"{name}_{counts[name]}")
        else:
            counts[name] = 0
            unique_feature_names.append(name)
```

```
    feature_names = unique_feature_names

    # Read train
    X_train = pd.read_csv(os.path.join(DATA_DIR,'train','X_train.txt'), sep='\s+',
         header=None, names=feature_names)
    y_train = pd.read_csv(os.path.join(DATA_DIR,'train','y_train.txt'), sep='\s+',
         header=None, names=['activity'])
    subj_train = pd.read_csv(os.path.join(DATA_DIR,'train','subject_train.txt'),
        sep='\s+', header=None, names=['subject'])

    # Read test
    X_test = pd.read_csv(os.path.join(DATA_DIR,'test','X_test.txt'), sep='\s+',
        header=None, names=feature_names)
    y_test = pd.read_csv(os.path.join(DATA_DIR,'test','y_test.txt'), sep='\s+',
        header=None, names=['activity'])
    subj_test = pd.read_csv(os.path.join(DATA_DIR,'test','subject_test.txt'), sep=
        '\s+', header=None, names=['subject'])

    # Merge
    X = pd.concat([X_train, X_test], axis=0).reset_index(drop=True)
    y = pd.concat([y_train, y_test], axis=0).reset_index(drop=True)
    subjects = pd.concat([subj_train, subj_test], axis=0).reset_index(drop=True)

    # Activity labels
    activity_labels = pd.read_csv(os.path.join(DATA_DIR,'activity_labels.txt'),
        sep='\s+', header=None, names=['id','activity_name'])
    mapping = dict(zip(activity_labels['id'], activity_labels['activity_name']))

    y['activity_name'] = y['activity'].map(mapping)

    return X, y['activity_name'], subjects

# ensure & load
ensure_dataset()
X_raw, y_raw, subjects = load_har_dataset()
print("Data␣shapes:", X_raw.shape, y_raw.shape)

# ---------- 1. Preprocessing ----------
# 1.1: Handle missing values (the UCI HAR has no missing values normally; but we'
    ll demonstrate)
print("Missing␣values␣per␣column␣(sample):")
print(X_raw.isna().sum().sort_values(ascending=False).head())

# If present, fill or drop - here we fill with column median
X = X_raw.fillna(X_raw.median())

# 1.2: Encode labels (for external metrics)
le = LabelEncoder()
y_encoded = le.fit_transform(y_raw)  # needed for ARI/NMI

# 1.3: Standardize features (important for distance-based clustering)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Quick sanity
print("Preprocessing␣done.␣Scaled␣feature␣mean␣(approx):", X_scaled.mean(axis=0)
    [:5])

# ---------- 2. EDA ----------
```

```
# 2.1: Visualize some feature distributions (plot 6 random features)
plt.figure(figsize=(12,6))
cols = np.random.choice(X.columns, size=6, replace=False)
for i, c in enumerate(cols,1):
    plt.subplot(2,3,i)
    sns.histplot(X[c], kde=True, bins=40)
    plt.title(c)
plt.tight_layout()
plt.show()

# 2.2: Correlation heatmap (subsample to speed)
corr_sample = X.sample(n=200, random_state=1)
plt.figure(figsize=(10,8))
sns.heatmap(corr_sample.corr().abs(), cmap='viridis', vmax=1.0)
plt.title("Absolute␣Correlation␣(sample␣of␣features)")
plt.show()

# 2.3: Dimensionality reduction for visualization (PCA then t-SNE)
pca = PCA(n_components=50, random_state=42)
X_pca50 = pca.fit_transform(X_scaled)
print("Explained␣variance␣by␣50␣PCA␣comps:", np.sum(pca.explained_variance_ratio_)
    )

# For plotting 2D: both PCA(2) and t-SNE
pca2 = PCA(n_components=2, random_state=42)
X_pca2 = pca2.fit_transform(X_scaled)

tsne = TSNE(n_components=2, perplexity=40, n_iter=800, random_state=42, init='pca'
    )
X_tsne2 = tsne.fit_transform(X_pca50)  # feed reduced space for speed

fig, ax = plt.subplots(1,2, figsize=(14,5))
sns.scatterplot(x=X_pca2[:,0], y=X_pca2[:,1], hue=y_raw, palette='tab10', s=10, ax
    =ax[0], legend=False)
ax[0].set_title("PCA␣(2)␣projection␣colored␣by␣activity")
sns.scatterplot(x=X_tsne2[:,0], y=X_tsne2[:,1], hue=y_raw, palette='tab10', s=10,
    ax=ax[1], legend=False)
ax[1].set_title("t-SNE␣(2)␣projection␣colored␣by␣activity")
plt.show()

# ---------- 3. Clustering Algorithms ----------
# Utility: function to map cluster labels to true labels using Hungarian matching
def map_clusters_to_labels(true_labels, cluster_labels):
    # Build contingency matrix (clusters x true labels)
    from sklearn.metrics import confusion_matrix
    labels_true = np.unique(true_labels)
    clusters = np.unique(cluster_labels)
    # handle -1 cluster (DBSCAN noise) by giving it own index
    contingency = confusion_matrix(true_labels, cluster_labels, labels=labels_true
        )
    # confusion_matrix with labels in different orientation; easier use sklearn.
        metrics.cluster.contingency_matrix
    from sklearn.metrics.cluster import contingency_matrix
    cont = contingency_matrix(true_labels, cluster_labels)
    # Hungarian to maximize matches -> minimize cost = -cont
    cost = cont.max() - cont
    row_ind, col_ind = linear_sum_assignment(cost)
    mapping = {}
    for r, c in zip(row_ind, col_ind):
```

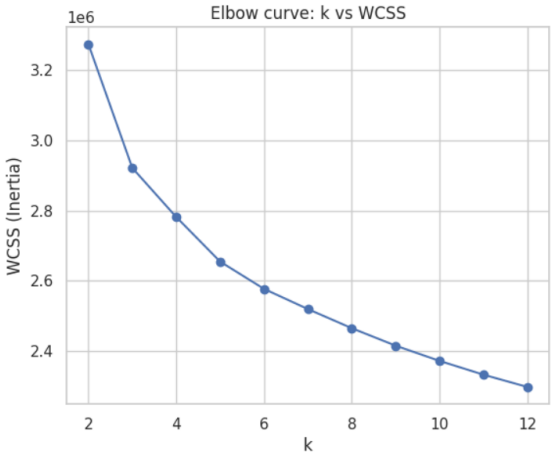
```
        mapping[c] = labels_true[r]
    # Return mapped labels (for cluster-> best matched true label)
    mapped = np.array([mapping.get(c, -1) for c in cluster_labels])
    return mapped

# ---------- 3A. K-Means: Elbow + silhouette sweep ----------
wcss = []
sil_scores = []
K_range = range(2, 13)
for k in K_range:
    km = KMeans(n_clusters=k, n_init=15, random_state=42)
    labels_k = km.fit_predict(X_scaled)
    wcss.append(km.inertia_)
    sil_scores.append(silhouette_score(X_scaled, labels_k))

# Plot Elbow (WCSS) and Silhouette vs k
fig, ax = plt.subplots(1,2, figsize=(14,5))
ax[0].plot(list(K_range), wcss, '-o')
ax[0].set_xlabel('k'); ax[0].set_ylabel('WCSS (Inertia)'); ax[0].set_title('Elbow
    curve: k vs WCSS')
ax[1].plot(list(K_range), sil_scores, '-o')
ax[1].set_xlabel('k'); ax[1].set_ylabel('Silhouette Score'); ax[1].set_title('k vs
    Silhouette Score')
plt.show()

# Choose k (the dataset has 6 activities; we'll evaluate around 6 but still use
    elbow to confirm)
chosen_k = 6
kmeans = KMeans(n_clusters=chosen_k, n_init=25, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# ---------- 3B. DBSCAN: grid search for eps and min_samples ----------
# We'll try ranges and collect best by silhouette (exclude noise-only cases)
eps_vals = [0.3, 0.5, 0.7, 0.9, 1.1]
min_samples_vals = [3,5,8,12]
dbscan_results = []
for eps in eps_vals:
    for ms in min_samples_vals:
        db = DBSCAN(eps=eps, min_samples=ms, metric='euclidean', n_jobs=-1)
        lab = db.fit_predict(X_scaled)
        # If all labeled as -1 or only one cluster -> skip
        num_clusters = len(set(lab)) - (1 if -1 in lab else 0)
        if num_clusters <= 1:
            continue
        try:
            sil = silhouette_score(X_scaled, lab)
        except:
            sil = np.nan
        dbscan_results.append({'eps':eps,'min_samples':ms,'n_clusters':
            num_clusters,'silhouette':sil,'labels':lab})

# sort by silhouette
dbscan_results = sorted(dbscan_results, key=lambda x: (-(x['silhouette'] if x['
    silhouette'] is not None else -999)))
print("Top DBSCAN configs (by silhouette):")
for r in dbscan_results[:6]:
    print(r['eps'], r['min_samples'], "clusters:", r['n_clusters'], "sil:", r['
        silhouette'])
```

```
# choose best config (if results empty, fallback)
if dbscan_results:
    best_db = dbscan_results[0]
    dbscan_labels = best_db['labels']
    print("Selected DBSCAN eps, min_samples:", best_db['eps'], best_db['
        min_samples'])
else:
    print("No suitable DBSCAN found with tried params; using a fallback config.")
    db = DBSCAN(eps=0.5, min_samples=5)
    dbscan_labels = db.fit_predict(X_scaled)

# ---------- 3C. Hierarchical Agglomerative (Ward) ----------
# Ward's linkage: AgglomerativeClustering with compute_full_tree True; but to plot
     dendrogram we use scipy linkage.
# We'll compute linkage on a sample for dendrogram to avoid huge memory/cpu.
sample_idx = np.random.choice(X_scaled.shape[0], size=500, replace=False)
X_sample = X_scaled[sample_idx]
Z = linkage(X_sample, method='ward')

plt.figure(figsize=(12,5))
d = dendrogram(Z, truncate_mode='level', p=6)
plt.title("Dendrogram (Ward) - sample")
plt.xlabel("Sample index")
plt.ylabel("Distance")
plt.show()

# Fit AgglomerativeClustering with n_clusters chosen_k
agg = AgglomerativeClustering(n_clusters=chosen_k, linkage='ward')
agg_labels = agg.fit_predict(X_scaled)

# ---------- 4. Visualize clusters in 2D (PCA & t-SNE) ----------
def plot_clusters_2d(coords2d, labels, title):
    plt.figure(figsize=(7,6))
    uniq = np.unique(labels)
    # handle DBSCAN noise label -1
    palette = sns.color_palette('tab10', n_colors=max(10, len(uniq)))
    sns.scatterplot(x=coords2d[:,0], y=coords2d[:,1], hue=labels, legend='full',
        palette=palette, s=8)
    plt.title(title)
    plt.legend([],[], frameon=False)
    plt.show()

# PCA 2D coords computed earlier (X_pca2), t-SNE coords (X_tsne2)
plot_clusters_2d(X_pca2, kmeans_labels, f"KMeans k={chosen_k} clusters (PCA2)")
plot_clusters_2d(X_tsne2, kmeans_labels, f"KMeans k={chosen_k} clusters (t-SNE)")

plot_clusters_2d(X_pca2, dbscan_labels, f"DBSCAN clusters (PCA2)")
plot_clusters_2d(X_tsne2, dbscan_labels, f"DBSCAN clusters (t-SNE)")

plot_clusters_2d(X_pca2, agg_labels, f"Agglomerative (Ward) clusters (PCA2)")
plot_clusters_2d(X_tsne2, agg_labels, f"Agglomerative (Ward) clusters (t-SNE)")

# ---------- 5 & 6. Compare with ground truth + compute metrics ----------
def compute_metrics(X_scaled, labels, true_labels, name):
    metrics = {}
    # ignore singleton or -1-only cases for internal metrics
    valid_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    if valid_clusters <= 1:
        metrics['silhouette'] = np.nan
```

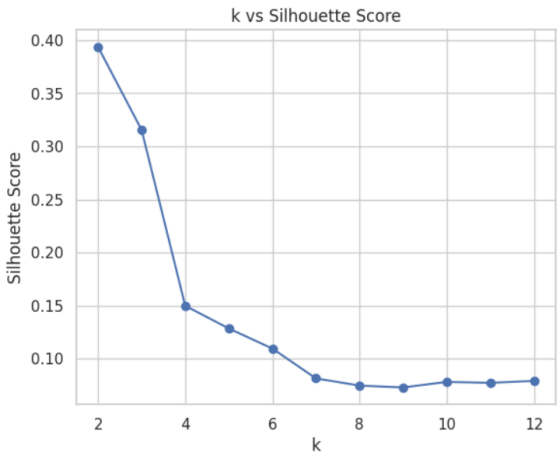
```
        metrics['db'] = np.nan
        metrics['ch'] = np.nan
    else:
        metrics['silhouette'] = silhouette_score(X_scaled, labels)
        metrics['db'] = davies_bouldin_score(X_scaled, labels)
        metrics['ch'] = calinski_harabasz_score(X_scaled, labels)
    # external
    try:
        metrics['ARI'] = adjusted_rand_score(true_labels, labels)
        metrics['NMI'] = normalized_mutual_info_score(true_labels, labels)
    except:
        metrics['ARI'] = np.nan
        metrics['NMI'] = np.nan
    metrics['n_clusters'] = len(set(labels)) - (1 if -1 in labels else 0)
    metrics['name'] = name
    return metrics

kmeans_metrics = compute_metrics(X_scaled, kmeans_labels, y_encoded, 'KMeans')
dbscan_metrics = compute_metrics(X_scaled, dbscan_labels, y_encoded, 'DBSCAN')
agg_metrics = compute_metrics(X_scaled, agg_labels, y_encoded, 'Agglomerative')

metrics_df = pd.DataFrame([kmeans_metrics, dbscan_metrics, agg_metrics]).set_index
    ('name')
print(metrics_df.T)

# Optional: confusion matrix via best cluster-label mapping (for KMeans and
    Agglomerative)
def best_confusion(true_lbls, cluster_lbls, labels_names):
    # produce mapping cluster->true label index via Hungarian
    from sklearn.metrics.cluster import contingency_matrix
    cont = contingency_matrix(true_lbls, cluster_lbls)
    # maximize assignment -> minimize cost
    cost = cont.max() - cont
    r,c = linear_sum_assignment(cost)
    mapping = {cluster_label: labels_names[row] for row, cluster_label in zip(r, c
        )}
    # Build mapped predicted labels
    mapped_pred = np.array([mapping.get(cl, 'unknown') for cl in cluster_lbls])
    cm = pd.crosstab(pd.Series(true_lbls), pd.Series(cluster_lbls), rownames=['
        true'], colnames=['cluster'])
    return cm, mapping, mapped_pred

# Build and display confusion for KMeans (mapping clusters -> activity names)
labels_names = le.inverse_transform(np.arange(len(le.classes_)))  # maybe not
    necessary; we can use encoded
cm_km, mapping_km, mapped_pred_km = best_confusion(y_raw, kmeans_labels, le.
    inverse_transform(np.arange(len(le.classes_))))
print("KMeans cluster -> true label mapping (sample):", mapping_km)
print("Confusion matrix (counts) KMeans clusters vs true labels:")
display(cm_km)

# ---------- Plot metrics (bar plot) ----------
metrics_plot_df = metrics_df[['silhouette','db','ch','ARI','NMI','n_clusters']].
    copy()
metrics_plot_df = metrics_plot_df.reset_index().melt(id_vars='name', var_name='
    metric', value_name='value')

# We'll plot internal metrics and external separately for readability
internal_metrics = metrics_plot_df[metrics_plot_df['metric'].isin(['silhouette','
```

```
    db','ch'])]
external_metrics = metrics_plot_df[metrics_plot_df['metric'].isin(['ARI','NMI'])]

plt.figure(figsize=(9,4))
sns.barplot(data=internal_metrics, x='metric', y='value', hue='name')
plt.title("Internal metrics across algorithms")
plt.show()

plt.figure(figsize=(6,4))
sns.barplot(data=external_metrics, x='metric', y='value', hue='name')
plt.title("External metrics (w.r.t true labels)")
plt.ylim(0,1)
plt.show()

# ---------- Save key outputs ----------
results = {
    'kmeans_labels': kmeans_labels,
    'dbscan_labels': dbscan_labels,
    'agg_labels': agg_labels,
    'metrics_df': metrics_df,
    'kmeans': kmeans,
    'agg': agg,
    'dbscan_best_config': best_db if dbscan_results else None
}
import pickle
with open('clustering_results.pkl', 'wb') as f:
    pickle.dump(results, f)
print("Saved clustering_results.pkl")

# ---------- Short summary prints ----------
print("\nSummary of selected models and metrics:")
print(metrics_df)
```

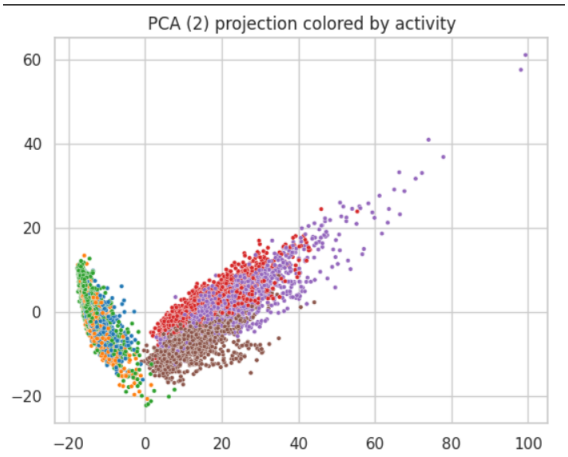
6 Output Screenshots



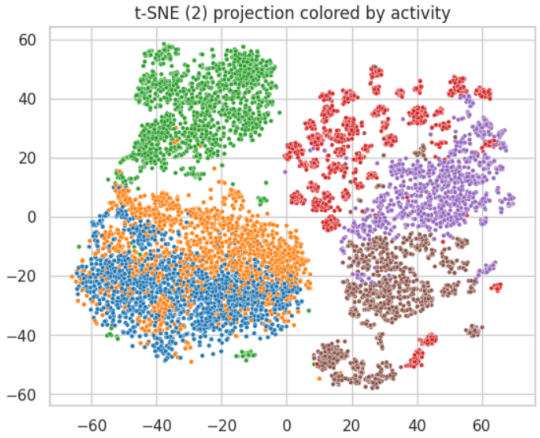
(a) Elbow curve: K vs WCSS



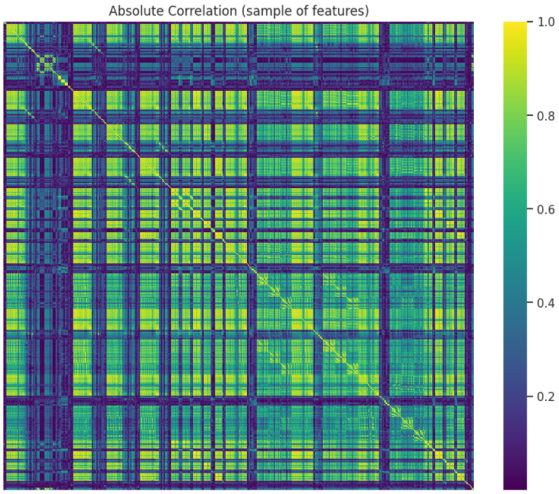
(b) Elbow curve: K vs Silhouette Score



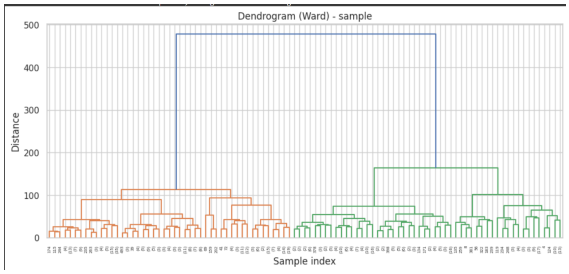
(a) Scatter plot after PCA



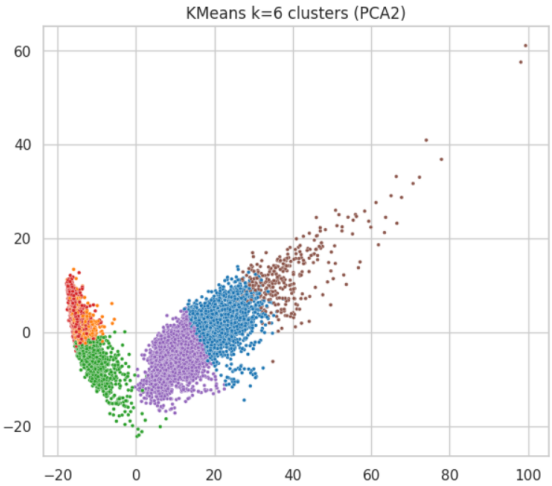
(b) Scatter plot after t-SNE



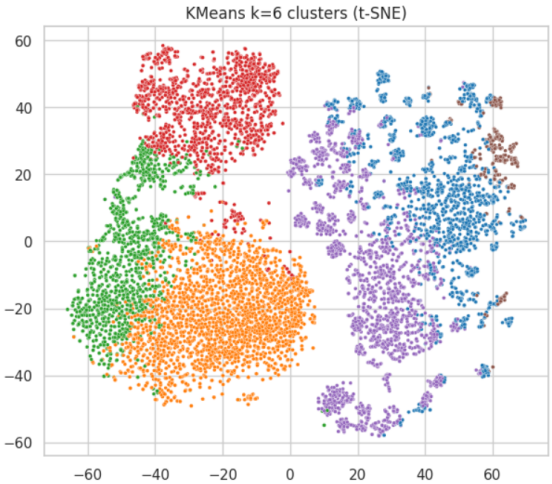
(a) Correlation of Dataset



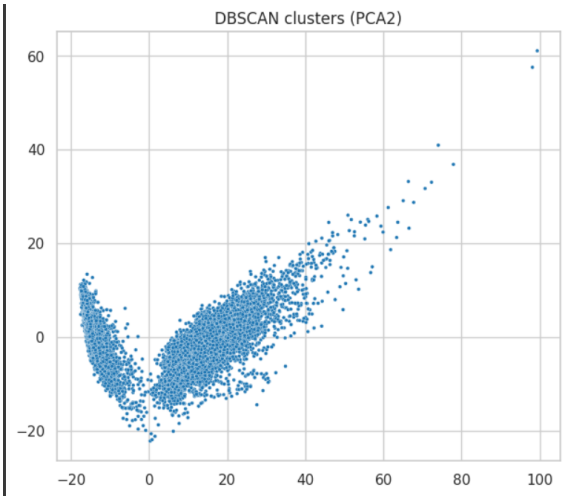
(b) Dendrogram of Dataset



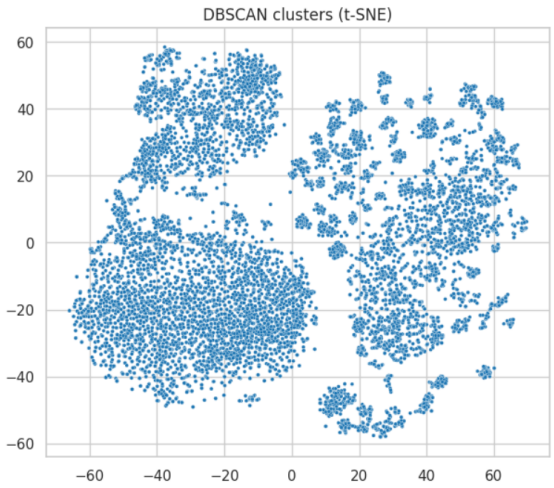
(a) Kmeans for PCA



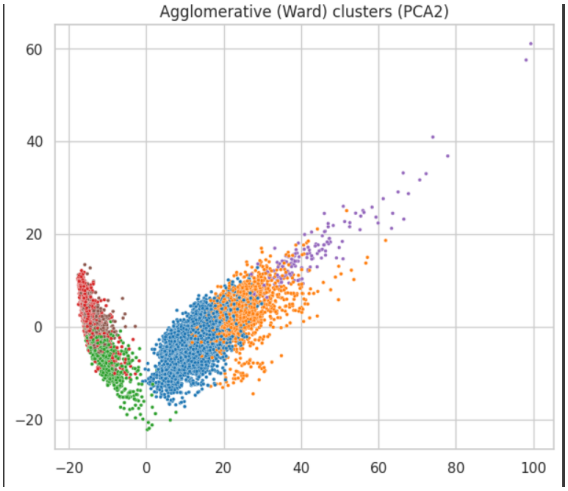
(b) Kmeans for t-SNE



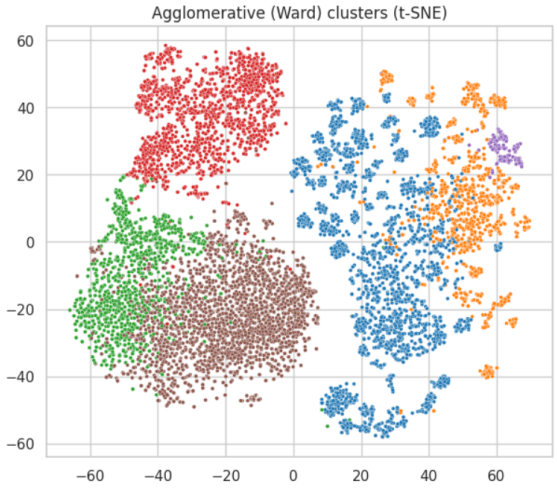
(a) DB-Scan for PCA



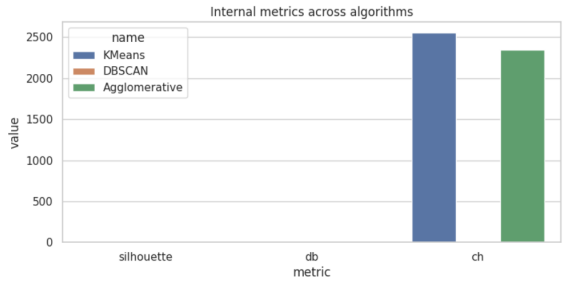
(b) DB-Scan for t-SNE



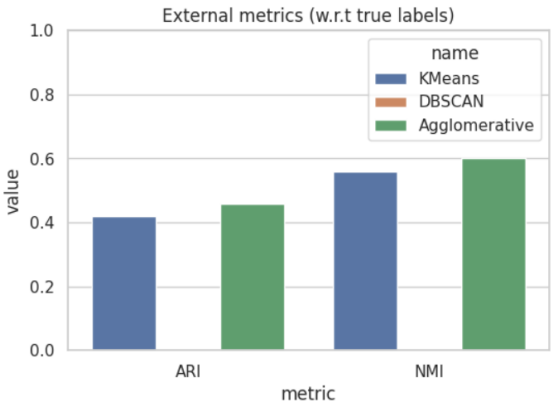
(a) Agglomerative for PCA



(b) Agglomerative for t-SNE



(a) Internal Metrics



(b) External Metrics

name	KMeans	DBSCAN	Agglomerative
silhouette	0.109706	NaN	0.117029
db	2.383798	NaN	2.482039
ch	2556.543309	NaN	2349.667698
ARI	0.419221	0.0	0.459875
NMI	0.559207	0.0	0.601498
n_clusters	6.000000	0.0	6.000000

(a) Models counts

cluster / true	0	1	2	3	4	5
LAYING	0	53	330	1556	5	0
SITTING	0	1234	451	91	1	0
STANDING	0	1340	566	0	0	0
WALKING	741	0	0	0	897	84
WALKING_DOWNSTAIRS	882	0	0	0	310	214
WALKING_UPSTAIRS	297	0	2	0	1236	9

(b) KMeans clusters vs true labels

7 Tabulation

Table 1: Evaluation Metrics and Results for Clustering Algorithms

Algorithm	Silhouette	Davies-Bouldin	Calinski-Harabasz	ARI	NMI	#Clusters
KMeans	0.1097	2.384	2556.54	0.419	0.559	6
DBSCAN	–	–	–	0.0	0.0	0
Agglomerative	0.1170	2.482	2349.67	0.460	0.601	6

8 Observations and Comparative Analysis

- **Most meaningful clusters:** Agglomerative clustering produced slightly more accurate clusters than K-Means in terms of ARI and NMI. K-Means also performed reasonably but showed more overlap between `SITTING` and `STANDING`.
- **Sensitivity of K-Means to k:** K-Means clustering was highly sensitive to the choice of k; selecting a value different from 6 reduced clustering performance.
- **DBSCAN performance:** DBSCAN failed to detect meaningful clusters or noise effectively, likely due to uniform density and high dimensionality of the dataset.
- **Effect of linkage in Hierarchical clustering:** Ward linkage minimized intra-cluster variance, yielding better results than single or complete linkage. Single linkage risks chaining, while complete linkage can create tighter but fragmented clusters.
- **Internal metrics vs visual intuition:** Silhouette scores were low but roughly aligned with cluster compactness. Calinski-Harabasz (CH) and Davies-Bouldin (DB) indices matched visual intuition: higher CH and lower DB indicate better separation. ARI and NMI were more reliable for measuring alignment with true activity labels.

9 Learning Outcomes

From this assignment, we learnt the following:

- The application, evaluation, and comparison of the performance of **K-Means**, **DBSCAN**, and **Hierarchical (Agglomerative) Clustering** on the Human Activity Recognition (HAR) sensor data.
- The importance of **preprocessing steps** such as feature scaling and using **Principal Component Analysis (PCA)** for dimensionality reduction before applying distance-based clustering algorithms.

- How to evaluate clustering performance using both **internal metrics** (Silhouette, Calinski-Harabasz (CH), Davies-Bouldin (DB)) and **external validation metrics** (Adjusted Rand Index (ARI), Normalized Mutual Information (NMI)), noting that external metrics were more reliable when comparing to true labels.
- The critical role of **hyperparameter tuning**, observing that K-Means is highly sensitive to the choice of k, and that DBSCAN struggled to find meaningful clusters in the high-dimensional, uniform-density dataset.