HUMAN ACTIVITY RECOGNITION USING ML TECHNIQUES

EE 599 Course Project

DECEMBER 5, 2024

PROJECT TEAM: MOHAMMED ALSARAIHI & TANVI GANDHI
Contacts: alsaraih@usc.edu / tanvibhu@usc.edu

Contents

1. Abstract

This project investigates the application of Supervised Learning (SL), Semi-Supervised Learning (SSL), and Unsupervised Learning (USL) models to human activity recognition using sensor data, offering a detailed performance comparison across models. The dataset includes six predefined human activities—WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, and LAYING—collected using smartphone sensors. Extensive preprocessing and feature engineering were conducted to enhance model accuracy and address data complexity. For SL, models such as Support Vector Machines (SVM), CART, Random Forest, Adaboost, and Logistic Regression were employed, with logistic regression achieving the highest test accuracy of 92%. SSL leveraged partially labeled data through Semi-Supervised SVM and Expectation-Maximization (EM), with Semi-Supervised SVM achieving 93% accuracy. The USL approach utilized clustering algorithms like K-Medoids, Agglomerative Clustering, and Expectation-Maximization with Gaussian Mixture Models (GMM) to identify latent patterns. Techniques such as Principal Component Analysis (PCA) improved clustering efficiency, with performance evaluated using metrics like Silhouette Score, Calinski-Harabasz index, and Davies-Bouldin index. The optimal cluster count (k=2) consistently separated static (e.g., sitting, standing) from dynamic activities (e.g., walking), with K-Medoids achieving the highest Silhouette Score (0.61). This project demonstrates the strengths of diverse machine learning methodologies in addressing human activity recognition challenges.

2. Introduction

2.1. Problem Type, Statement and Goals

This project tackles a multi-class classification problem with the goal of accurately predicting class labels for unseen data. The work spans supervised learning (SL), semi-supervised learning (SSL), and unsupervised learning (USL) approaches to address diverse challenges in data classification and clustering.

Objectives:

1. *Supervised Learning (SL):*

- Train and evaluate classification models using labeled datasets for activity recognition tasks.
- Explore algorithms such as CART, Adaboost, Linear Regression, Random Forest and SVM to build robust baseline models.

2. *Semi-Supervised Learning (SSL):*

- Utilize partially labeled datasets to propagate labels to unlabeled instances.
- Apply methods like Expectation Maximization and S3VM (Semi-Supervised SVM) to leverage unlabeled data and enhance generalization.

3. *Unsupervised Learning (USL):*

- Remove activity labels and explore clustering to identify latent patterns in the dataset.
- Apply clustering algorithms like K-means and Gaussian Mixture Models (GMM) to uncover groupings without supervision.

4. *Feature Engineering, feature extraction & Dimensionality Reduction:*

- Extract meaningful features from raw signals to improve model performance across SL, SSL, and USL tasks.
- Apply techniques like Principal Component Analysis (PCA) and custom feature selection to handle high dimensionality.

5. *Cluster Evaluation and Comparison:*

- Evaluate clustering quality using metrics such as Silhouette Score and Adjusted Rand Index (ARI).
- Compare unsupervised clusters with true labels for insight into algorithm effectiveness.

Challenges:

1. *High Dimensionality:*
    - The original dataset comprised 561 features, which was even expanded to 81,936 in Trial 3 for USL due to advanced feature engineering for USL experiments, introducing computational bottlenecks.
2. *Nonlinear Relationships:*
    - Complex interactions between features created redundancy and increased computational complexity, especially in SSL and USL pipelines.
3. *Imbalanced Label Distribution:*
    - In SSL, the imbalance between labeled and unlabeled data presented challenges in label propagation and model optimization.

This project integrates SL, SSL, and USL techniques to comprehensively address the classification and clustering of human activity data, showcasing the interplay between diverse machine learning paradigms.

2.2. Our Prior and Related Work

None

2.3. Overview of Our Approach

2.3.1. Supervised Learning

Baselines:

1. Trivial Baseline (Majority Class Classifier): Predicts the most frequent class label to establish minimal accuracy.

2. Non-Trivial Baseline (Perceptron): A simple linear classifier that updates weights based on errors, serving as a fundamental benchmark.

Models:

1. **Logistic Regression**: Baseline performance was evaluated and improved using GridSearchCV to optimize hyperparameters such as regularization strength and penalty type. Logistic Regression optimizes the log-loss function and supports multi-class classification through extensions like one-vs-rest (OvR).

$$P(y = 1|x) = \frac{1}{1 + e^{-wx+b}}$$

2. **CART (Decision Tree):** Implemented with a max depth of 5, evaluated for train-test accuracy to analyze generalization.
3. **Random Forest**: Trained with 100 estimators and a max depth of 5. Learning curves highlighted generalization, with regularization applied to avoid overfitting.
4. **AdaBoost**: Built with 50 estimators and default decision trees. Further optimization included varying learning rates, tree depths, and algorithms to improve performance.
5. **SVM:** A linear kernel SVM trained on labeled data, evaluated for accuracy over multiple runs to ensure robust performance.

2.3.2. Semi Supervised Learning

1. **S3VM:** Combines labeled and unlabeled data to propagate labels, enhancing classification with limited labels. Performance compared against supervised SVM.
2. **EM:** Iteratively refines cluster assignments by combining labeled and pseudo-labeled data. Updates maximize learning from unlabeled data, addressing scenarios with sparse labels.

Evaluation Metrics Used for SL and SSL:

- **Supervised Learning (SL):** Accuracy, precision, recall, and F1-score across classes were used to evaluate the models, providing insights into both overall performance and per-class effectiveness.
- **Semi-Supervised Learning (SSL):** Accuracy was the primary metric, supplemented by cross-validation accuracy to assess generalization and robustness

    1. **Accuracy**: Measures the proportion of correctly classified samples over all samples:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

2. **Precision**: Proportion of true positives among all positive predictions, useful for imbalanced datasets:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

3. **Recall (Sensitivity):** Proportion of true positives among all actual positives:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

4. **F1-Score:** Harmonic mean of precision and recall

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. **Confusion Matrix**: Visualizes model performance by showing actual versus predicted class distributions.

2.4.3 Unsupervised Learning (USL)

For this section, a real-life scenario for unsupervised learning was implemented by removing all activity the labels from the dataset. The goal was to discover clusters and patterns in the data using various models and evaluate the performance using metrics established for USL.

The models used are:

- **Expectation Maximization - Gaussian Mixture Model (GMM):** Model clusters as a mixture of several Gaussian distributions with unknown parameters. It provides probabilistic soft assignments for data points.
- **K-Means**: K-Means is a special case of EM and known to be computationally efficient and faster. It assumes all clusters have the same prior probabilities and share a spherical covariance matrix.
- **K-Medoids- Partitioning Around Medoids (PAM):** A variant of K-Means where the cluster center is one of the data points (a medoid) rather than the mean. It's more robust to outlier since it minimizes pairwise dissimilarities but it's computationally slow because of the need to recalculate cluster labels and medoids iteratively.
- **Agglomerative Clustering:** Graphical-based hierarchical method which iteratively merges the clusters based on closeness based on identified linkage-criterion. This project used

Ward's linkage method, which minimizes the variance within merged clusters. Other linkage methods covered in class, such as nearest linkage, farthest linkage, and average linkage, could also be applied.

The baseline:

The baseline system assumes 6 clusters, leveraging prior knowledge on human activities and readings based on the triaxial acceleration and angular velocity data captured by the sensors which capture distinct activities.

The baseline assumption reflects a reasonable reference to evaluate the clustering using domain knowledge of the activities. The clustering was performed without using activity labels to simulate real-life unsupervised scenarios. Below are the results for the 3 major metrics used for all algorithms.

Clustering Method (K= 6)	Silhouette Score	Calinski-Harabasz Index	Davies-Bouldin Index
K-Means (Baseline	0.110	2556.542	2.384
EM-GMM	0.135	2458.728	2.581
Agglomerative	0.117	2349.668	2.482
K-Medoids	0.062	2398.673	2.750

We have optimized the evaluation metrics across various feature engineering trials for cluster counts ranging from 2 to 15. The metrics selected were comprehensive and cover most of the well-known best practices among researchers in the field.

Evaluation Metrics Used for USL:

1. **Silhouette Score:** It's a popular recognized method among researchers and engineers to evaluate the quality of clusters in USL. It offers quantitative measures by assessing how similar each point is to its assigned cluster (Cohesion) compared to other clusters (Separation).

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

- a(i) is the mean distance within clusters (intra-cluster)
- b(i) is the mean distance between clusters (nearest cluster)

Interpretation of the Silhouette score:

The score evaluation can range from -1 to 1:

- +1: The point is matching with its assigned clusters and well separated from other clusters
- 0: The point lies on the boundary between clusters
- -1: The point is assigned to wrong clusters (not desired)

In summary, a well-designed model should have a silhouette score as close as possible to 1 to indicate well defined clusters. However, due to real-world complexity a score around 0.5 or above is generally acceptable and a score below 0.25 suggests weak clustering with overlap among clusters. [2]

2. **Calinski-Harabasz Index (Variance Ratio Criterion):** It's another similar method that evaluates the clustering quality. It measures the sample variance between cluster means and compares it to the sample variance within clusters.

$$CH(k) = \frac{\frac{1}{k-1}\sum_{k=1}^{K} N_k \|m_k - m\|^2}{\frac{1}{N-k}\sum_{k=1}^{K}\sum_{x_i \in C_k} \|x_i - m_k\|^2}$$

- N_k: Total number of points in cluster C_k
- m_k: mean of points in cluster C_k
- m: means of the dataset
- k: number of clusters

A higher score indicates better clustering with low variance within clusters and high separation between clusters. The optimal number of clusters (k) is the cluster that maximizes the score. [2]

3. **Davies-Bouldin Index:** It's another quantitative measure that focuses on the compactness and how tight the points within clusters compare with the distance between cluster's centroids.

$$DBI = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i} \left(\frac{s_i + s_j}{d_{ij}} \right)$$

- K: Number of clusters
- S_i: Average intra-cluster distance for cluster i (compactness of cluster i)
- d_{ij}: Distance between clusters i and j centroids (Separation)

There's no defined range for DBI score but generally the lower the better clustering because it gives lower intra-cluster distance and higher inter-cluster distances. A score close to 0 demonstrates excellent clustering while below 1.5 is good enough in real-world scenarios.

4. **Stepwise Criterion- Agglomerative Clustering only** : It's a method used in Agglomerative Clustering to evaluate and select the optimal number of clusters during the merging. It assesses the change in the minimum cluster distance at each iteration and identifies the significant jump to determine the optimal k. [2]

3. Implementation

3.1. Data Set

The Human Activity Recognition (HAR) dataset was constructed using recordings from 30 participants performing daily activities (ADLs) while carrying a waist-mounted smartphone with embedded inertial sensors. The goal is to classify these activities into one of six predefined categories.

Experiment Details:

- Participants: 30 volunteers aged between 19–48 years.
- Activities: Each participant performed six activities: WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, and LAYING.
- Device: A smartphone (Samsung Galaxy S II) worn at the waist recorded:
  - Triaxial acceleration (total and body acceleration) from the accelerometer.
  - Triaxial angular velocity from the gyroscope.
- Sampling Rate: Data was collected at a constant rate of 50Hz.
- Data Labeling: Activities were manually labeled using video recordings.

Data Partitioning:

The dataset was randomly divided into two sets from the source, as follows:

- Training Data: 70% of total data
- Test Data: 30% of total data

We have furtherly separated using the 70-15-15 splitting the test data into 15% validation and 15% test set.

Features:

- A 561-feature vector containing time and frequency domain variables (numerical).
- Subject Identifiers: To link data to individual participants. (categorical)

Multi-Classes:

- Activity Labels: Corresponding to the performed activity which are WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, and LAYING

This dataset serves as a comprehensive benchmark for evaluating activity recognition algorithms.

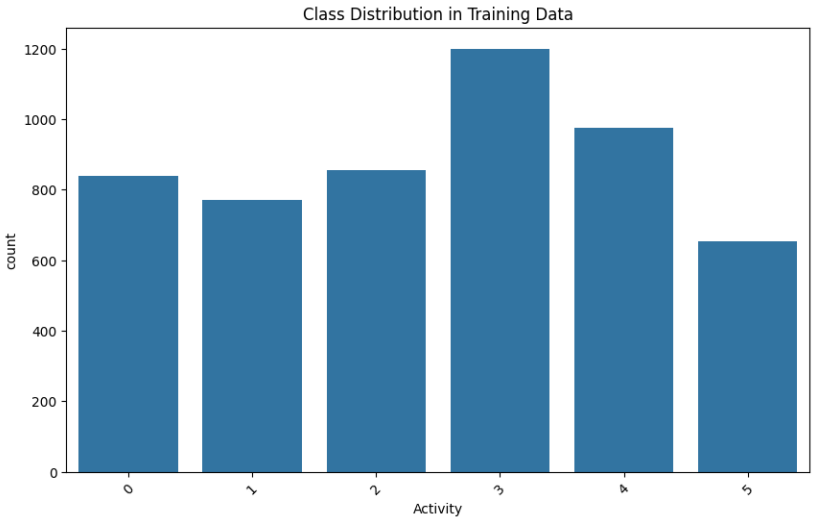
3.2. Dataset Methodology

The dataset used in this project consisted of approximately 10,299 data points, split into 70% training, 20% validation, and 10% testing. The training set included both labeled (70%) and unlabeled (30%) data, enabling a semi-supervised learning approach where the initial model was trained on labeled data, and pseudo-labels were generated for the unlabeled portion. These pseudo-labeled examples were combined with the labeled data to retrain the model for improved performance. A 3-fold cross-validation approach was used during training to tune hyperparameters and evaluate model consistency, with nested cross-validation employed within RandomizedSearchCV for parameter optimization. The validation set guided decisions on hyperparameter tuning and model adjustments, while the test set, accessed only once after training, provided an unbiased evaluation of the final model's performance, ensuring robust generalization.

3.3. Preprocessing, Feature Extraction, Dimensionality Adjustment

Preprocessing

- **Initial Data Assessment:**
  - Checked for missing values; none were present.
  - Verified class distribution in the training set, which appeared balanced.



- **Simulating Real-World Challenges:**
  To simulate real-world scenarios and enhance robustness, the following modifications were introduced to the dataset:
  - Missing Values:
    - Added 30% missing values randomly across the dataset.
    - Filled these missing values with the class-wise mean of each feature.

  - Outliers:
    - Introduced outliers in 5% of the data points, detected using Z-score analysis.
    - Removed these outliers to maintain data quality, which was feasible due to the high ratio of data points to features.
  - Inconsistent Scales:
    - Selected three features at random and scaled them differently.
    - Applied normalization to standardize scales across all features.
  - Class Imbalance:
    - Introduced class imbalance by reducing the representation of four specific classes: LAYING, SITTING, STANDING, WALKING_UPSTAIRS, to 60% of their original size.
    - This was achieved by randomly dropping samples.
    - SMOTE (Synthetic Minority Oversampling Technique) was applied to generate synthetic samples for the minority classes, restoring a balanced class distribution.
  - Label Noise:
    - Randomly altered 10% of the labels in the training dataset to simulate label noise.
    - Addressed this challenge using:
      - Label smoothing techniques.
      - Algorithms robust to label noise, such as Random Forest.

- **Encoding:**

Transformed the Activity field's categorical labels into numeric form using LabelEncoder.

Feature Engineering and feature extraction

- New Features Added:
  - Statistical Features:
    - Mean Magnitude: Captures the overall intensity of motion.
    - Max-Min Values: Highlights extreme variations in key features.
  - Domain-Specific Features:
    - Derived jerk magnitudes from sensor readings to capture sudden movements.
  - Feature Interactions:
    - Calculated interactions by multiplying accelerometer and gyroscope means across axes to better represent combined motion patterns.
- Feature Selection:
  - Performed correlation analysis to identify and remove redundant features with high correlation values, using a predefined correlation coefficient threshold.
  - Applied Principal Component Analysis (PCA):
    - Reduced dimensionality by transforming the data into orthogonal components.

- Addressed the high feature correlation caused by sensor data's inherent characteristics, particularly its representation of body movement.

3.4. Training Process

In this section, the training processes, parameter tuning, and final chosen configurations for each model are explained, highlighting their justification and impact on performance.

Trivial Model

- Description:
  The Most Frequent model was used as a trivial baseline. This classifier always predicts the majority class from the training data, serving as a benchmark to compare against more complex models.
- Justification:
  This model provides a baseline accuracy achievable without any meaningful learning process. Comparing advanced models to this benchmark helps evaluate their effectiveness.
- Parameters:
  - Strategy: "most_frequent" to ensure predictions are based solely on the most frequent class in the training set.

Non-Trivial Model: Perceptron

- Description:
  A Perceptron model was implemented as a linear classifier, using the perceptron learning rule to iteratively adjust weights for classification tasks.
- Justification:
  The perceptron provides a step up from the trivial model, introducing a learning process to improve prediction accuracy. As a simple linear model, it is suitable for understanding dataset separability and serves as a steppingstone to more sophisticated models.
- Parameters:
  - max_iter=1000: Limits the number of iterations for convergence.
  - random_state=42: Ensures reproducibility.

Baseline Logistic Regression

- Description:
  A Logistic Regression model was trained using the lbfgs solver, effective for small-to-medium-sized datasets. Features were standardized using StandardScaler to improve convergence and accuracy.
- Justification:
  Logistic Regression was chosen for its solid linear baseline, probabilistic outputs, and

interpretability. Standardization ensures equal contribution of features to the learning process.

- Parameters:
  - max_iter=1000: Ensures sufficient iterations for convergence.
  - solver='lbfgs': Suitable for multi-class classification.
  - random_state=42: Ensures reproducibility.

Optimized Logistic Regression

- Description:
  Hyperparameter tuning was conducted using GridSearchCV with 3-fold cross-validation. Regularization strength (C), penalty type (l1 or l2), and solver (liblinear) were optimized.
- Justification:
  Grid search ensures the best parameter combination is selected, enhancing the model's generalization capability.
- Parameters:
  - C: Regularization strength (values tested: [0.01, 0.1, 1, 10, 20, 30]).
  - penalty: Regularization type (l1 for sparse models, l2 for ridge-like regularization).
  - solver='liblinear'`: Compatible with l1 regularization.

Classification and Regression Tree (CART)

Baseline Model

- Performance Metrics:
  - Train Error: 0.21 (79% Train Accuracy)
  - Test Error: 0.22 (78% Test Accuracy)
  - Cross-Validation Accuracy: 0.70
- Interpretation:
  The model underfits the data, evident from its relatively high training error and limited test performance. Cross-validation confirms its modest generalization capability.

Tuned Model

- Performance Metrics:
  - Train Error: 0.07 (93% Train Accuracy)
  - Test Error: 0.24 (76% Test Accuracy)
- Interpretation:
  Increasing depth reduces bias, leading to lower training error. However, a rise in test error indicates overfitting, as the model captures noise specific to the training set.

CART

Model Description:
Implemented using the sklearn.tree module. The Decision Tree Classifier recursively splits data at decision nodes to minimize class impurity, using the Gini Impurity criterion. This approach is effective for high-dimensional data with 561 features and 6 target classes.

Chosen Parameters:

- Criterion: 'gini' for computational efficiency and effective class separation.
- Maximum Depth:
    - Baseline: max_depth=5 for simplicity and interpretability.
    - Tuned: max_depth=10 to explore greater model complexity.
- Random State: 42 to ensure reproducibility.

Justification:

- Baseline Model: Selected for its simplicity, quick benchmarking, and reduced risk of overfitting.
- Tuned Model: Increased depth to mitigate underfitting and capture more information from the dataset.

Training Process:

- Baseline Training: Trained a shallow tree (max_depth=5) as an initial benchmark, balancing bias and variance.
- Tuned Model Training: Increased depth to max_depth=10 to identify more patterns in the data. This reduced training error but increased overfitting, emphasizing the need for careful hyperparameter tuning.

Random Forest

Model Description:
Implemented using the RandomForestClassifier from Scikit-learn v1.3.0. This ensemble method combines predictions from multiple decision trees to improve accuracy and reduce overfitting.

Chosen Parameters:

- Initial Configuration: n_estimators=100, max_depth=5, min_samples_split=2.
- Tuned Configuration: n_estimators=50, max_depth=5, min_samples_split=4.

Justification:
Random Forest leverages ensemble learning to handle high-dimensional data effectively,

reducing the risk of overfitting by aggregating diverse tree predictions. Shallow depth was chosen to balance model complexity and interpretability.

Training Process:

1. Data Splitting: The Random Forest was trained on the X_train dataset and evaluated on X_test.
2. Cross-Validation: Hyperparameters were tuned using GridSearchCV with 5-fold cross-validation.
3. Tree Training: Each tree was trained on bootstrapped subsets, with features for splitting selected randomly to enhance model robustness.

AdaBoost

Model Description:
Implemented using the AdaBoostClassifier from Scikit-learn v1.3.0, with a DecisionTreeClassifier as the base estimator. AdaBoost focuses sequentially on harder-to-classify instances, improving overall model accuracy.

Chosen Parameters:

- Default Configuration: n_estimators=50, learning_rate=1.0.
- Tuned Configuration: n_estimators=200, learning_rate=0.1, max_depth=3.

Justification:
AdaBoost is particularly effective for datasets with mixed complexity, where distinguishing between static and dynamic activities is challenging. Sequentially correcting misclassified instances improves model performance.

Training Process:

1. Data Splitting: The model was trained on an 80/20 split of the data (X_train, X_test).
2. Iterative Training: Weak learners (shallow decision trees) were trained iteratively, correcting errors from prior trees.
3. Loss Minimization: Followed AdaBoost's exponential loss function to guide the learning process.
4. Cross-Validation: Hyperparameters were optimized using GridSearchCV.

SVM

Model Description:
Implemented using the SVC from Scikit-learn's svm module. SVM models were adapted for semi-supervised learning to incorporate both labeled and unlabeled data.

Chosen Parameters:

- Initial Configuration: C=1, kernel='linear', probability=True.
- Tuned Parameters: Explored RBF and polynomial kernels through hyperparameter tuning.

Justification:
SVM was selected for its ability to handle linear separability with a simple training process. Semi-supervised learning using pseudo-labeling extended its utility by leveraging unlabeled data.

Training Process:

- Data Preparation:
    - Split 70% of the data as labeled (labeled_X, labeled_y) and 30% as unlabeled (unlabeled_X).
- Ensured reproducibility with a fixed random seed.
- Initial Training: Trained a linear SVM on labeled data (C=1).
- Pseudo-Labeling:
    - Predicted labels for unlabeled data using the initial model.
    - Combined labeled and pseudo-labeled data to form a new training set.
- Retraining: Retrained the SVM on the combined dataset.
- Testing: Evaluated the final model on X_test using accuracy and classification metrics.

S3VM Model

Model Description:
The Semi-Supervised Support Vector Machine (S3VM) leverages both labeled and unlabeled data, making it suitable for scenarios with limited labeled data. Using an iterative pseudo-labeling process, it assigns labels to unlabeled data and retrains the model to improve generalization. A linear SVM was implemented using Scikit-learn's sklearn.svm.SVC module for simplicity and computational efficiency.

Training Process:

1. Data Splitting:

- 70% of the training data was treated as labeled.
- 30% was treated as unlabeled.

2. First Iteration:
   - Trained an initial SVM on the labeled data.
   - Generated pseudo-labels for the unlabeled data.
3. Retraining:
   - Combined labeled and pseudo-labeled data.
   - Retrained the SVM on the combined dataset.
4. Evaluation:
   - Tested the final model on the test set (X_test, y_test) and computed accuracy and classification metrics.

Parameter Tuning:

- C: Default value of 1.0 to balance margin maximization and classification error.
- Kernel: Linear, for an interpretable baseline.

EM Model

Model Description:
The Expectation-Maximization (EM) algorithm was used to simulate a semi-supervised learning scenario by iteratively refining pseudo-labels for unlabeled data with a Gaussian Mixture Model (GMM). The GMM was initialized with 6 components (matching activity classes) and means estimated from labeled data.

Training Process:

1. Initialization:
   - 70% of training labels were masked as unlabeled.
   - GMM initialized with 6 components.
2. EM Iterations:
   - E-Step: Predicted class probabilities using GMM and assigned pseudo-labels based on the highest probability.
   - M-Step: Refitted the GMM using both true and pseudo-labeled data.
3. Iteration:
   - Repeated for 20 iterations, monitoring labeled subset accuracy.

Label Propagation

Model Description:
The Label Propagation model is a graph-based semi-supervised learning approach that uses labeled and unlabeled data. It constructs a similarity graph and propagates labels iteratively to unlabeled data points based on graph connectivity. An RBF kernel with a gamma value of 20 was employed to define similarity between points.

Justification:
Label Propagation leverages both labeled and unlabeled data, making it particularly suitable for datasets where labeled samples are limited. It effectively utilizes the underlying cluster structures in the data to improve label prediction for unlabeled instances.

Training Process:

- kernel='rbf': Ensures the similarity graph can capture non-linear relationships.
- gamma=20: Controls the influence of the RBF kernel on the similarity graph.
- max_iter=1000: Ensures sufficient iterations for convergence.

3.5. Model Selection and Comparison of Results

Trivial Model

The trivial model, with a training accuracy of 16.83% and a presumed equivalent test accuracy, serves as a baseline by predicting only the majority class. This approach highlights the dataset's class imbalance and provides a minimal benchmark for more sophisticated models. However, the trivial model lacks any generalization capability or interpretability, making it unsuitable for practical use.

Perceptron

The perceptron achieves a high training accuracy of 88.36%, but its test accuracy plummets to 32.51%, indicating severe overfitting. With a macro-average precision of 89%, recall of 87%, and F1-score of 88%, the model performs well during training but fails to generalize. Confusion matrices reveal its tendency to predict a single class, leading to imbalanced precision and recall for minority classes. While it surpasses the trivial model, the perceptron's limited generalization and inability to handle class imbalance necessitate improvements in robustness, such as incorporating regularization or exploring non-linear decision boundaries.

Baseline Logistic Regression

The baseline logistic regression significantly outperforms the perceptron, with a training accuracy of 93.22% and a test accuracy of 82.46%. It strikes a better balance between bias and variance, as reflected in higher recall for certain classes, though slight overfitting is evident. The classification report highlights consistent improvements across precision, recall, and F1-scores compared to earlier models, establishing logistic regression as a reliable linear baseline.

Optimized Logistic Regression

Optimization through L1 regularization (C=0.1, penalty='l1') enhances logistic regression's performance, achieving a cross-validation accuracy of 85.02% and a test accuracy of 89.99%. Precision, recall, and F1-scores are balanced across classes, with macro and weighted averages nearing 90%. The improvements reflect better handling of sparsity and class variability, reducing overfitting and enhancing generalization by 7.53% compared to the baseline logistic regression.

CART (Classification and Regression Trees)

The baseline CART model with a maximum depth of 5 achieves a training accuracy of 79% and a test accuracy of 78%, underfitting the data due to limited depth. Increasing the depth to 10 improves training accuracy to 93%, but test accuracy drops to 76%, indicating overfitting as the model captures noise. While CART provides interpretable decision boundaries, its sensitivity to hyperparameters necessitates tuning for optimal performance.

Random Forest

The random forest baseline configuration achieves a training accuracy of 81% and a test accuracy of 79%, with moderate overfitting due to limited depth. Hyperparameter tuning (n_estimators=50, max_depth=5, min_samples_split=4) improves performance slightly, with training and test accuracies of 82% and 80%, respectively. This suggests that random forest offers robust performance with reduced overfitting compared to individual decision trees, benefiting from ensemble averaging.

AdaBoost

The default AdaBoost model achieves a training accuracy of 74% and a test accuracy of 76%, exhibiting a stable learning curve but significant training error (~26%). Tuning parameters (n_estimators=200, learning_rate=0.1, max_depth=3) improves test accuracy to 83%, reducing overfitting and enhancing precision and recall across classes. AdaBoost demonstrates the importance of combining weak learners for better generalization and robustness.

Support Vector Machine (SVM)

With the RBF kernel, the baseline SVM achieves a test accuracy of 73.97%. Hyperparameter tuning (C=10, kernel='rbf', gamma='scale') boosts cross-validation accuracy to 86.21% and test accuracy to 86%. The optimized SVM performs significantly better than its linear counterpart, balancing precision, recall, and F1-scores across all classes. The non-linear kernel effectively captures complex relationships in the data, making SVM a strong contender.

Semi-Supervised Support Vector Machine (S3VM)

The linear S3VM leverages both labeled and pseudo-labeled data, achieving a training accuracy of 94% and a test accuracy of 93.25%. Optimized hyperparameters via RandomizedSearchCV lead to a slight reduction in test accuracy to 92.64%, indicating convergence. The RBF S3VM and PCA-augmented S3VM underperform significantly, with test accuracies of 47.7% and 43.3%, respectively, reflecting their incompatibility with the dataset. S3VM showcases the potential of semi-supervised learning when data scarcity is a concern, with linear S3VM emerging as a top-performing model.

EM

The Expectation-Maximization (EM) Model, trained on 70% unlabeled data, struggled to perform effectively, achieving a training accuracy of 18.5% and a test accuracy of 16%. The heavy reliance on pseudo-labels, combined with the dataset's high dimensionality and class imbalance, made it difficult for EM to converge meaningfully. This resulted in poor precision,

recall, and F1-scores, as the model primarily favored the majority class. While EM is theoretically advantageous for semi-supervised learning, its reliance on a large proportion of unlabeled data limited its effectiveness in this scenario compared to more robust models like Random Forest, Adaboost, SVM, S3VM.

Label propagation

The Label Propagation model achieved a perfect training accuracy of 1.0 due to its iterative optimization of the similarity graph, which ensures all labeled points are classified correctly. On the test dataset, it achieved an accuracy of 74.96%, with macro-average precision, recall, and F1-scores of 0.75, 0.74, and 0.74, respectively. While the model performed well on majority classes like activity 0 (F1-score 0.90), it struggled with minority classes such as activity 1 (F1-score 0.66) due to class imbalance. Its reliance on similarity propagation makes it effective in structured data but limits its ability to handle complex boundaries and imbalanced distributions.

Comparison Among Models

The **Optimized Logistic Regression** and **Linear S3VM** are the standout performers, offering balanced precision, recall, and F1-scores, as well as high generalization. While SVMs excel with non-linear relationships, their performance does not surpass the optimized linear models for this dataset. Ensemble methods like **Random Forest** and **AdaBoost** demonstrate robust generalization but fall short of the best-performing models. The **Perceptron**, **Trivial Model**, and **EM** lag significantly, emphasizing the need for optimization and advanced techniques to handle high-dimensional, imbalanced datasets.

Model	Training Accuracy	Test Accuracy	Notes
Trivial (Majority)	16.83%	15%	Baseline; predicts majority class
Perceptron	60.37%	56.16%	Linear classifier with moderate improvement
Logistic Regression	93.22%	82.46%	Good Linear Regression Baseline
Optimized Logistic Regression	94%	89.99%	Improved Accuracy
CART depth 5	79%	80%	Simple model is more efficient
CART depth 10	93%	76%	Simple model is more efficient
Random Forest	81%	78.9%	Good baseline Accuracy
Tuned Random Forest	82%	80%	Hyperparameter optimization process effectively enhanced the model's generalization ability

Adaboost	74%	76%	Initial AdaBoost model shows moderate overfitting but achieves reasonable generalization.
Tuned Adaboost	85%	83%	Tuning improved both training and test accuracy significantly, reducing overfitting and enhancing model performance.
Baseline SVM	72%	73.97%	The non-linear RBF kernel likely captured complex relationships in the feature space.
Optimized SVM	89%	86.00%	The optimized model with RBF kernel performed significantly better than the linear baseline.
Linear S3VM	94%	93.25%	Strong baseline performance using labeled + pseudo-labeled data
Optimized S3VM (Randomized Search)	93.8%	92.64%	Similar accuracy, suggesting model convergence with tuned hyperparameters
RBF S3VM	50.80%	47.7%	Poor performance, indicating non-linearity did not align with this dataset
PCA + Linear S3VM	45.10%	43.3%	PCA for dimensionality reduction significantly reduced performance
EM	18.50%	16%	Expected accuracy for 70% unlabeled data
Label Propagation	100%	74.96%	Strong performance leveraging cluster structures, but struggled with imbalanced classes

4. Final Results and Interpretation

Supervised Learning

Over all algorithms we run for SL, the best results are presented below and further summarized in Summary and Conclusion section

Logistic Regression:

- Final Model: Logistic Regression with default hyperparameters.

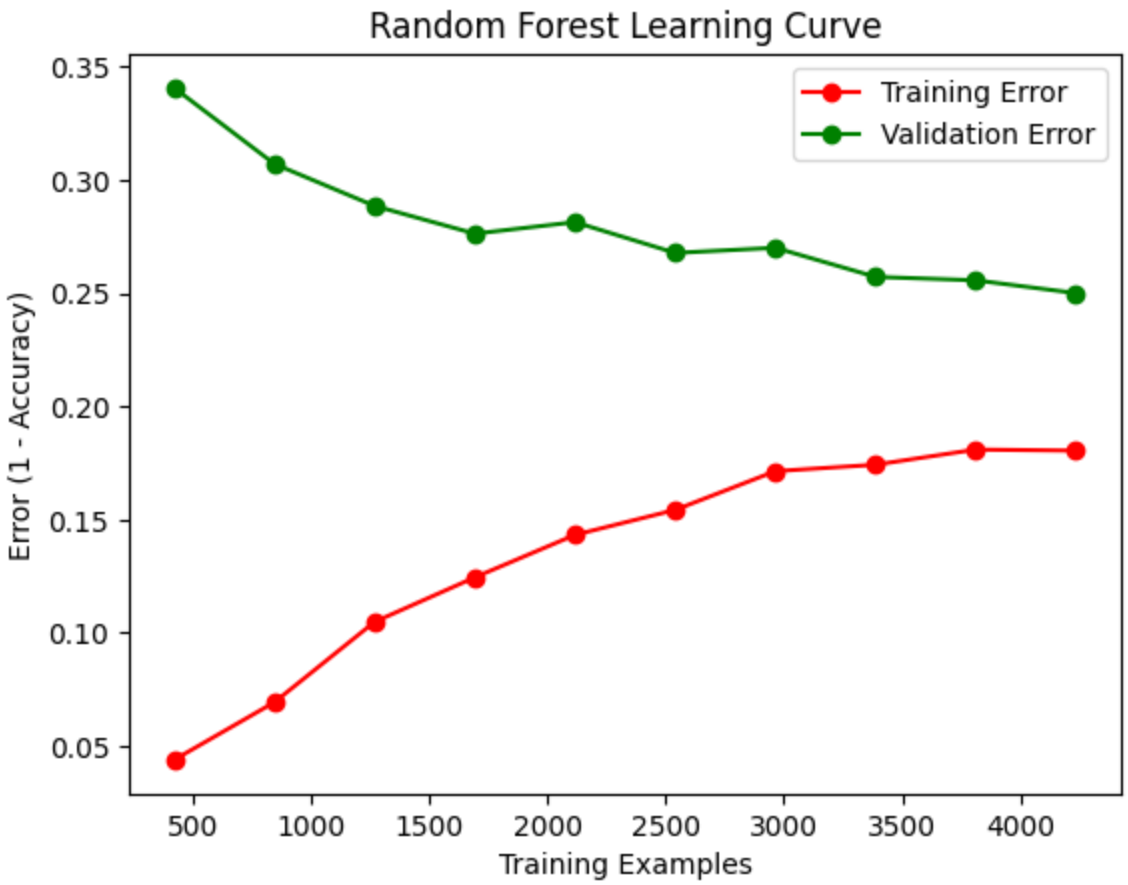
- Performance: 93.22% training accuracy and 82.46% test accuracy, showing good generalization with slight overfitting.
- Key Insights:
    - The model achieves high performance overall but struggles with certain classes due to imbalanced precision and recall, particularly for class 1 (low precision, high recall) and class 5 (low recall).
    - Precision values range from 64% to 98%, and recall ranges from 72% to 93%, highlighting variability in how well the model identifies true positives across classes.
    - Macro and weighted F1-scores are 83%, indicating balanced performance across classes, but class imbalance impacts specific predictions.
    - Further improvements could involve rebalancing the dataset or optimizing features for underperforming classes.

CART

- Final Model: Baseline (Depth=5).
- Performance: 78% test accuracy, balancing bias and variance better than the tuned model (Depth=10).
- Key Insights:
    - Increasing depth reduces training error but leads to overfitting.
    - Cross-validation is critical for selecting model depth, especially for high-dimensional data

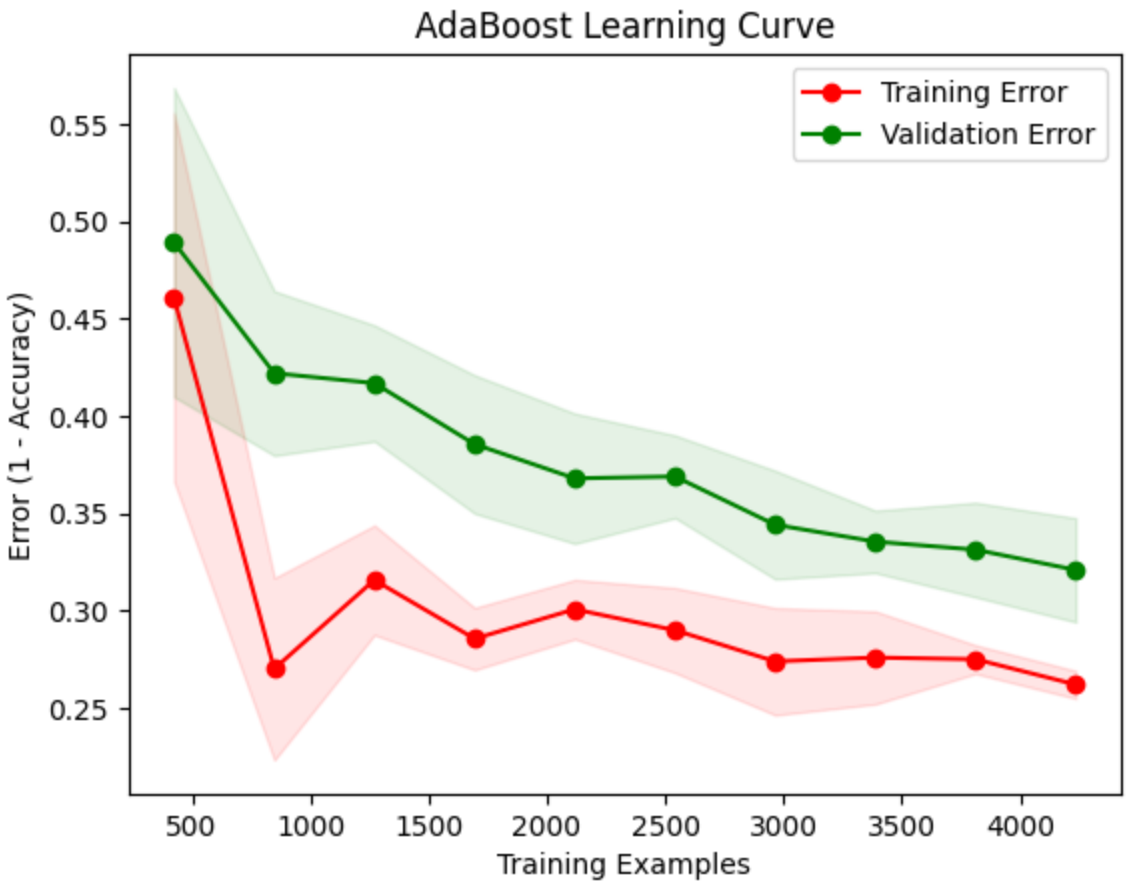
Random Forest

- Final Model: Tuned Random Forest Classifier (n_estimators=50, max_depth=5, min_samples_split=4).
- Performance: Test accuracy improved from 79% (baseline) to 80% after tuning, indicating minimal overfitting.
- Key Insights:
    - Training errors stabilized at ~18%, while validation errors plateaued at ~25%, reflecting good generalization.
    - Hyperparameter tuning marginally reduced test error, emphasizing ensemble stability.



AdaBoost

- Final Model: Tuned AdaBoost (n_estimators=200, learning_rate=0.1, max_depth=3).
- Performance: Achieved 83% test accuracy, significantly outperforming the default configuration.
- Key Insights:
    - Tuning n_estimators and learning_rate improved ensemble accuracy.
    - Allowing a slightly deeper base estimator captured more complex decision boundaries.
    - Highlighted AdaBoost's strength in distinguishing static and dynamic activities with proper tuning.



SVM

- Final Model: Support Vector Classifier (RBF kernel, C=10, gamma='scale').
- Performance: Achieved 86% test accuracy.
- Test Result:
  - Macro Average Precision: 88%
  - Recall: 86%
  - F1-Score: 86%
- Training Results
  - Macro Average Precision: 95%
  - Recall: 95%
  - F1-Score: 95%
- Key Insights:
  - The RBF kernel effectively captured non-linear decision boundaries.
  - Hyperparameter tuning via GridSearchCV was essential to exploit SVM's full potential.
  - Semi-supervised pseudo-labeling expanded the training set, improving performance.

Semi-Supervised Learning

Over all algorithms for SSL we run and optimized, the best model was the Semi-Supervised Support Vector Machine (S3VM)

1. S3VM
    - Final Model: Optimized S3VM (C=10.0, linear kernel, gamma='scale').
    - Performance: Achieved 92.64% test accuracy and 94.37% cross-validation accuracy.
    - Key Insights:
        - Pseudo-labeling effectively leveraged unlabeled data, enhancing model generalization.
        - Semi-supervised learning significantly improved performance over supervised approaches.
        - The dataset's activity classes exhibit a strong correlation with specific sensor features (e.g., triaxial acceleration for dynamic activities like walking). S3VM could exploit this correlation, even in the presence of sparse labels.
        - A linear kernel aligned with the dataset's largely linear separability, avoiding overfitting while benefiting from the added unlabeled samples.
        - Tuning regularization strength (C) and kernel parameters ensured the S3VM model struck the right balance between margin maximization and generalization. The linear kernel and carefully chosen parameters reduced complexity while maximizing performance.
2. EM
    - Performance: Final test accuracy of 16%.
    - Key Insights:
        - Poor performance was attributed to insufficient labeled data for initializing the Gaussian Mixture Model (GMM).
        - Highlighted the sensitivity of EM-based methods to initialization and dataset size.

3. Label Propagation
    - Performance: Final test accuracy of 16%.
    - **Final Model:** Label Propagation with an RBF kernel and gamma=20.
    - **Performance:**
    - **Training Accuracy:** 1.0, reflecting perfect label propagation on the training set.
    - **Test Accuracy:** 74.96%, demonstrating moderate generalization.
    - **Key Insights:**
        - Precision ranged from 0.70 to 0.89, and recall ranged from 0.61 to 0.92, showing variability across classes.
        - Class 1 exhibited the weakest performance with a precision of 0.71 and recall of 0.61, indicating challenges with minority class identification.

- The algorithm effectively propagates labels within clusters but struggles with imbalanced data and noisy boundaries, impacting its ability to generalize to unseen samples.
- Future improvements could involve enhancing the similarity graph with advanced kernel functions.

5. Implementation for the extension (USL)

Previous sections are for the main part of your project (SL and SSL). Briefly repeat section 3 here for the case of USL or HI. In this section, please just put anything different for the extension here, such as dataset methodology. For duplicate part such as feature extraction, you can simply state that they are the same.

5.1 Dataset Methodology

The dataset for USL combines both the original train and test sets, resulting in a total of 10,299 samples, and 561 features. There was no need to separate the dataset since the clustering quality metrics are used here to evaluate the clustering quality instead of the traditional training-validation-test splits.

The activity labels were removed to align with the unsupervised learning methodology.

5.2 Preprocessing, Feature Extraction, Dimensionality Adjustment

The sensors reading data were standardized to improve the clustering and normalize scale differences. Building upon the feature engineering techniques applied in SL/SSL, the feature space for USL was evaluated further through multiple feature engineering trials to evaluate the impact on clustering quality and optimal number of clusters (k).

The sub-sections below explain the feature engineering and feature extraction techniques applied in three different trials and the dimensionality reduction implemented using PCA to capture key components. For the baseline system with 6 clusters, the model used the original feature space and there were no feature engineering techniques or PCA applied.

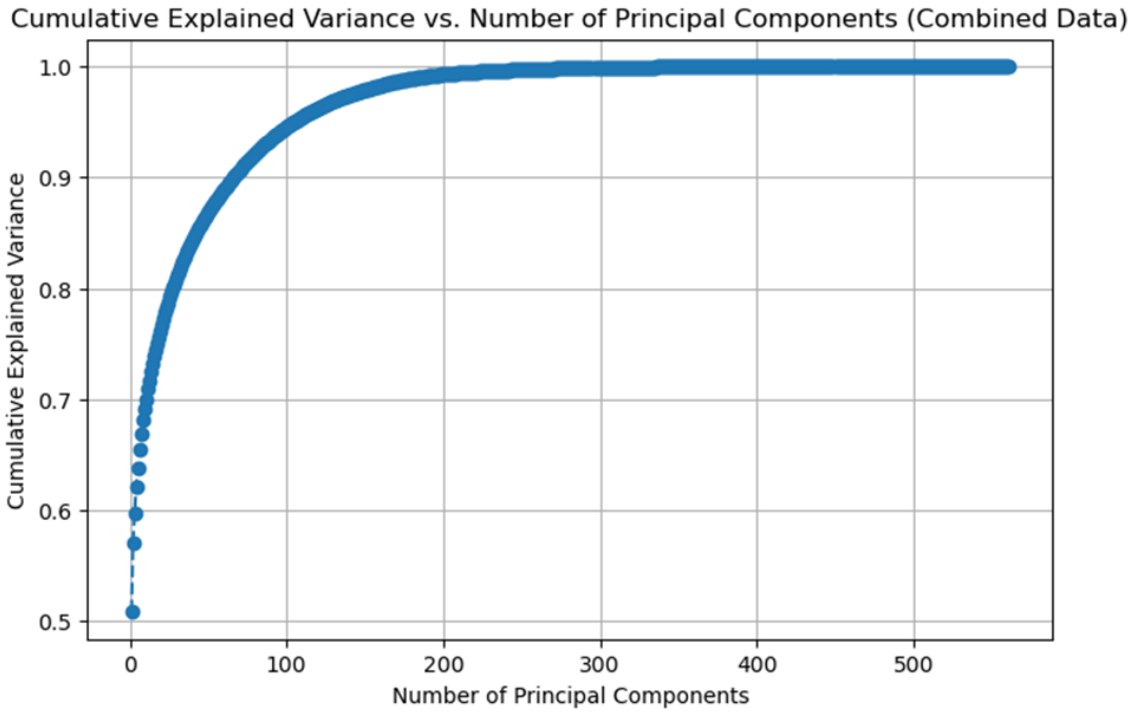
5.2.1 First Trial

Feature Engineering and Features extraction:

In the first trial, the same feature engineering techniques previously applied in SL and SSL were used which includes statistical features such mean and standard deviation magnitude which capture overall motion intensity, range (min-max difference) which capture extreme motions, and interaction terms between accelerometer and gyroscope which capture combine movement.

PCA Dimensionality Adjustment:

The cumulative variance chart for the first trial shows that 99% of the variance was retained using 184 components which significantly reduce the feature space from more than 561 features.



5.2.2 Second Trial

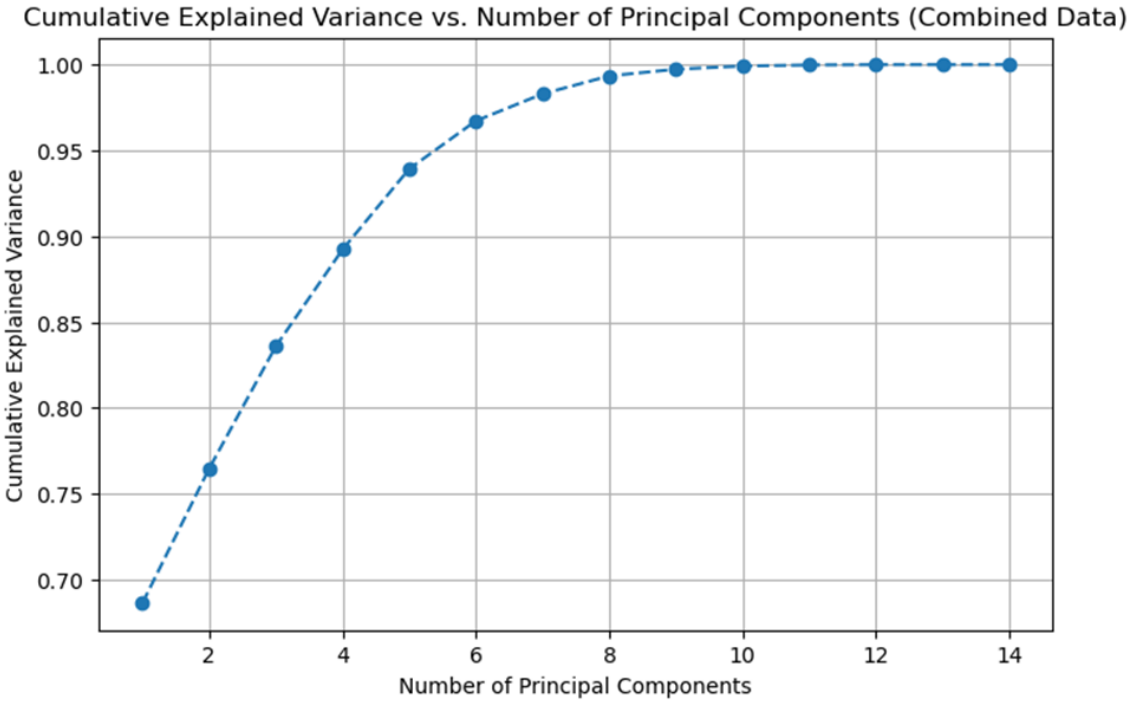
Feature Engineering and Features extraction:

In the second trial, features were separated into two subsets: Accelerometer (10,299 samples, 345 features) and Gyroscope Features (10,299 samples, 213 features). Statistical measures were applied independently to each sensor, including mean, std, min, max, skew, kurtosis and range.

All algorithms applied for the first trial were re-applied here again, interestingly results were slightly better even though the number of features were reduced from 561 to 14 before the PCA.

PCA Dimensionality Adjustment:

As shown in the below chart, the cumulative variance reached around 99% with 12 components out of the 14 statistical features from both sensors.



5.2.3 Third Trial

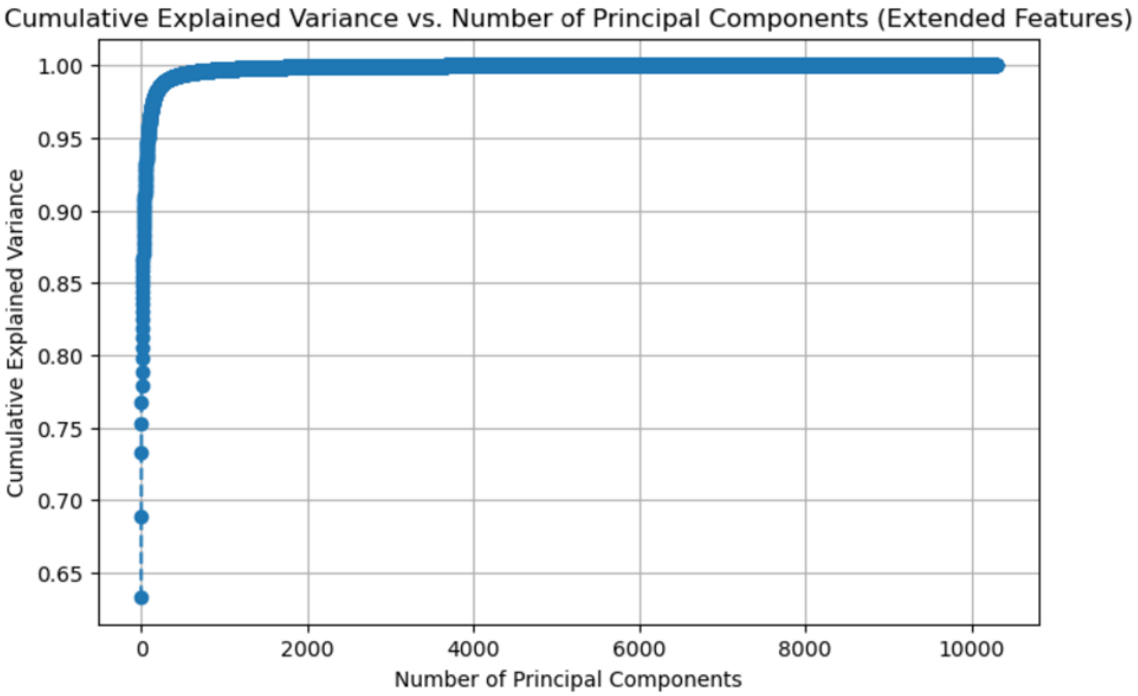
Feature Engineering and Features extraction:

Building upon the second trial, additional complexity was introduced by Frequency-Domain Features (FFT) which capture periodic patterns by transforming time-domain signals into frequency-domain. This will help in distinguishing frequency of the activities like walking which exhibit repetitive motion from stationary activities like sitting. Moreover, the product among all feature combinations showing pairwise interactions were introduced to explore non-linear relationships.

This trial was computationally expensive and increased the feature count dramatically to 81,936. However, after applying PCA retaining 97% variance, the dimensionality was reduced to 171 features. This suggests that the extended feature space contained significant redundancy in interactions and periodic patterns, which PCA effectively captured the key features crucial for clustering.

PCA Dimensionality Adjustment:

As demonstrated in the third chart, most of the variance was captured with fewer than 500 components which is aligned with previous trials and original dataset even despite the dramatic increase in the feature space dimensionality. PCA was applied to retain 97% of the variance which reduced the feature space to 171 components. This highlights the observation of significant feature redundancy introduced in this trial.



5.3 Training Process

The training process for USL aimed on discovering patterns in the Human Activity Recognition dataset by applying various clustering algorithms. Unlike SSL and SL, the clustering is performed without any activity labels mirroring real-life scenarios for USL. The Key Implementation milestones are outlined below:

1. Clustering algorithms used:
    a. K-Means.
    b. Expectation Maximization (EM-GMM)
    c. Agglomerative Clustering
    d. K-Medoids.
2. Optimization of Cluster Count (K):
    a. Each of the four clustering algorithms was trained across a range of k values, from k=2 to k=14.
    b. Separate evaluations were conducted for each feature engineering trial and dimensionality reduction (detail in 5.2),
    c. The optimal number of cluster (K) was determined based on the comprehensive evaluation for all metrics
        i. Metric Used for all algorithms: Silhouette Score, Calinski-Harabasz Index, Davies-Bouldin Index.
        ii. Algorithm specific Metrics:
            1. Stepwise Criterion for Agglomerative Clustering

Result for each feature engineering trial:

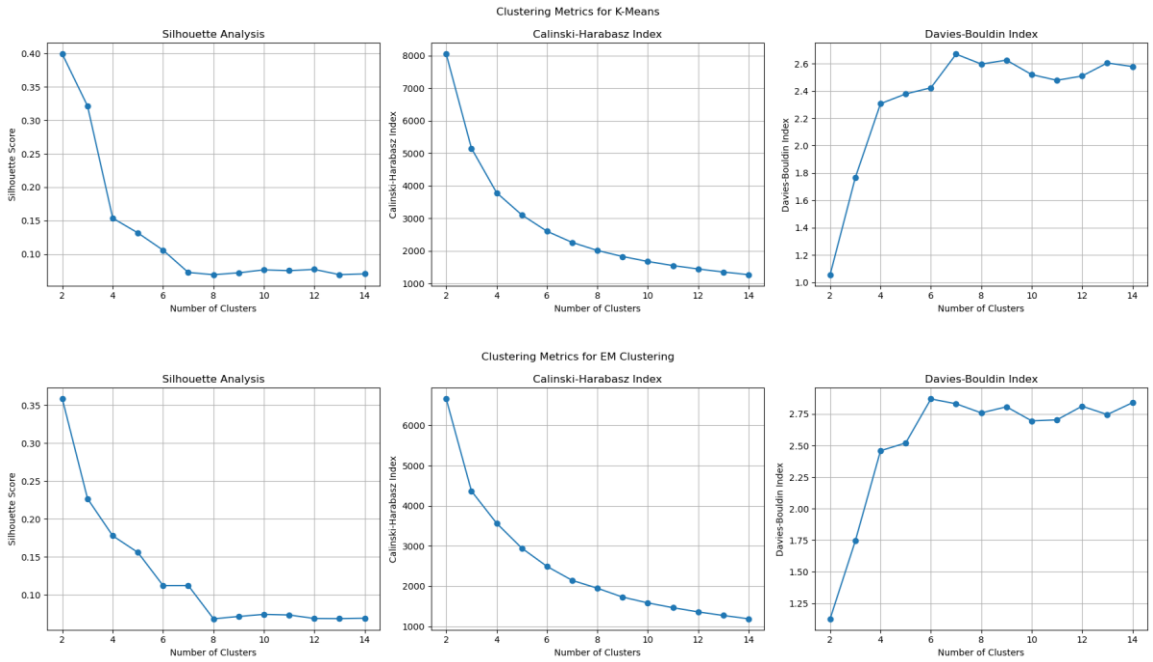
1. **First Trial:**

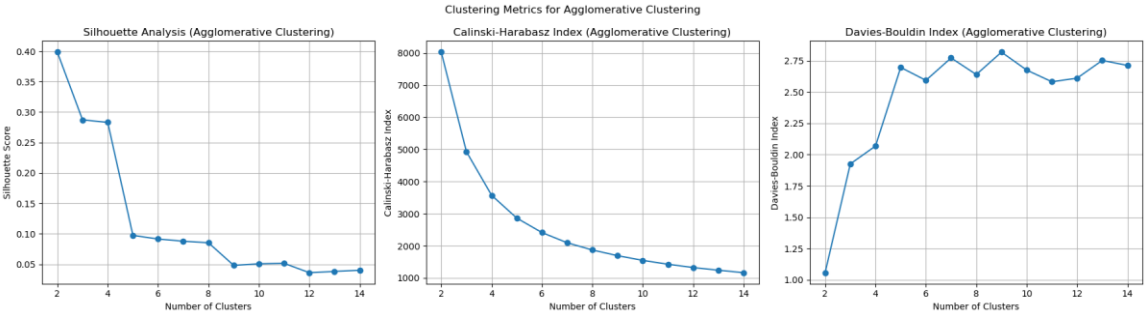
All the four algorithms have shown similar trends across the three general metrics (Silhouette Score, Calinski-Harabasz Index, and Davies-Bouldin Index). The optimal number of clusters was determined to be 2 according to all the three measures, which achieved a significantly higher score than the baseline(0.35-0.5 compared to around 0.1 in the baseline).

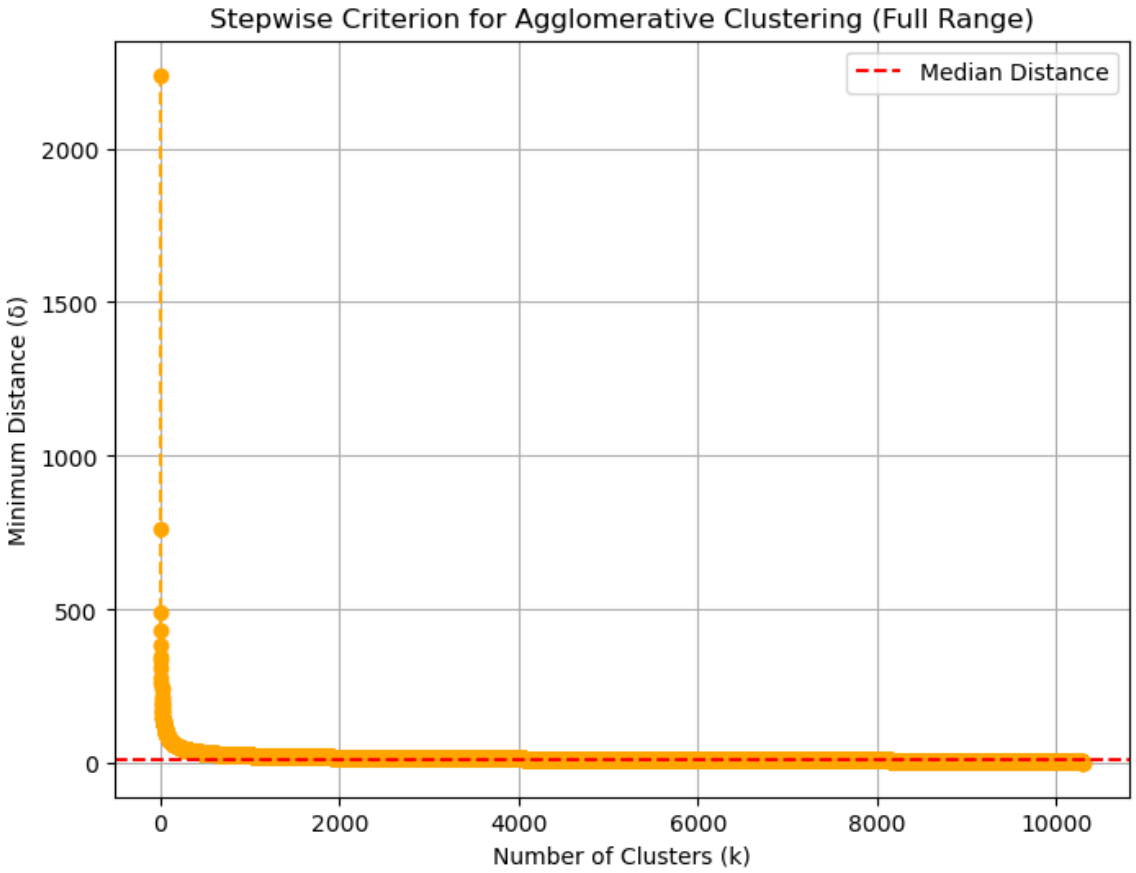
However, Silhouette score for all algorithms remained below 0.5 even for all algorithms with optimal k indicating a room for further improvement. For 6 clusters, compared to the baseline the scores were similar and feature engineering did not significantly improve clustering performance, the only notable improvement observed t for K-Medoids was improved from 0.06 to 0.09.

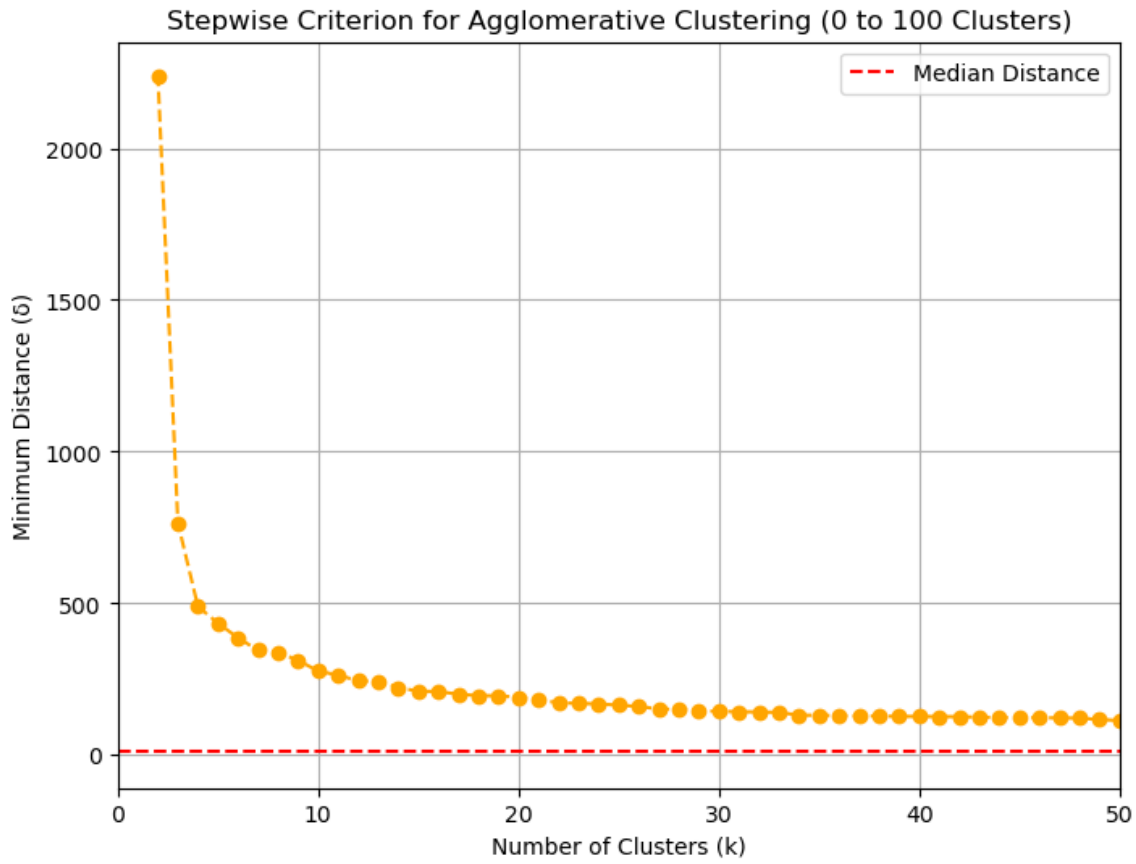
Additionally, the specific metric used in Agglomerative clustering confirmed 2 as the optimal number of clusters, indicated by a significant jump in the minimum distance difference from 2 to 3 clusters.

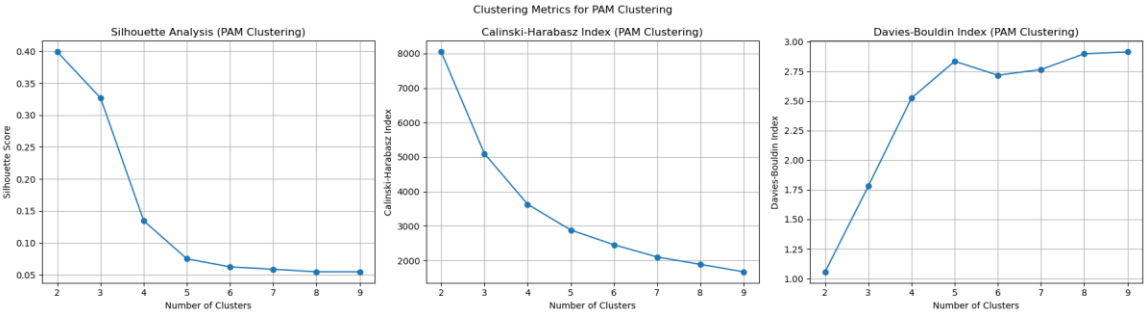
The results for the first trial are shown below for all algorithms.




Clustering Metrics for Agglomerative Clustering
Silhouette Analysis (Agglomerative Clustering)
Silhouette Score
Number of Clusters
Calinski-Harabasz Index (Agglomerative Clustering)
Calinski-Harabasz Index
Number of Clusters
Davies-Bouldin Index (Agglomerative Clustering)
Davies-Bouldin Index
Number of Clusters


Stepwise Criterion for Agglomerative Clustering (Full Range)
Median Distance
Minimum Distance (δ)
Number of Clusters (k)

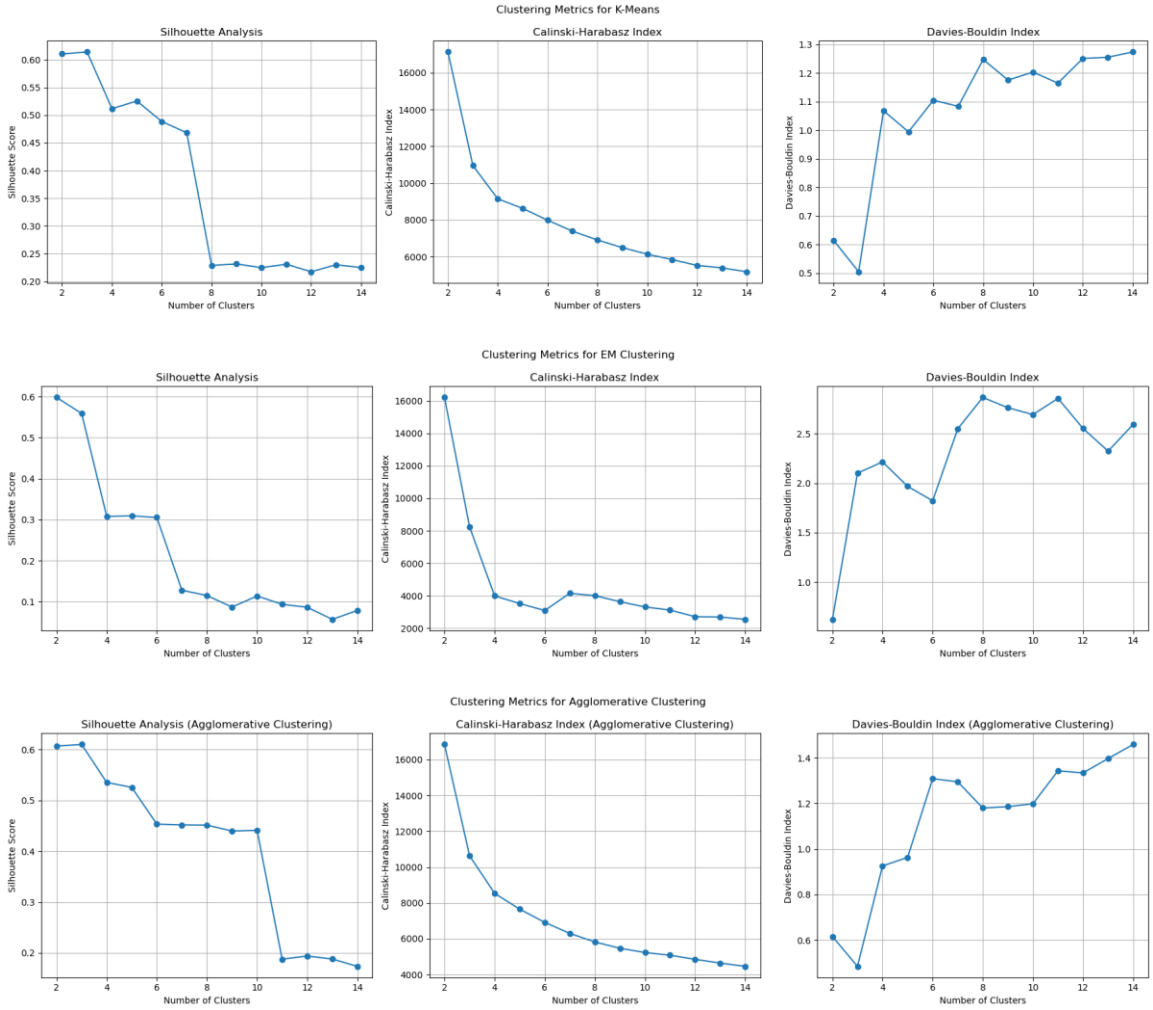


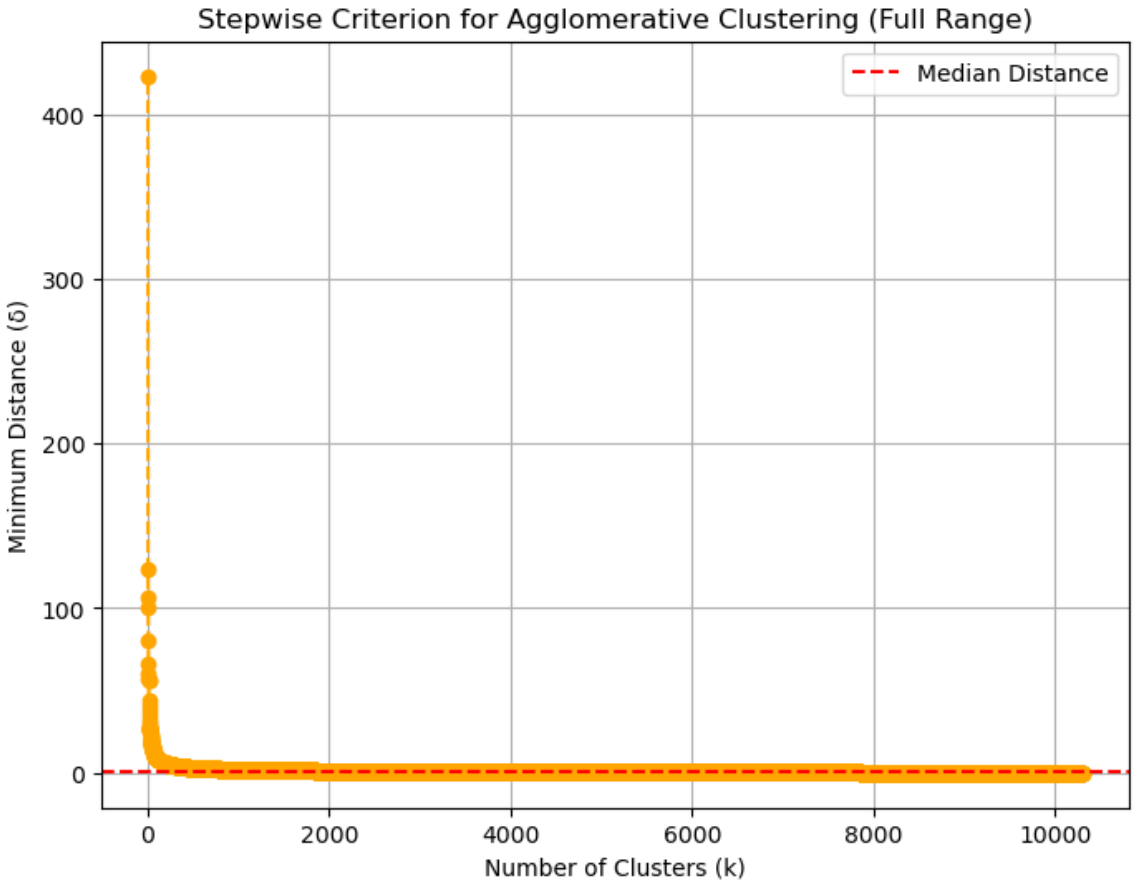


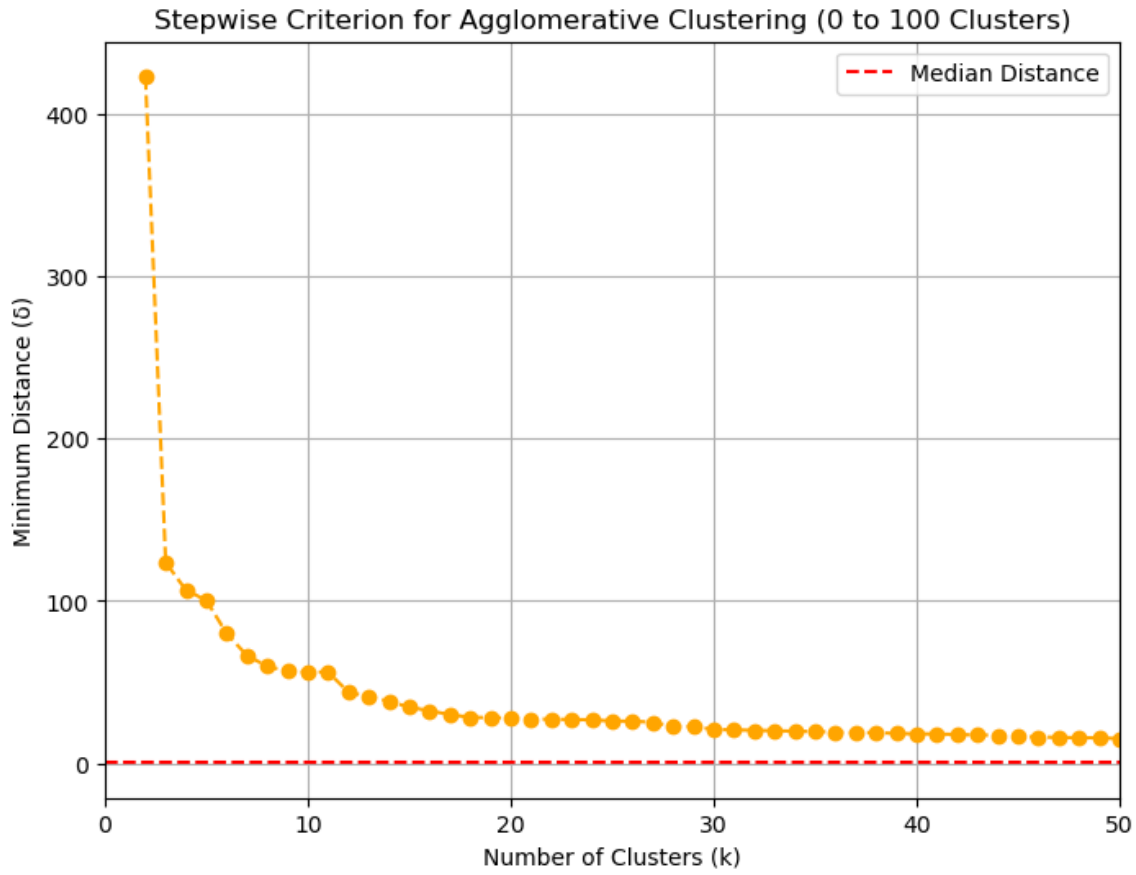
2. Second Trial:

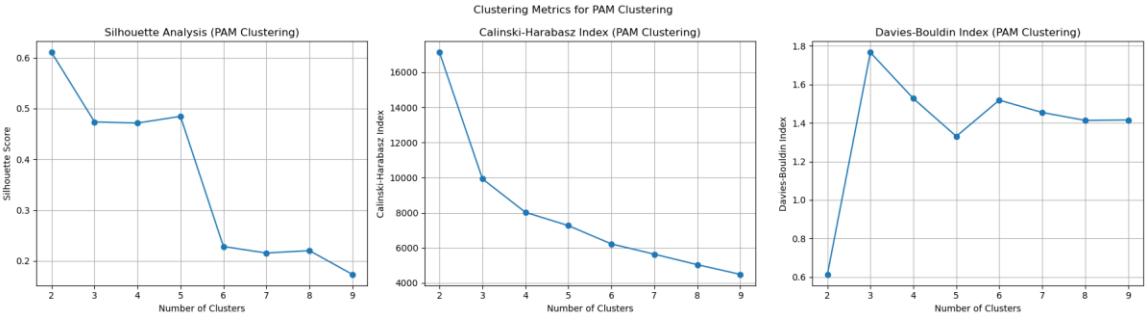
Similar trends were observed as in Trial 1, but scores improved across all algorithms and metrics. Scores for up to 5-6 clusters improved to acceptable range for Slhouette score and even more than 10 clusters using Davis-Boulden for K-Means and Agglomerative Clustering. Notably, k=2 and k=3 scored above 0.6 for the two lastly mentioned algorithms. Despite the slightly higher performance for k=3 in Slhouette and Davis-Boulden for K-Means and Agglomerative Clustering , the stepwise criterion confirmed k=2 as the optimal number of clusters. Also, the K-Medoids (PAM) and EM improved to above 0.5, but still 2 remained the optimal number of clusters.

The results for the second trial are shown below for all algorithms.




Stepwise Criterion for Agglomerative Clustering (Full Range)
Median Distance
Minimum Distance (δ)
400
300
200
100
0
0
2000
4000
6000
8000
10000
Number of Clusters (k)

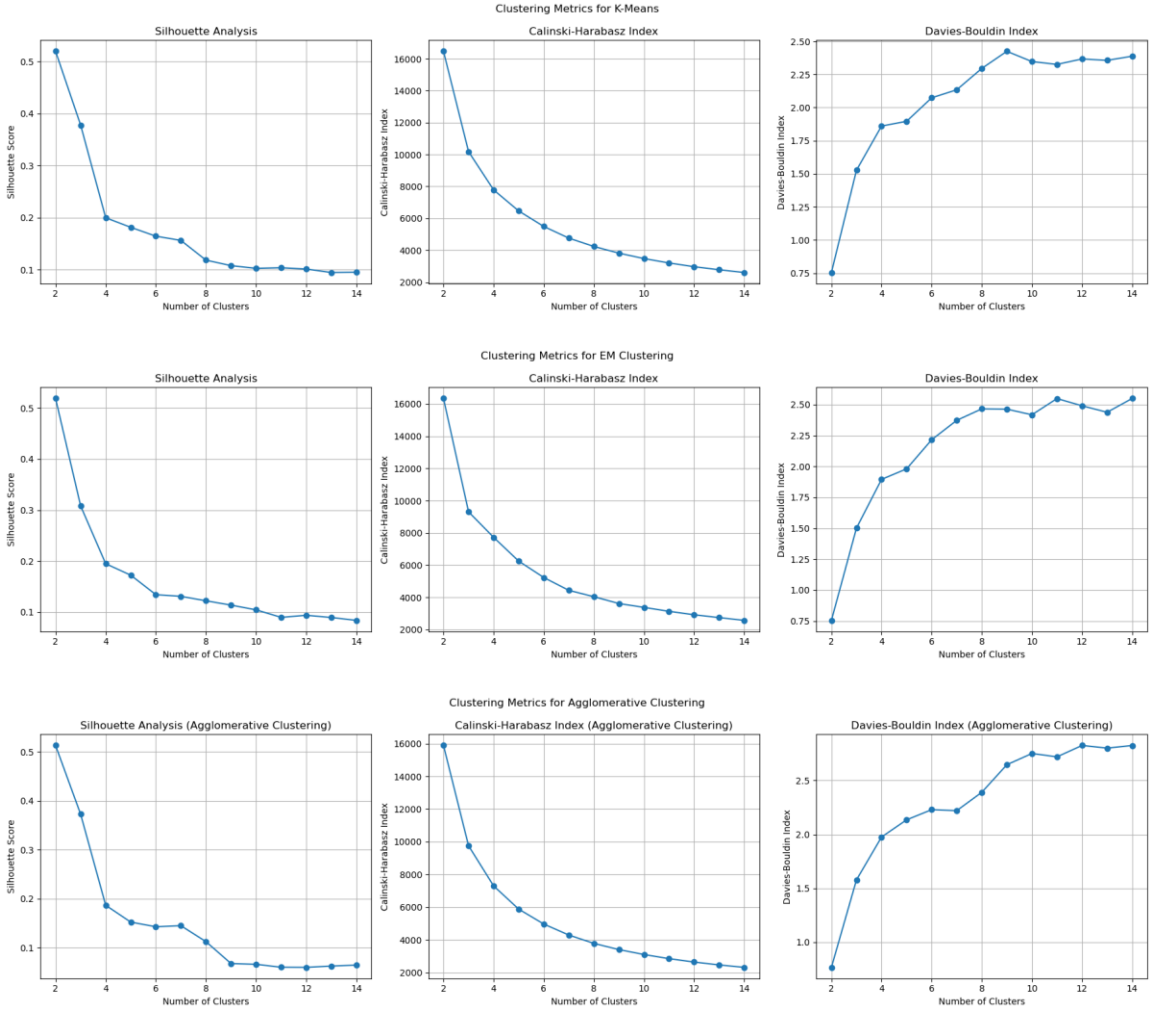


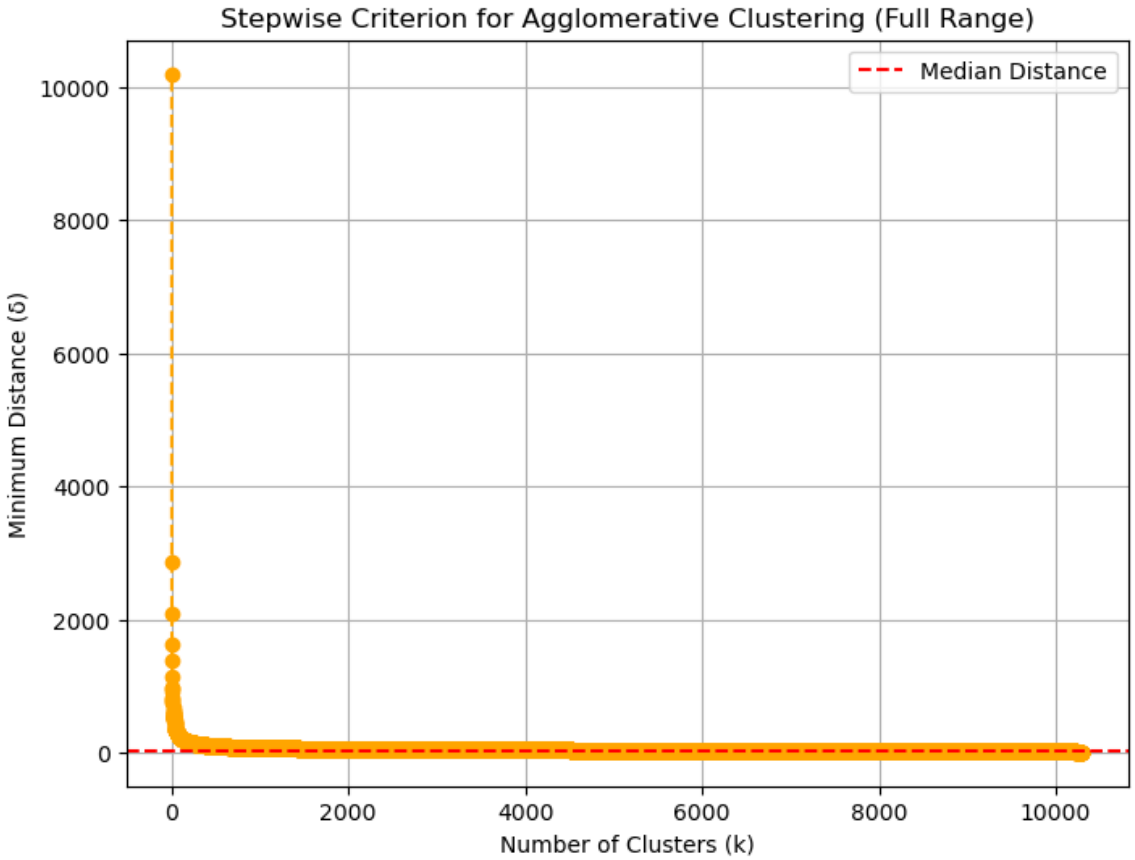


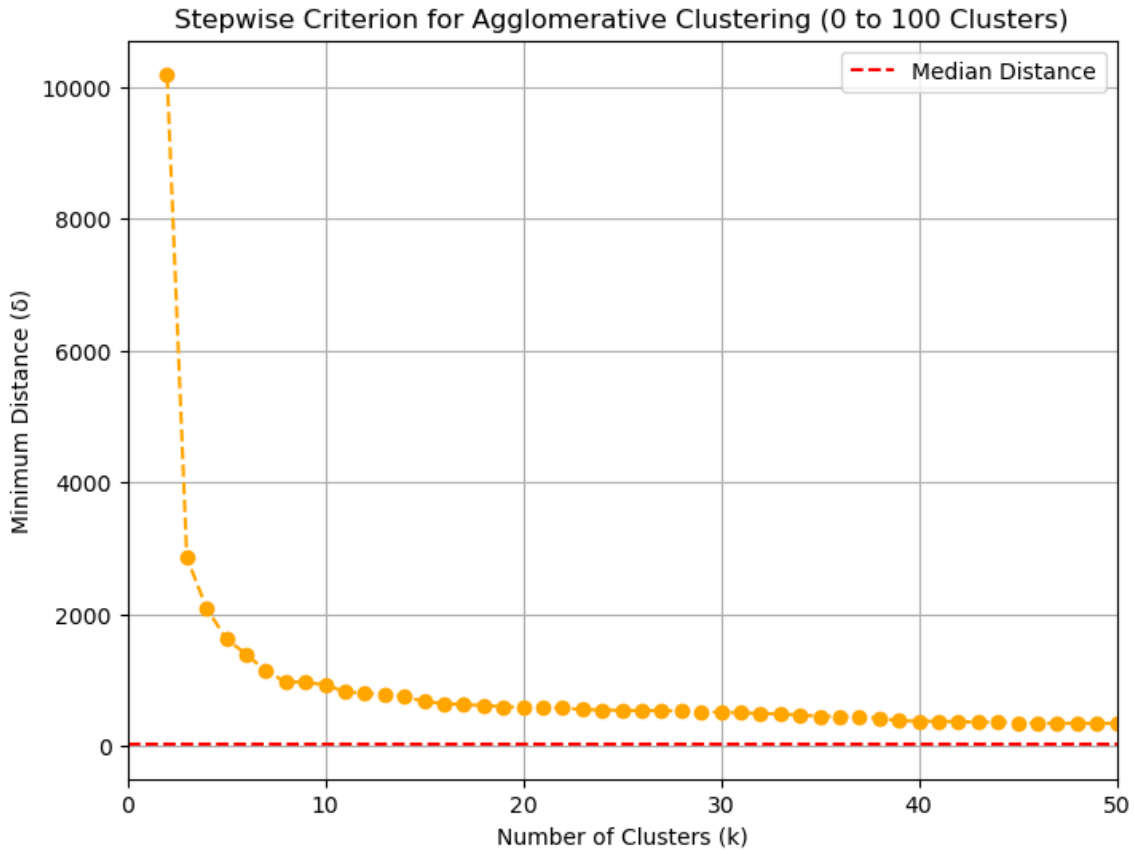
3. Third Trial:

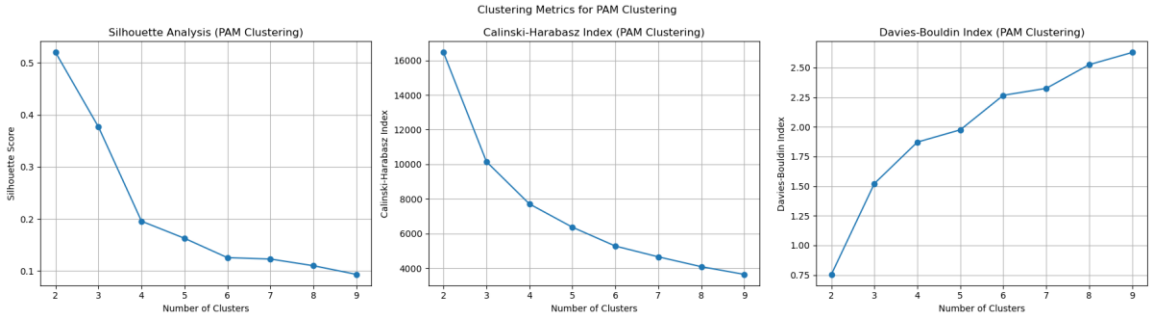
The third trial performed better than the first trial but didn't suppress the second trial performance despite incorporating advanced feature engineering. It followed similar trends as other trials but with higher scores compared to the first trial. The optimal number of clusters remained 2 with above 0.5 in the Silhouette score, below 1 in the Davis-Bouldin Index and Calinski-Harabasz Index remained the highest at k=2.

The results for the third trial are shown below for all algorithms.


Clustering Metrics for K-Means
Silhouette Analysis
Silhouette Score
Number of Clusters
Calinski-Harabasz Index
Calinski-Harabasz Index
Number of Clusters
Davies-Bouldin Index
Davies-Bouldin Index
Number of Clusters
Clustering Metrics for EM Clustering
Silhouette Analysis
Silhouette Score
Number of Clusters
Calinski-Harabasz Index
Calinski-Harabasz Index
Number of Clusters
Davies-Bouldin Index
Davies-Bouldin Index
Number of Clusters
Clustering Metrics for Agglomerative Clustering
Silhouette Analysis (Agglomerative Clustering)
Silhouette Score
Number of Clusters
Calinski-Harabasz Index (Agglomerative Clustering)
Calinski-Harabasz Index
Number of Clusters
Davies-Bouldin Index (Agglomerative Clustering)
Davies-Bouldin Index
Number of Clusters


Stepwise Criterion for Agglomerative Clustering (Full Range)
Median Distance
Minimum Distance (δ)
Number of Clusters (k)
0
2000
4000
6000
8000
10000


Stepwise Criterion for Agglomerative Clustering (0 to 100 Clusters)
Median Distance
Minimum Distance (δ)
Number of Clusters (k)
0
2000
4000
6000
8000
10000
0
10
20
30
40
50


Clustering Metrics for PAM Clustering
Silhouette Analysis (PAM Clustering)
Silhouette Score
Number of Clusters
Calinski-Harabasz Index (PAM Clustering)
Calinski-Harabasz Index
Number of Clusters
Davies-Bouldin Index (PAM Clustering)
Davies-Bouldin Index
Number of Clusters

6 Final Results and Interpretation for the extension (USL)

The best results were achieved in trial 2 and shown below the obtained metrics for k=2 (optimal number of clusters)

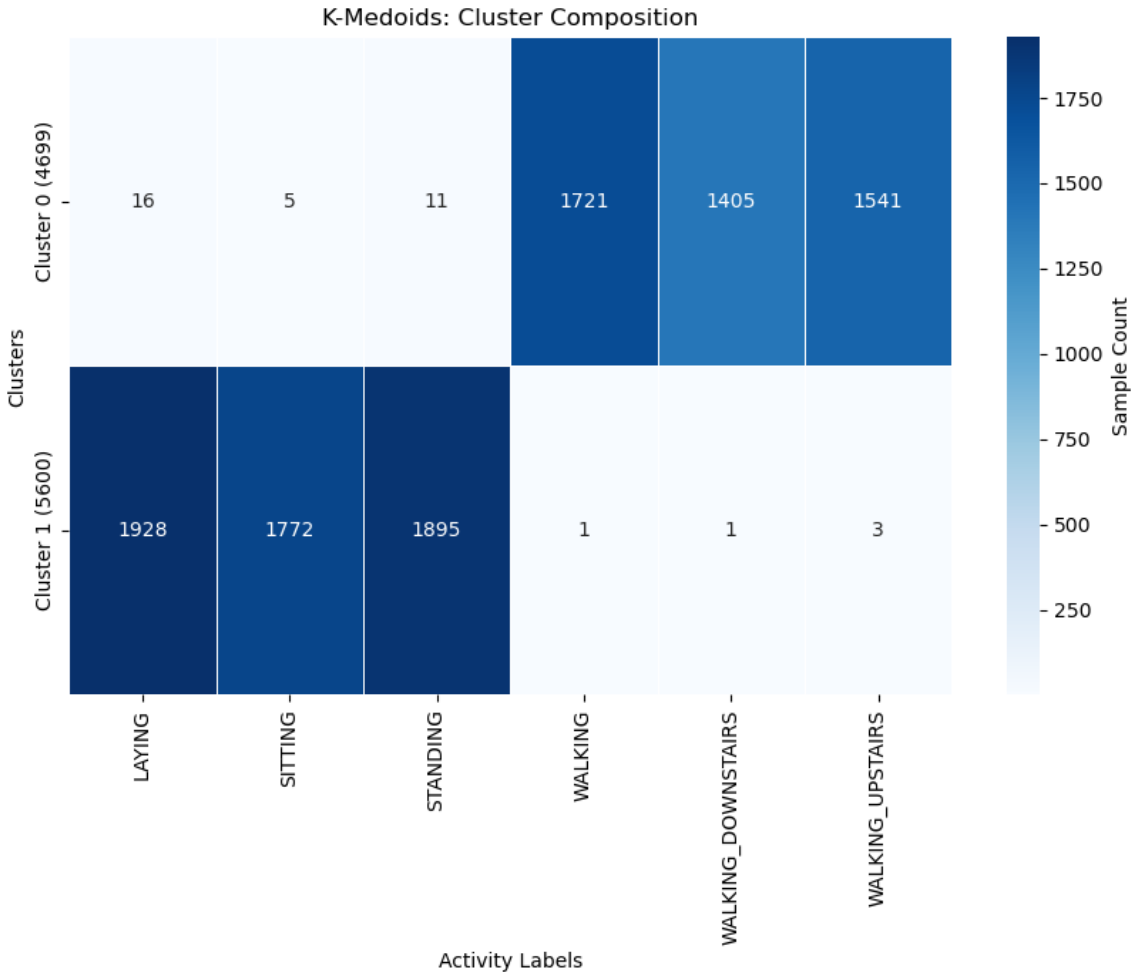
Clustering Algorithm	Silhouette Score	Calinski-Harabasz Index	Davies-Bouldin Index
K-Means	0.610	17131.67	0.614
GMM	0.598	16211.81	0.622
K-Medoids	0.610	17124.12	0.613
Agglomerative	0.606	16852.79	0.616

All algorithms achieve similar performance in the second trial. Trial 2 emphasizes on the importance of selecting relevant features over increasing the dimensionality and complexity of the models.

The silhouette score for all algorithms was around 0.6. The K-means and K-Medioids were the highest in Silhouette score with 0.61 and both showed lower Davies-Bouldin Index and higher Calinski-Har index.

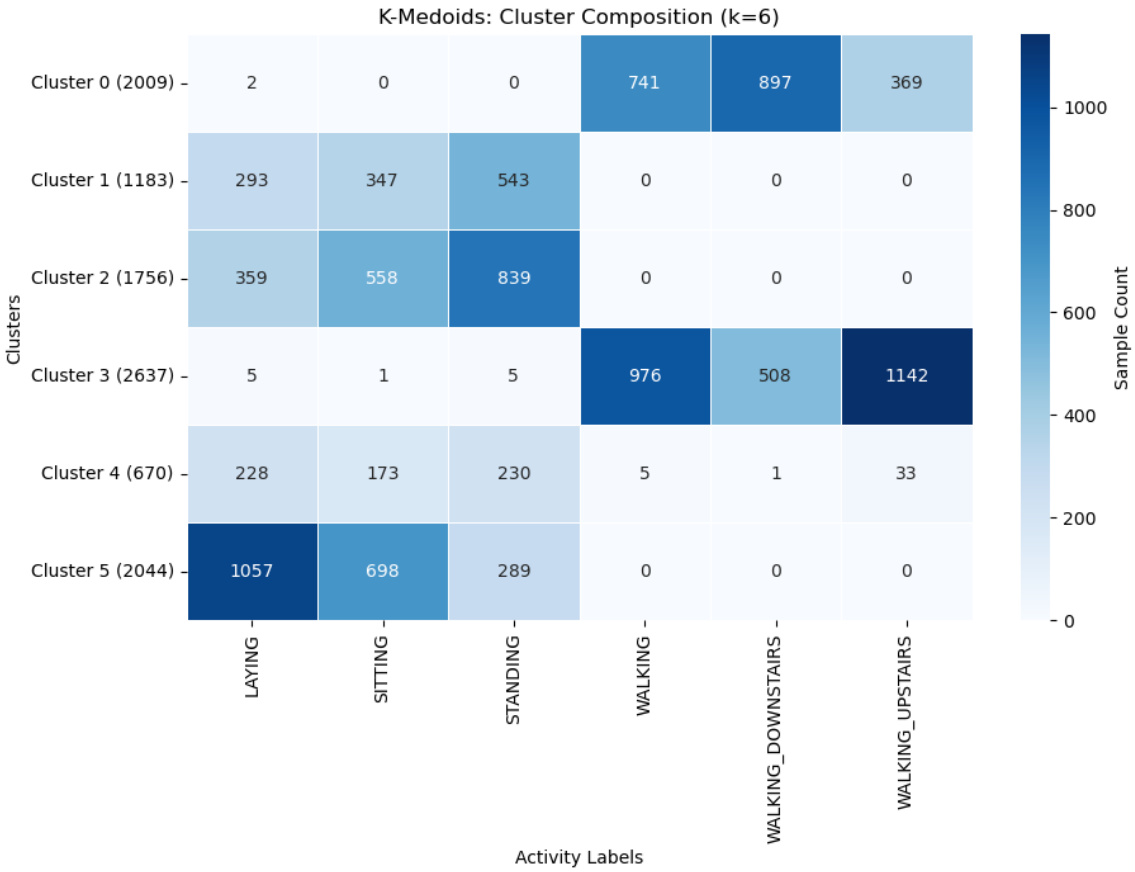
Therefore, although they were all comparable performance using 2 clusters. The optimal Model was chosen to be K-Medoids due to its highest performance in Silhouette score (0.61) and lowest in Davies-Bouldin index (0.63).

To further evaluate the optimal model, we compared the composition of the two clusters with original activity labels. Below is the confusion matrix visualization.



It's very insightful that the clustering model was able to capture the movement patterns and separate the activities into Static Activities (LAYING, SITTING, and STANDING) and Dynamic activities (WALKING, WALKING_DOWNSTAIRS, and WALKING_UPSTAIRS). This aligns with the expected intuition in capturing major movement and change in the body acceleration. Also, despite the simplicity of feature engineering done in trial 2, it outperforms more complex approaches like trial 3 with FFT features. This demonstrates the importance of keeping the model with relevant feature selection over increasing its complexity.

Furthermore, we compared the original labels with 6 clusters (not the optimal) to compare with the original number of labels. Cluster 1,2, 4 and 5 show significant overlap in capturing static activities. This suggests that the model struggles to distinguish the subtle differences between these activities, likely due to similar sensors reading from accelerometer and gyroscope. Cluster 0 and 3 captures the dynamic activities including a different walking setup. The overlap between the dynamic activities suggests the similar observation for static activities that confuse the clustering algorithm. This confirms the earlier finding that 2 is the best number of clusters which shows clean separation between clusters.



7 Contributions of each team member

We have worked jointly in almost all parts of the group with fairly and equal distribution of the load. The project proposal, reports and codes were created by both team members and some parts were led by each member as follows

Mohammed Alsaraihi:

- Led the USL extension, including design, implementation, clustering analyses, and evaluation. Conducted clustering optimization using metrics and documented USL findings.
- Participated in SL/SSL, contributing to preprocessing, conducted the feature engineering, and feature selection and added and finalized the evaluation metrics.
- Reviewed and advised to improve the models for SL/SSL as needed.

Tanvi Gandhi:

- Proposed the dataset idea and overall project plan.

- Focused on SL and SSL, handling dataset preparation, hyperparameter tuning, and model evaluations.
- Participated in USL by reviewing the advice to improve the USL models and metric.

8 Summary and conclusions

The **Optimized Logistic Regression** demonstrated superior performance, achieving a test accuracy of 89.99%, with balanced precision, recall, and F1-scores across all classes. Its success can be attributed to effective L1 regularization, which encouraged sparsity in feature weights and mitigated overfitting. This made the model robust to noise while maintaining its interpretability and computational efficiency. Logistic regression's strong performance highlights its suitability for datasets with linear separability, where it balances bias and variance effectively. Unlike models such as **CART** and **Random Forest**, which exhibited sensitivity to hyperparameters and a tendency to overfit deeper configurations, logistic regression capitalized on its simpler structure to achieve consistently high generalization across the test data.

The **Semi-Supervised Support Vector Machine (S3VM)** leveraged unlabeled data effectively, achieving competitive results through pseudo-labeling and iterative refinement of decision boundaries. The linear kernel S3VM, in particular, excelled by aligning with the dataset's structural simplicity, achieving test accuracies comparable to optimized logistic regression. However, models like **EM**, which relied heavily on unlabeled data, struggled due to class imbalance and high-dimensional features, resulting in poor convergence. Comparatively, S3VM's ability to integrate labeled and pseudo-labeled data allowed it to outperform several supervised models, including baseline **Logistic Regression**, in scenarios with limited labeled data. These results underscore the importance of matching model complexity to dataset characteristics, where logistic regression thrives in linear regimes, and S3VM offers a robust approach for leveraging unlabeled data in semi-supervised settings.

The USL extension focused on clustering human activity data without labels using four algorithms: K-Means, K-Medoids, GMM, and Agglomerative Clustering. Various feature engineering trials were conducted, with Trial 2, featuring simple statistical features, achieving the best results. The optimal number of clusters was consistently k=2, and K-Medoids emerged as the best model (Silhouette Score: 0.61), effectively separating static and dynamic activities. For k=6 significant overlapping among activities were found highlighting limitations in the sensor's readings and feature distinctions across all trials

9 References

[1] Ketan Rajshekhar Shahapure, Charles Nicholas. "Cluster Quality Analysis Using Silhouette Score." 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), 2020. DOI: 10.1109/DSAA49011.2020.00096.

[2] Lecture 11, Dr. B. Keith Jenkins. "Machine Learning IIB: Learning Realms and Algorithms." EE 599, November 13, 2024, Page 3-6.