

# Report of mini project 3

Nazmus Sakib

nazmus.x.sakib@abo.fi

## Problem statement:

This project aims to apply K-Means and DBSCAN clustering algorithms to a given dataset. The tasks include determining the optimal number of clusters, finding optimal parameters, and understanding the impact of dimensionality reduction on clustering. I will have to compare the outcome of these models before and after dimensionality reduction was performed. Finally I will have to visualize the clusters. The dataset used in this project comprises sensor data collected from 30 volunteers, aged between 19 to 48 years, who wore a smartphone (Samsung Galaxy S II) on their waist while performing various activities. There are in total 361 columns in the dataset. Here is the google colab link:

[https://colab.research.google.com/drive/1-4eA\\_-fgWXemEDNuWKWSYkffs2PtzaeY?usp=sharing](https://colab.research.google.com/drive/1-4eA_-fgWXemEDNuWKWSYkffs2PtzaeY?usp=sharing)

## Data processing

### 1. Data Retrieval:

- Two datasets, X\_train.txt and X\_test.txt, were retrieved from the specified URLs. I uploaded them on s3 bucket earlier so that I don't have to upload them in colab.
- I didn't include the y\_train.txt and y\_test.txt because it is unsupervised learning.

### 2. Data Loading:

- The pandas library was utilized to load the datasets into dataframes, df\_train and df\_test, using the read\_csv function.
- Since the datasets were space-separated, the sep parameter was set to "\\s+", and no header was present, hence header=None.

### 3. Merging Datasets:

- Since we are performing unsupervised learning, there is no need for a test dataset.
- The training and test datasets were concatenated along the row axis using the concat function from pandas, resulting in a single dataframe, df\_merged.

## Modeling

### *Task 1.1 How do you choose the number of clusters in K-Means, is it the same number of clusters for DBSCAN?*

One common technique to determine the number of clusters is the elbow method. I tried the elbow method where I iteratively applied K-Means with different numbers of clusters and plotted the resulting Within-Cluster Sum of Squares (WCSS) against the number of clusters. By visually inspecting the plot, I looked for the point where the rate of decrease in WCSS significantly slowed down, forming an "elbow" shape. This point represented the optimal number of clusters. In my case I found it to be 2.

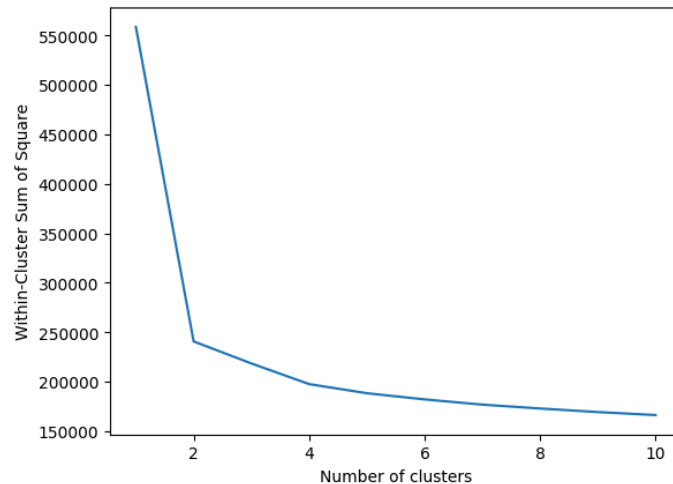


Fig: Elbow method to find the number of clusters for k-means

However, for DBSCAN, the optimal number of clusters was different. DBSCAN does not require pre-specification of the number of clusters. Instead, it clusters the data based on density connectivity. In my analysis, DBSCAN identified 3 clusters, with one cluster representing noise points. Therefore, while the elbow method led to the choice of 2 clusters for K-Means, DBSCAN determined 3 clusters, including noise points.

### ***Task 1.2 How do you find the optimal parameters' values?***

For K-Means, I simply employed the elbow method, which suggested using two clusters based on the analysis of Within-Cluster Sum of Squares (WCSS).

Meanwhile, for DBSCAN, I relied on the k-distance graph. By analyzing the k-distance graph, I visually inspected the plot to identify the epsilon  $\epsilon$  (epsilon) value. The graph depicted the distances to the k-th nearest neighbor for each data point, with the x-axis representing the data points sorted by distance and the y-axis indicating the distance to the neighbor. By scrutinizing the graph, I looked for a noticeable change in slope or curvature, signifying a transition from close proximity to greater separation among data points. Upon identifying this transition point, I determined it to be the epsilon value, serving as the distance threshold for defining neighborhoods in DBSCAN. There are two curvatures but I took the higher epsilon because it gave better results.

*Iterative Parameter Tuning:* I also looped through combinations of MinPts and  $\epsilon$  (epsilon) values, exploring variations both above and below the estimated thresholds. This iterative approach aimed to identify the best-fit model by fine-tuning parameter values. I found the best epsilon on slightly above the curvature and found the best min points value by trying out some different values.

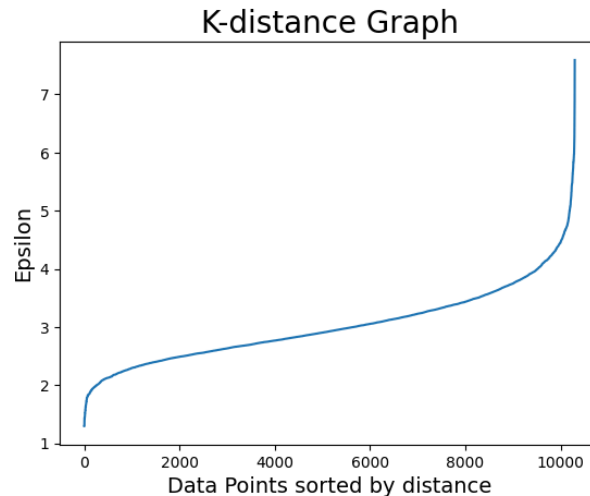


Fig: K-distance graph to find  $\epsilon$  for DBSCAN

**Task 1.3 What data processing steps do you apply and why?**

Data processing steps have already been explained in the Data Processing section. There were no additional processing needed to run k-means and DBSCAN

**Task 2.1 What is the dimensionality reduction technique that you choose, and why?**

I chose PCA (Principal Component Analysis) as the dimensionality reduction technique. PCA is a widely-used method known for its ability to effectively reduce the dimensionality of a dataset while retaining most of its important information. It accomplishes this by transforming the original features into a new set of orthogonal variables called principal components. PCA is preferred for its efficiency, interpretability, and ability to handle noise and collinearity among features. Overall, PCA offers a robust solution for preparing data for clustering analysis by reducing its dimensionality while preserving its meaningful structure.

I employed `PCA(n_components=0.8, svd_solver='full')` in my analysis. This configuration allowed me to utilize Principal Component Analysis (PCA) to reduce the dimensionality of the dataset while retaining at least 80% of its variance. This approach enabled me to effectively capture the essential information in the data while reducing its dimensionality, facilitating more efficient and interpretable clustering analysis.

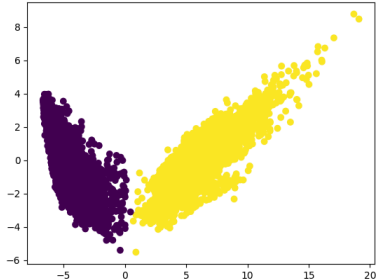
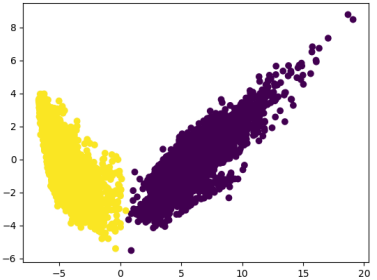
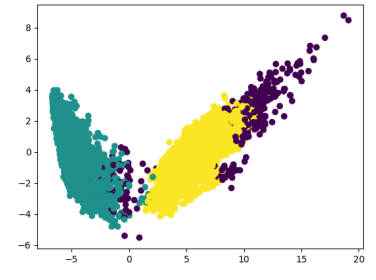
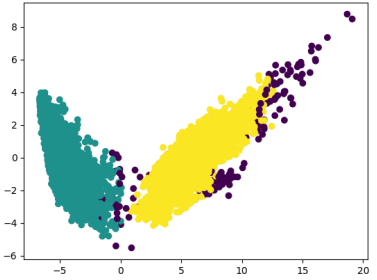
```
pca = PCA(n_components=0.8, svd_solver = 'full')
pca.fit(df_merged)
pca_array = pca.transform(df_merged)

pca_array.shape

(10299, 11)
```

**Dimensionality was reduced to 11 using PCA while retaining at least 80% of its variance**

**Task 2.2 Does it have any effect on your code efficiency, both in terms of computational efficiency and clustering output?**

Before dimensionality reduction	After dimensionality reduction
<pre>%%time kmeans = KMeans(n_clusters=2,init='k-means++') y_labels_knn = kmeans.fit_predict(df_merged) y_labels_knn print(Counter(y_labels_knn))</pre> <pre>/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py: warnings.warn( Counter({1: 5612, 0: 4687}) CPU times: user 1.93 s, sys: 552 ms, total: 2.48 s Wall time: 1.84 s</pre>	<pre>%%time kmeans = KMeans(n_clusters=2,init='k-means++') y_labels_pca_knn = kmeans.fit_predict(pca_array) y_labels_pca_knn print(Counter(y_labels_pca_knn))</pre> <pre>/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870 warnings.warn( Counter({1: 5612, 0: 4687}) CPU times: user 412 ms, sys: 278 ms, total: 690 ms Wall time: 361 ms</pre>
<pre>%%time dbscan = DBSCAN(eps=6, min_samples=1500) dbscan.fit(df_merged) y_labels_dbscan = dbscan.labels_ print(Counter(y_labels_dbscan))</pre> <pre>Counter({0: 5555, 1: 4377, -1: 367}) CPU times: user 8.93 s, sys: 579 ms, total: 9.51 s Wall time: 5.23 s</pre>	<pre>%%time dbscan = DBSCAN(eps=2.7, min_samples=50) dbscan.fit(pca_array) y_labels_pca_dbscan = dbscan.labels_ print(Counter(y_labels_pca_dbscan))</pre> <pre>Counter({0: 5592, 1: 4562, -1: 145}) CPU times: user 1.8 s, sys: 15.8 ms, total: 1.82 s Wall time: 1.85 s</pre>
 <p>K-means - 2 clusters</p>	 <p>K-means - 2 clusters</p>
 <p>DBSCAN - 3 clusters (1 for noise)</p>	 <p>DBSCAN - 3 clusters (1 for noise)</p>

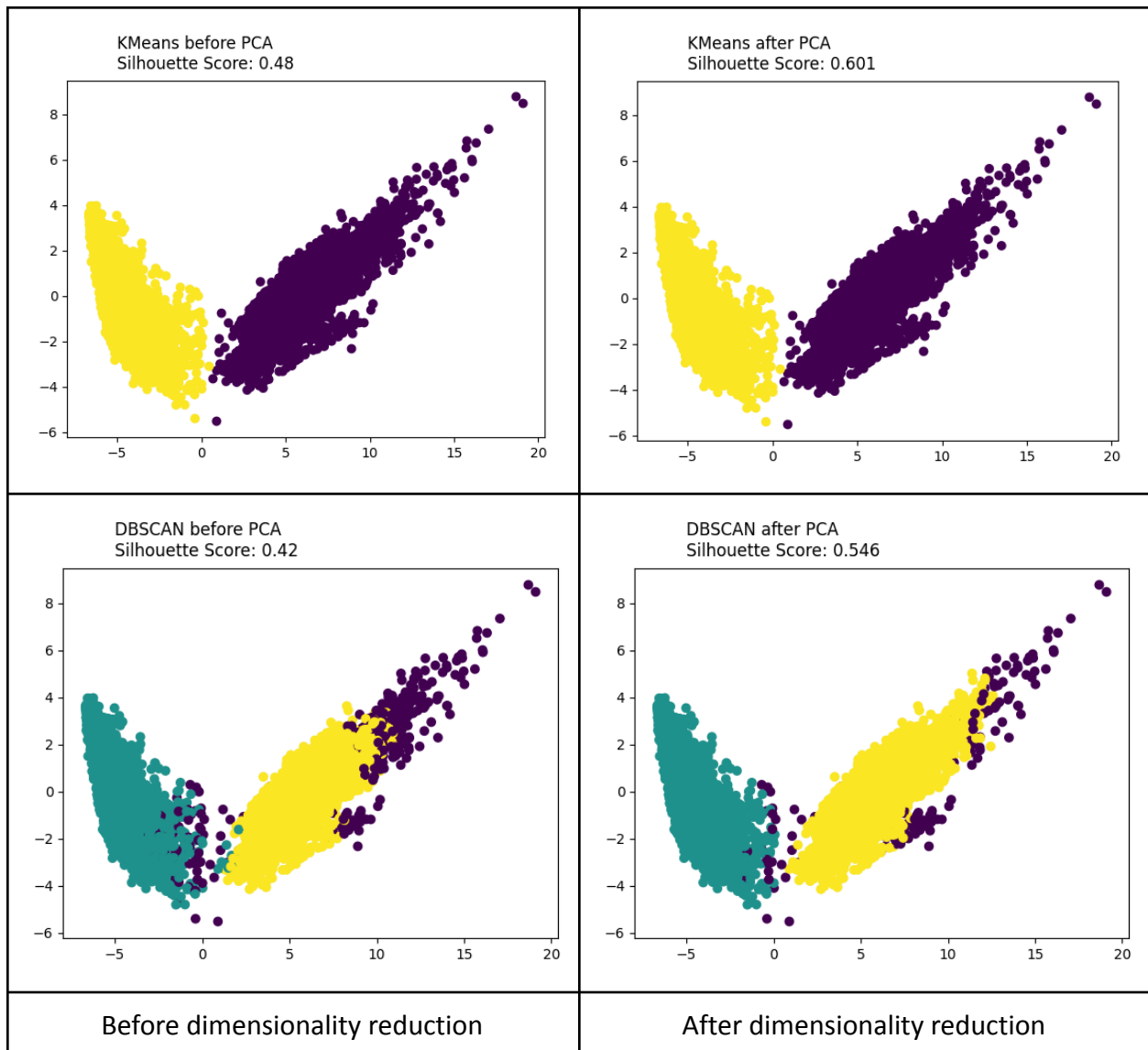
Here we can see k-means clustering took **1.84s** to complete before dimensionality reduction whereas it took **361ms** after dimensionality reduction.

For DBSCAN it took **5.23s** to complete before dimensionality reduction whereas it took **1.85s** after dimensionality reduction.

**So for both K-means and DBSCAN, dimensionality reduction increased computational efficiency**

But for clustering output, there was no effect on the number of clusters. Although I had to tune the hyperparameters again, to get the desired results. For DBSCAN, the numbers of points identified as noise (-1) decreased after applying PCA.

**Task 2.3 How do you compare the outcome of this model with the model where the dimensionality reduction technique was not applied to the dataset?**



After using dimensionality reduction, Silhouette Score was increased from **0.48** to **0.60** for K-means, whereas the same was increased from **0.42** to **0.54** for DBSCAN.

### ***Task 3.1 Have you applied any dimensionality reduction techniques? Why?***

Yes, I have applied dimensionality reduction techniques for visualization purposes. I applied Principal Component Analysis (**PCA**) to reduce the dimensionality of the dataset to **two** dimensions.

Dimensionality reduction techniques such as PCA (Principal Component Analysis) are commonly used to visualize high-dimensional data in lower-dimensional spaces, typically two or three dimensions.

The reason for applying dimensionality reduction techniques for visualization is the difficulty in visualizing data with more than two dimensions. Human perception is limited to three dimensions at best, making it challenging to effectively visualize high-dimensional data directly. By reducing the dimensionality of the data using techniques like PCA, we can project the data onto a lower-dimensional space, typically two or three dimensions. This transformation enables us to create visualizations that are easier to interpret and comprehend. In essence, dimensionality reduction for visualization simplifies the data representation, allowing us to explore and understand complex datasets more intuitively.

### **Conclusion:**

The scientific bottlenecks in this clustering task included determining the optimal parameters for the algorithms (K-Means and DBSCAN) and reducing the dimensionality of the dataset effectively. To overcome these challenges, I employed iterative exploration, visualization, and comparative analysis. I experimented with different parameter values and visualization techniques, such as the elbow method and k-distance graph, to identify patterns and make informed decisions. By evaluating multiple models and techniques, I successfully navigated the bottlenecks, leading to robust clustering results.

### ***How do you interpret the identified clusters? What do they represent?***

I got 2 clusters for KNN and 3 for DBSCAN. They represent two groups of activity. One cluster represents the moving activities (1 WALKING, 2 WALKING UPSTAIRS, 3 WALKING DOWNSTAIRS) and the other cluster represents the stationary states (4 SITTING, 5 STANDING, 6 LAYING).

In the violin plot we can see that activity class 1,2,3 belong to one cluster and activity class 4,5,6 belong to the other cluster in k-means. For DBSCAN clusters it's quite similar except some data points are labeled as noise (-1). In the scatter plot, we again see the similar trend.

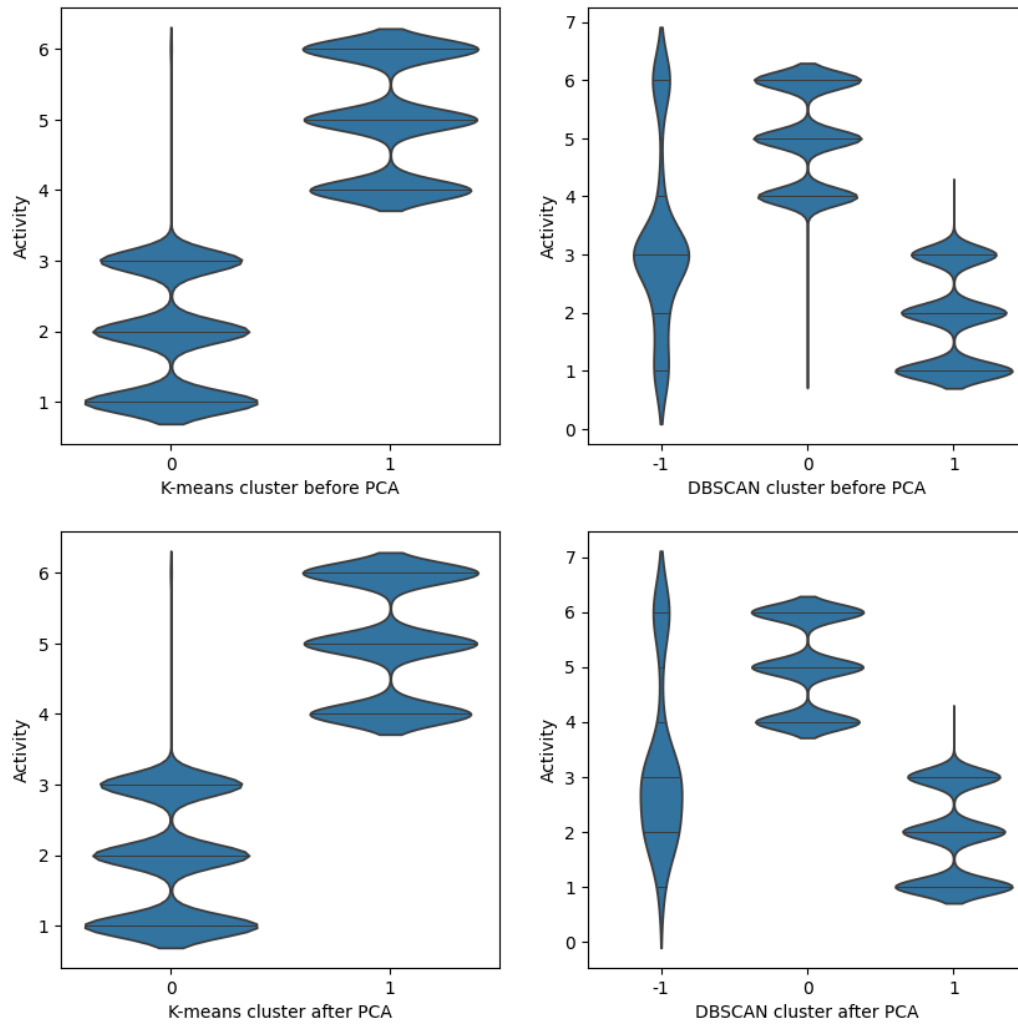


Figure: Violin plot showing how activity classes are distributed among the clusters

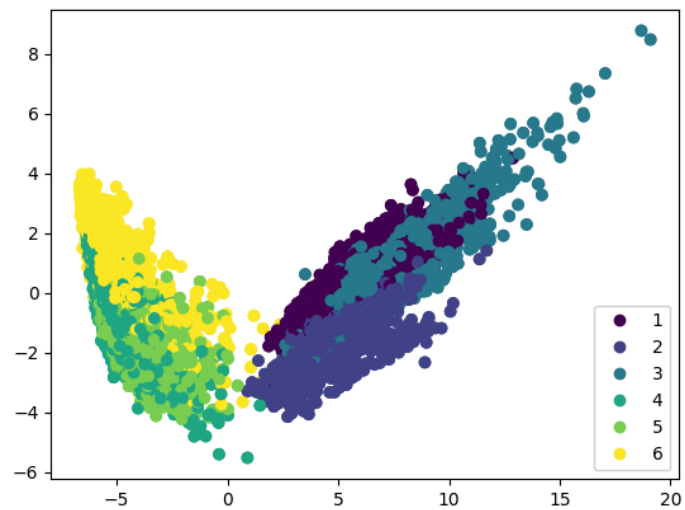


Figure: Scatter plot showing how activity classes are distributed among two clusters