Ihsan Dogramaci Bilkent University
GE 461: Introduction to Data Science
Project W13: Telehealth - Fall Detection Report

Name and Surname: Ali Aral Takak
Student ID: 22001758
Department: EEE

Introduction

This report describes our approach to using wearable sensor data for fall detection. In the first part, we performed unsupervised analysis by reducing the high-dimensional sensor data using Principal Components Analysis (PCA) and then applied k-means clustering to reveal natural groupings in the data. We compared these clusters with the known action labels to assess whether the sensor features can distinguish between falls and non-falls. In the second part, we took a supervised learning approach by splitting the dataset into training, validation, and test sets. We built two classifiers—a Support Vector Machine (SVM) and a Multi-Layer Perceptron (MLP)—and tuned their hyperparameters to achieve the best performance in detecting falls.

Mathematical Background and Theory

Principal Component Analysis

Principal Component Analysis (PCA) is a statistical unsupervised learning technique used to reduce the dimensionality of a dataset while preserving as much variance as possible. Mathematically, assume we have a data matrix $X \epsilon R^{n \times p}$ where n is the number of observations and p is the number of features. The first step is to center the data by subtracting the mean of each feature, hence each column of X has a mean of zero. The goal of PCA is to find a set of orthogonal vectors (principal components) $\{\omega_1, \omega_2, \dots, \omega_p\}$ that capture the directions of maximum variance in the data. The first principal component, ω_1, is the unit vector that maximizes the variance of the projected data [1]:

$$\omega_1 = argmax_{||\omega||=1} Var(X\omega) = argmax_{||\omega||=1} \omega^T S \omega$$

Where $S = \frac{1}{n-1} X^T X$ is the sample covariance matrix. Subsequent components $\omega_2, \omega_3, \dots$ are chosen similarly, with the limitation that they are orthogonal to the previously selected components [1]:

$$\omega_k = argmax_{||\omega||=1, \omega \perp \omega_1, \dots, \omega_{k-1}} \omega^T S \omega$$

This optimization task leads to the eigenvalue decomposition of the covariance matrix [1]:

$$S\omega_i = \lambda_i \omega_i$$

Where λ_i is the eigenvalue corresponding to the eigenvector ω_i. The eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$ represent the variance captured along each principal component.

K-Means Clustering

K-means clustering is an unsupervised learning algorithm that partitions a set of n data points $\{x_1, x_2, \dots, x_n\}$ in R^d into K clusters, each represented by a centroid. The aim is to assign each data point to the nearest cluster center such that the sum of squared distances between points and their corresponding centroid is minimized. Mathematically, the goal is to find clusters $C_1, C_2, \dots, C_K$ and their corresponding centroids $\mu_1, \mu_2, \dots, \mu_K$ that solves [2]:

$$min_{C_1,\dots,K} \sum_{k=1}^{K} \sum_{x_i \epsilon C_k} ||x_i - \mu_k||^2$$

Where the centroid of cluster C_k is defined as [2]:

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \epsilon C_k} x_i$$

The k-means algorithm follows the steps given below [2]:

1. **Initialization:** Randomly select K initial centroids.
2. **Assignment Step:** Assign each data point x_i to the cluster with the nearest centroid $k = argmin_j ||x_i - \mu_j||^2$
3. **Update Step:** Recalculate the centroids for each cluster using the current cluster members $\mu_k = \frac{1}{|C_k|} \sum_{x_i \epsilon C_k} x_i$.
4. **Convergence Check:** Repeat the assignment and update steps until the centroids no longer change significantly or a maximum number of iterations is reached.

Support Vector Machines (SVMs)

Support Vector Machines are supervised learning models designed for binary classification. Given a training dataset $\{(x_i, y_i)\}_{i=1}^{n}$ where $x_i \in R^d$ and $y_i \in \{-1, +1\}$, the goal of a linear Support Vector Machine is to find a hyperplane defined by a normal vector $\omega \in R^d$ and a bias $b \in R$ that separates the two classes with the maximum margin. The optimization problem for a linearly separable case is formulated as [3]:

$$minimize_{\omega,b} \frac{1}{2}||\omega||^2$$

$$subject\ to\ y_i(\omega^T x_i + b) \geq 1, \qquad i = 1, \dots, n$$

For non-linearly separable data, SVMs employ the kernel trick to map the input data into a higher-dimensional feature space where a linear separator might exist.

Multi-Layer Perceptron (MLP) Algorithm

An MLP is a type of feedforward neural network used for supervised learning tasks. It consists of an input layer, one or more hidden layers, and an output layer. Each layer l has a weight matrix W^l and a bias vector b^l. The network computes its output by propagating inputs forward through the layers. The mathematical formulation for this operation can be defined as [4]:

$$a^l = f(W^l a^{(l-1)} + b^l)$$

The network's prediction is compared to the true label using a loss function L, such as cross-entropy loss or mean squared error loss. Training the MLP also involves adjusting the weights and biases to minimize the loss using backpropagation combined with an optimization algorithm, such as the gradient descent. The weight updates are typically computed as [4]:

$$W^l \leftarrow W^l - \eta \frac{\partial L}{\partial W^l}$$

Where η is the learning rate. The backpropagation algorithm computes the gradients by propagating error backwards through the network.

Results for Part A

In Part A, we visualize the telehealth dataset by using Principal Component Analysis, and projecting the data two two dimension. The figure given below displays the PCA explained variance ratio versus the principal components:

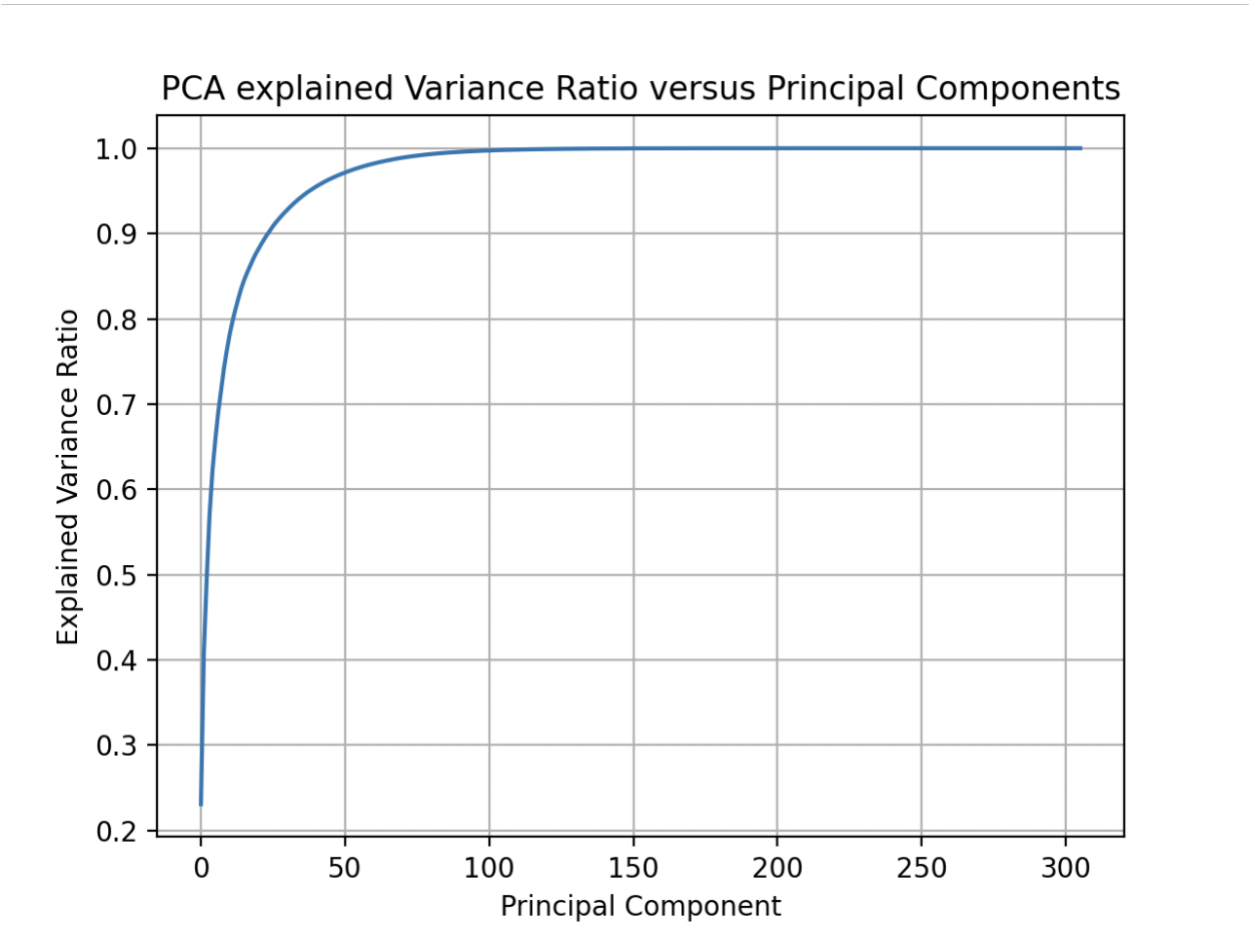

Figure 1: *Plot displaying PCA explained variance ratio versus principal components, with normalized values.*

It is evident that the first two principal components of the dataset capture approximately 23.1% and 17.6% of the data, respectively. The figure given below displays the ratio of variance explained by the first two principal components:

```
Explained Variance Ratio for PC1 and PC2: 0.231 0.176
```

Figure 2: *Figure displaying the ratio of variance explained by the first two principal components.*

Continuing, we project the data into the first two principal components, which is visualized in the figure given below:

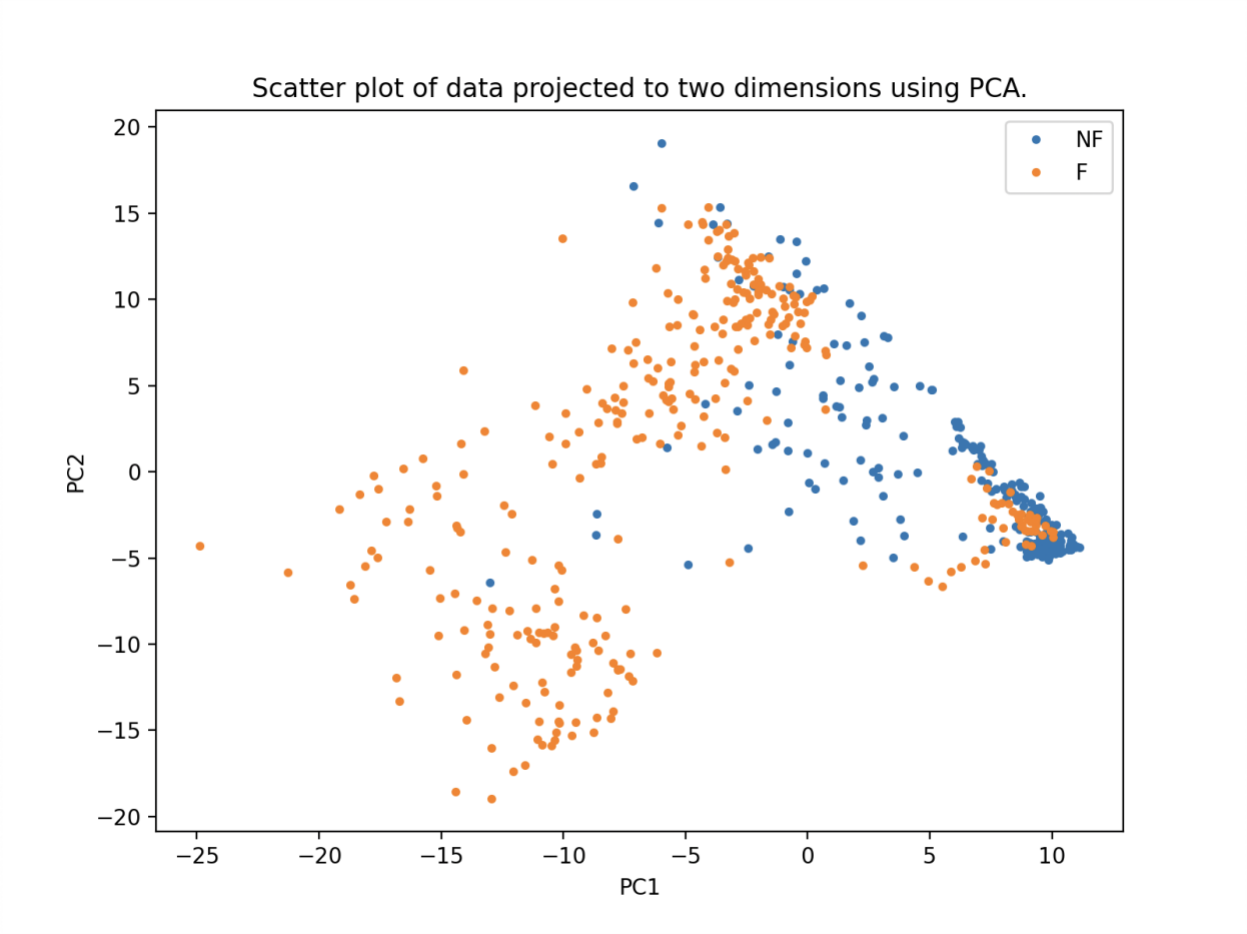

Figure 3: *Scatter plot of data projected to two dimensions using Principal Component Analysis.*

It is evident that there occurs overlapping in our data. Continuing, we apply K-Means clustering to our dataset, where the changind variable is the number of clusters. The figures given below display the clustering results for varying number of clusters:

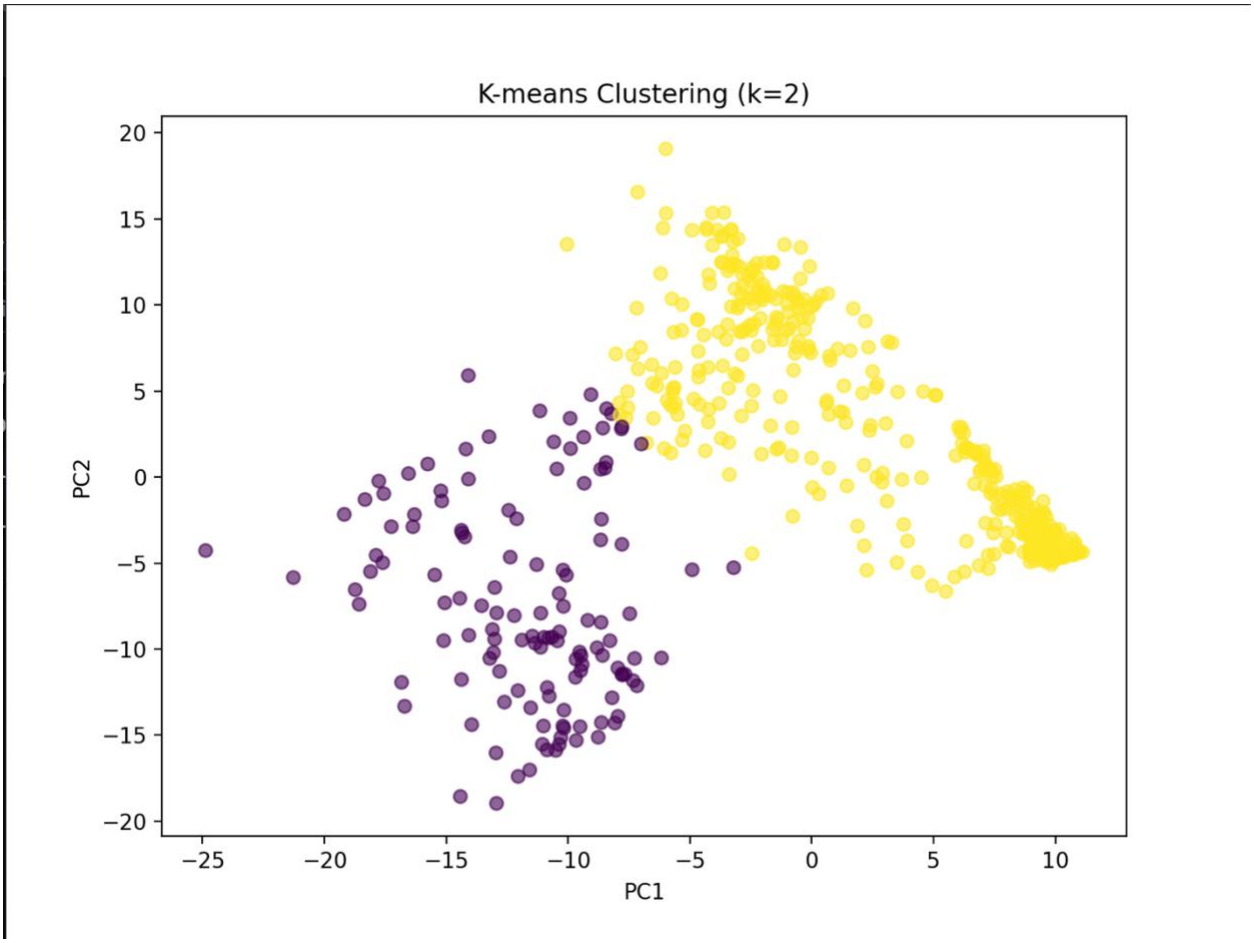

Figure 4: *K-means clustering with two clusters.*

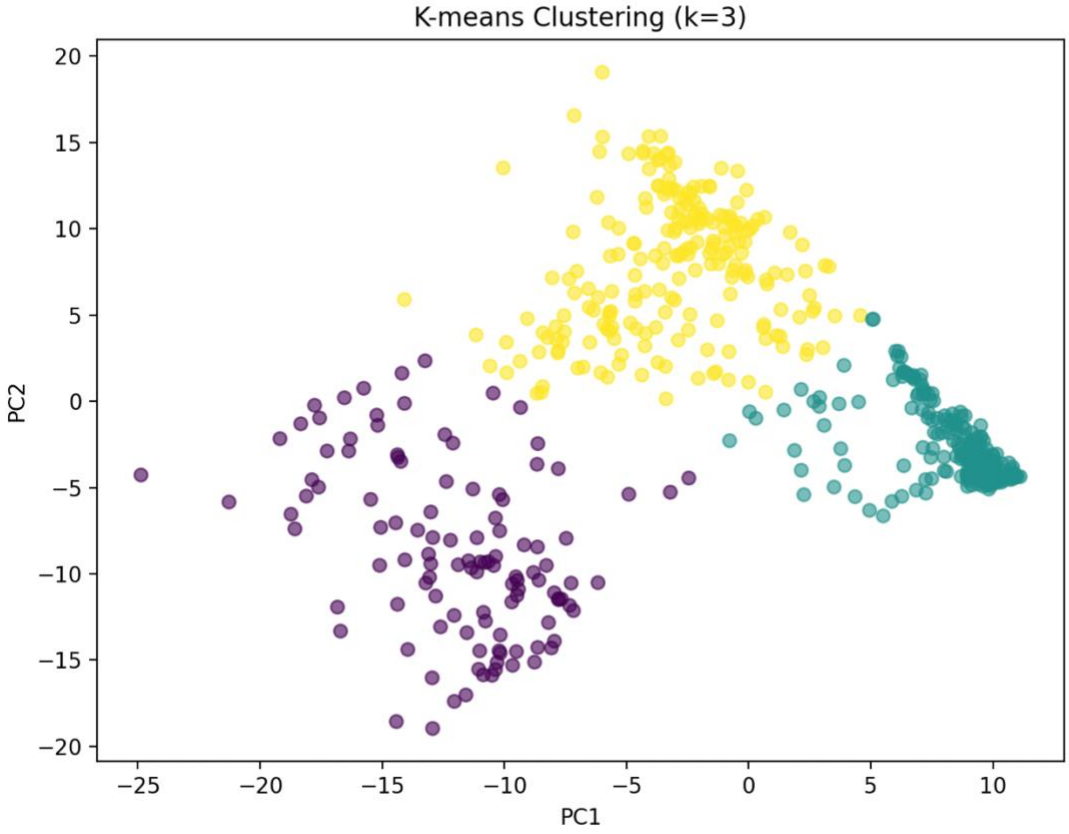

Figure 5: *K-means clustering with three clusters.*

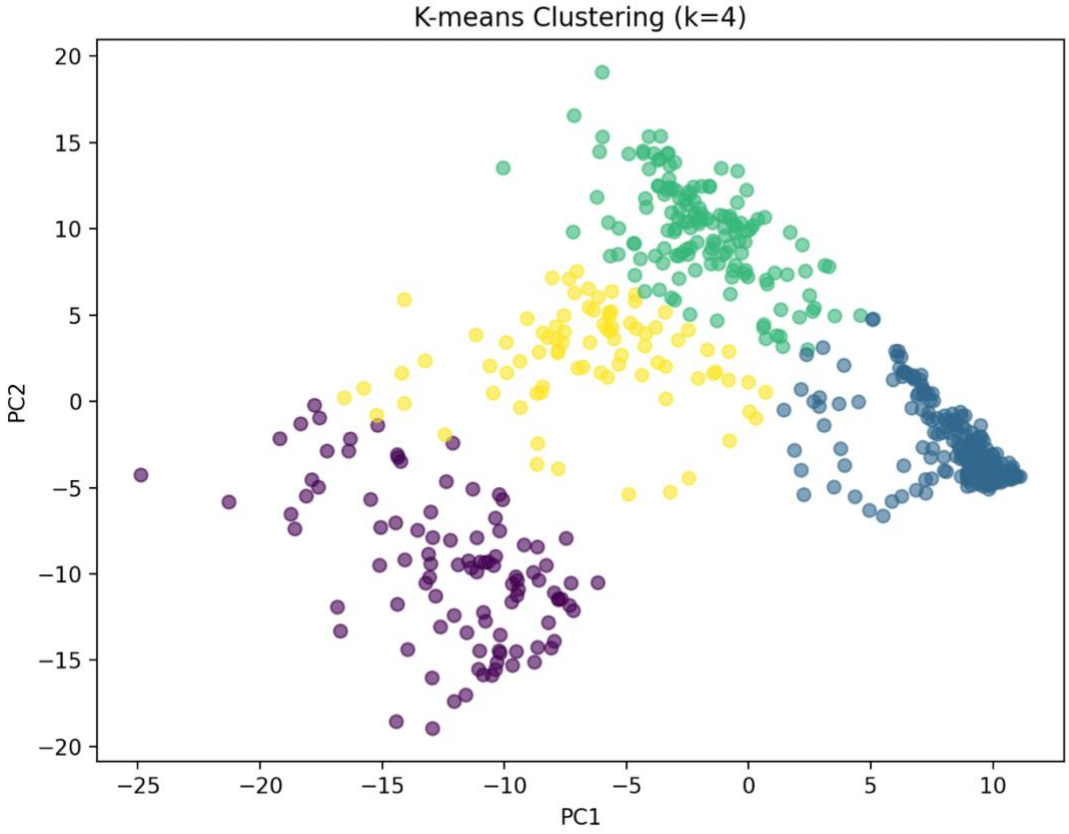

Figure 6: *K-means clustering with four clusters.*

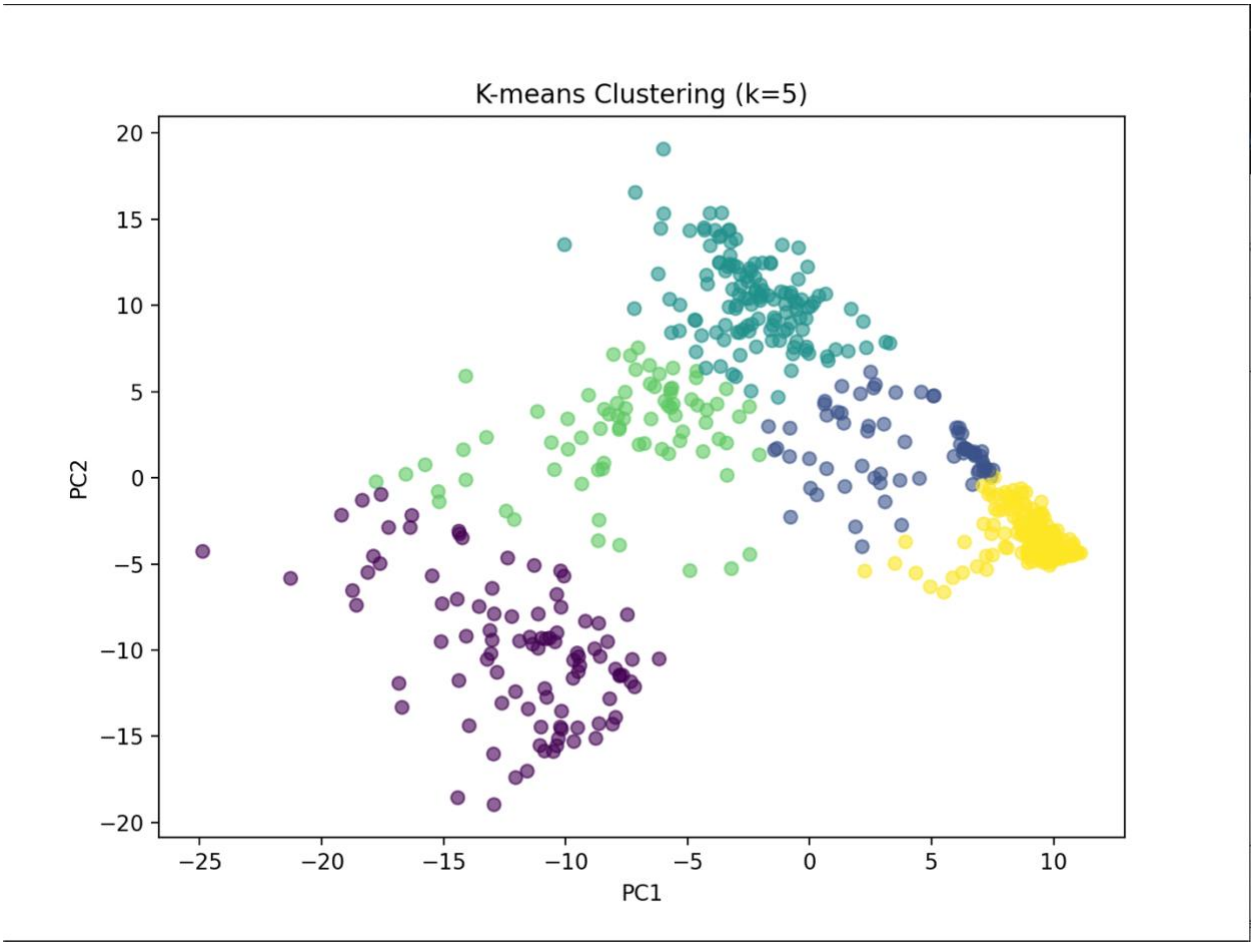

Figure 7: *K-means clustering with five clusters.*

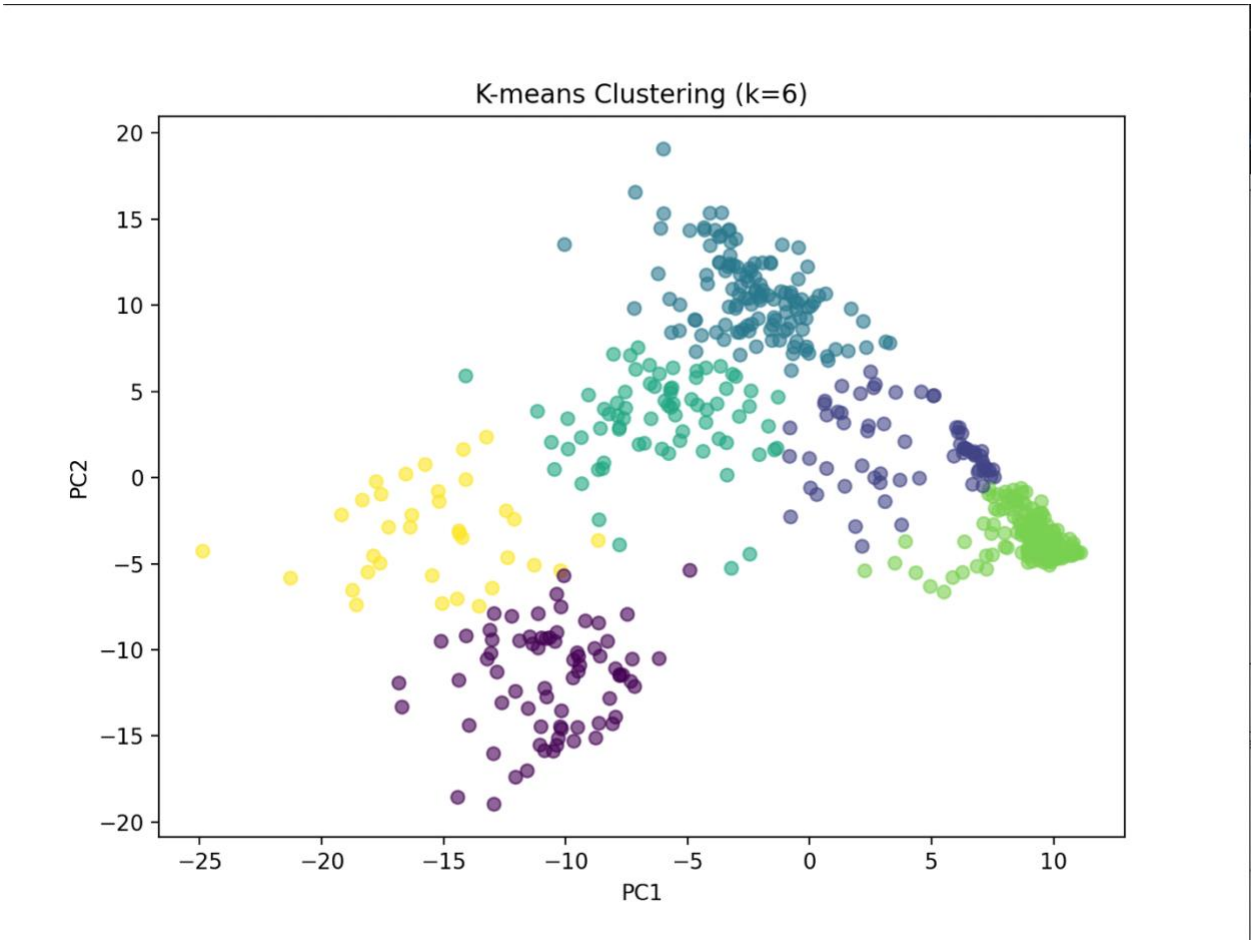

Figure 8: *K-means clustering with six clusters.*

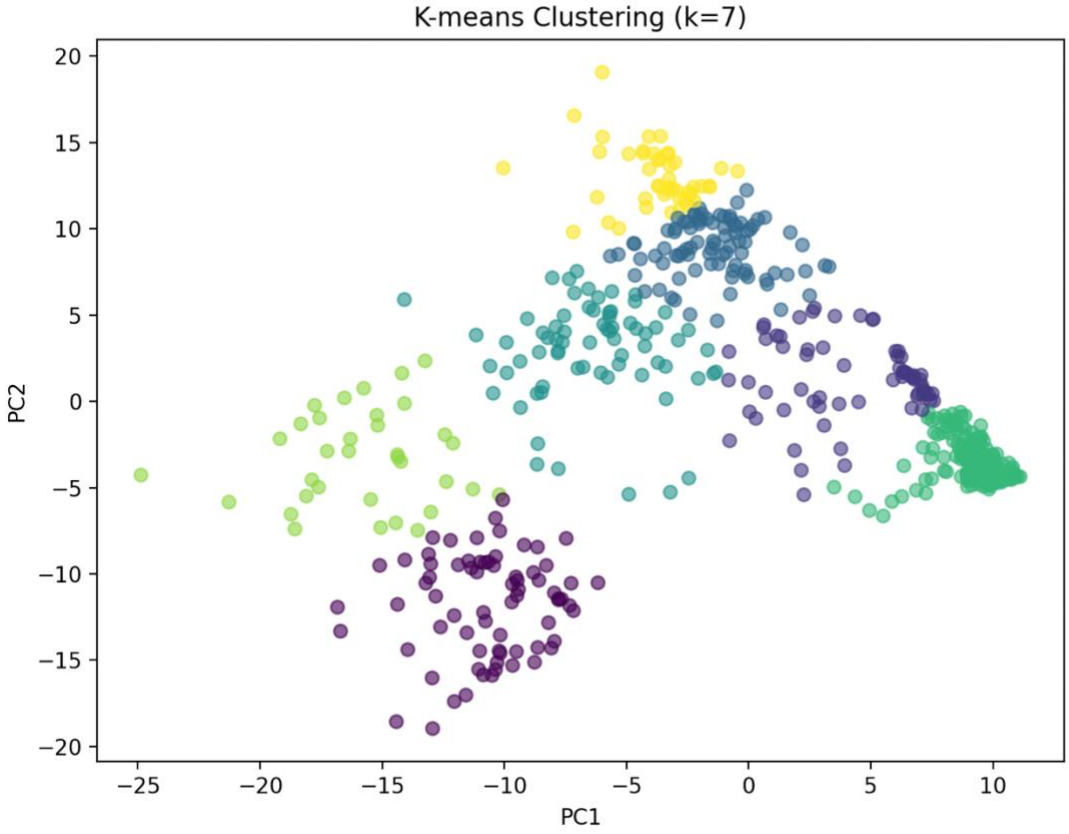

Figure 9: *K-means clustering with seven clusters.*

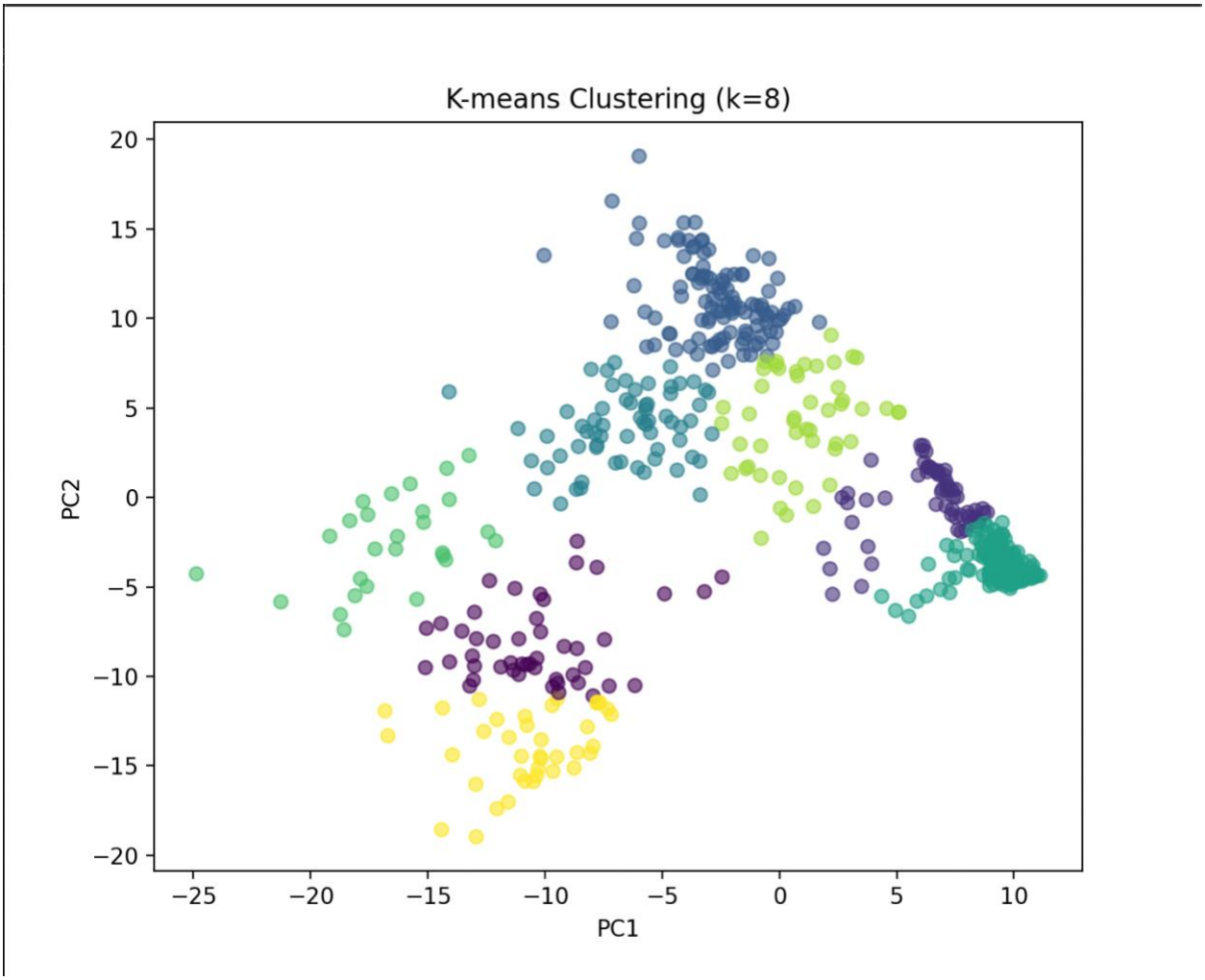

Figure 10: *K-means clustering with eight clusters.*

The figures provided above display that while fall detection is possible, there exists no apparent decision boundary for various number of clusters, in other terms, as we mentioned before, there exists overlapping results and labels.

Results for Part B

In Part B, we use two supervised learning methods, Support Vector Machines and Multilayer Perceptron Algorithm, respectively, in order to classify our dataset. We start by splitting our dataset into training, test, and validation with ratios $75\%, 15\%$ and 15%, respectively. Next, we apply our Support Vector Machine classifier. We test different combinations of hyperparameters by varying the regularization parameter and trying different kernels, such as linear, radial basis function, and polynomial. For each combination, we train an SVM on the training set and measure

its accuracy on the validation set. The model that achieves the highest validation accuracy is saved as the best SVM model. Finally, we evaluate this best model on the test set and display its performance metrics. The figure given below displays the parameter combinations and tuning process:

```
SVM Hyperparameter Tuning:
SVM (kernel=linear, C=0.1) - Validation Accuracy: 1.0000
SVM (kernel=rbf, C=0.1) - Validation Accuracy: 0.9333
SVM (kernel=poly, C=0.1) - Validation Accuracy: 0.8500
SVM (kernel=linear, C=0.5) - Validation Accuracy: 1.0000
SVM (kernel=rbf, C=0.5) - Validation Accuracy: 1.0000
SVM (kernel=poly, C=0.5) - Validation Accuracy: 0.9333
SVM (kernel=linear, C=1) - Validation Accuracy: 1.0000
SVM (kernel=rbf, C=1) - Validation Accuracy: 1.0000
SVM (kernel=poly, C=1) - Validation Accuracy: 0.9667
SVM (kernel=linear, C=2) - Validation Accuracy: 1.0000
SVM (kernel=rbf, C=2) - Validation Accuracy: 1.0000
SVM (kernel=poly, C=2) - Validation Accuracy: 0.9833
SVM (kernel=linear, C=5) - Validation Accuracy: 1.0000
SVM (kernel=rbf, C=5) - Validation Accuracy: 1.0000
SVM (kernel=poly, C=5) - Validation Accuracy: 0.9833
SVM (kernel=linear, C=10) - Validation Accuracy: 1.0000
SVM (kernel=rbf, C=10) - Validation Accuracy: 1.0000
SVM (kernel=poly, C=10) - Validation Accuracy: 0.9833
```

Figure 11: *SVM hyperparameter combinations and their respective validation accuracies.*

The figure given below displays the best model parameters and its accuracy:

```
Best SVM classifier test accuracy: 0.9824
SVM Classification Report:
              precision    recall  f1-score   support

          NF       1.00      0.96      0.98        76
           F       0.97      1.00      0.98        94

    accuracy                           0.98       170
   macro avg       0.98      0.98      0.98       170
weighted avg       0.98      0.98      0.98       170


Best SVM parameters:
{'C': 0.1, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_ite
r': -1, 'probability': False, 'random_state': 22001758, 'shrinking': True, 'tol': 0.001, 'verbose': False}
```

Figure 12: *SVM classification report with best model parameters.*

After applying the Support Vector Machine classifier, we use a Multilayer Perceptron classifier. In a similar manner, we experiment with various network configurations by testing different sizes for the hidden layers and different values of regularization parameters. The Multilayer Perceptron model is trained on the training set and then validated on the validation set to select the best configuration. The chosen model is then evaluated on the test set. The figures given below display the parameter optimization process:

```
MLP Hyperparameter Tuning:
MLP (hidden_layer_sizes=(50,), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50,), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50,), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50,), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50,), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50,), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50,), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50,), alpha=0.5) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75,), alpha=0.5) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100,), alpha=0.5) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125,), alpha=0.5) - Validation Accuracy: 1.0000
```

Figure 13: *Parameter optimization process for single layer.*

```
MLP (hidden_layer_sizes=(50, 50), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50, 50), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50, 50), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50, 50), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50, 50), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50, 50), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50, 50), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(50, 50), alpha=0.5) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(75, 75), alpha=0.5) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(100, 100), alpha=0.5) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.0001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.0005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.001) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.005) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.01) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.05) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.1) - Validation Accuracy: 1.0000
MLP (hidden_layer_sizes=(125, 125), alpha=0.5) - Validation Accuracy: 1.0000
```

Figure 14: *Parameter optimization process for two layers.*

```
Best MLP classifier test accuracy: 0.9882
MLP Classification Report:
              precision    recall  f1-score   support

          NF       1.00      0.97      0.99        76
           F       0.98      1.00      0.99        94

    accuracy                           0.99       170
   macro avg       0.99      0.99      0.99       170
weighted avg       0.99      0.99      0.99       170


Best MLP parameters:
{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'early_stopping': False, 'epsilon': 1e-08, 'hidden_layer_sizes': (50,), 'learning_rate
': 'constant', 'learning_rate_init': 0.001, 'max_fun': 15000, 'max_iter': 1000, 'momentum': 0.9, 'n_iter_no_change': 10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state':
 22001758, 'shuffle': True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': False, 'warm_start': False}
```

Figure 15: *Best MLP classifier results and parameters.*

It is evident that by using supervised learning methods such as Support Vector Machines and Multilayer Perceptrons, we can successfully classify and determine whether a person has fell or not using telehealth data.

Conclusion

In conclusion, we analyzed wearable sensor data to detect falls using both unsupervised and supervised methods. In the first part, we reduced the high-dimensional data using Principal Component Analysis (PCA) and then applied k-means clustering to examine how the data grouped into clusters. We compared these clusters with the known fall and non-fall labels to understand the natural separation in the data. In the second part, we divided the data into training, validation, and testing sets, and built two classifiers using Support Vector Machines (SVM) and Multi-Layer Perceptrons (MLP). We tuned the models' parameters and evaluated their performance with accuracy and other metrics. These steps show that the combination of unsupervised and supervised methods is effective for fall detection.

References

[1] "Principal component analysis(pca)," GeeksforGeeks, https://www.geeksforgeeks.org/principal-component-analysis-pca/ (accessed Mar. 26, 2025).

[2] "K means clustering - introduction," GeeksforGeeks, https://www.geeksforgeeks.org/k-means-clustering-introduction/ (accessed Mar. 26, 2025).

[3] "Support Vector Machine (SVM) algorithm," GeeksforGeeks, https://www.geeksforgeeks.org/support-vector-machine-algorithm/ (accessed Mar. 26, 2025).

[4] GeeksforGeeks, "Multi-layer perceptron learning in tensorflow," GeeksforGeeks, https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/ (accessed Mar. 26, 2025).

Appendices

Appendix A: Python Script for Part A

```
# Import the required libraries.
# Import the required libraries.
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset.
data = pd.read_csv('falldetection_dataset.csv', header=None)
data = data.drop(0, axis=1)
data = data.replace('F', 1)
data = data.replace('NF', 0)
labels = data[1]
cl = ['NF', 'F']
data = data.drop(1, axis=1)
print(data.head())

# Standardize the features of the dataset.
scaler = StandardScaler()
scaledFeatures = scaler.fit_transform(data)

# Apply PCA and plot the explained variance ratio versus principal components.
pca_full = PCA()
PCAresult_full = pca_full.fit_transform(scaledFeatures)
plt.plot(range(1, len(pca_full.explained_variance_ratio_)+1),
        np.cumsum(pca_full.explained_variance_ratio_))
plt.xlabel('Principal Component')
plt.ylabel('Cumulative Explained Variance Ratio')
```

```
plt.title('PCA Explained Variance Ratio versus Principal Components')
plt.grid()
plt.show()

# Apply PCA again with only two components.
pca = PCA(n_components=2)
PCAresult = pca.fit_transform(scaledFeatures)
print("Explained Variance Ratio for PC1 and PC2:", round(pca.explained_variance_ratio_[0],3),
round(pca.explained_variance_ratio_[1],3))

# Plot the projection of the first two components classified with their labels.
plt.figure(figsize=(8,6))
for i in range(2):
    ind = np.where(labels == i)[0]
    plt.plot(PCAresult[ind, 0], PCAresult[ind, 1], 'o', label=cl[i], markersize=3)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Scatter plot of data projected to two dimensions using PCA.")
plt.legend()
plt.show()

# Apply K-means clustering on the first two components.
for k in range(2, 9):
    kmeans = KMeans(n_clusters=k, random_state=22001758)
    clusters = kmeans.fit_predict(PCAresult)
    plt.figure(figsize=(8,6))
    plt.scatter(PCAresult[:, 0], PCAresult[:, 1], c=clusters, cmap='viridis', alpha=0.6)
    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.title(f'K-means Clustering (k={k})')
    plt.show()

# Run k-means clustering with k=2.
kmeans_2 = KMeans(n_clusters=2, random_state=22001758)
clusters_2 = kmeans_2.fit_predict(PCAresult)
comparison_df = pd.DataFrame({'Cluster': clusters_2, 'Action Label': labels})
print(comparison_df.groupby(['Cluster', 'Action Label']).size())
```

Appendix B: Python Script for Part B

```
encodedLabels = labels.values

# Split the data into training (70%), validation (15%), and testing (15%) sets.
X_train_val, X_test, y_train_val, y_test = train_test_split(
    scaledFeatures, encodedLabels, test_size=0.30, random_state=22001758, stratify=encodedLabels
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.15, random_state=22001758, stratify=y_train_val
)
print("\nTraining set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", X_test.shape)

# Apply the support vector machine classifier.
bestValAccuracy = 0
bestSVM = None
print("\nSVM Hyperparameter Tuning:")
for C in [0.1, 0.5, 1, 2, 5, 10]:
    for kernel in ['linear', 'rbf', 'poly']:
        svm_model = SVC(C=C, kernel=kernel, random_state=22001758)
```

```
        svm_model.fit(X_train, y_train)
        val_preds = svm_model.predict(X_val)
        val_acc = accuracy_score(y_val, val_preds)
        print(f"SVM (kernel={kernel}, C={C}) - Validation Accuracy: {val_acc:.4f}")
        if val_acc > bestValAccuracy:
            bestValAccuracy = val_acc
            bestSVM = svm_model

# Evaluate the best SVM on the test set.
SVMTestPredictions = bestSVM.predict(X_test)
SVMTestAccuracy = accuracy_score(y_test, SVMTestPredictions)
print(f"\nBest SVM classifier test accuracy: {SVMTestAccuracy:.4f}")
print("SVM Classification Report:")
print(classification_report(y_test, SVMTestPredictions, target_names=cl))
print("\nBest SVM parameters:")
print(bestSVM.get_params())

# Run the Multilayer Perceptron training algorithm.
bestValAccuracyMLP = 0
bestMLP = None
print("\nMLP Hyperparameter Tuning:")
for hidden_layer_sizes in [(50,), (75,), (100,), (125,), (50,50), (75,75), (100,100), (125,125)]:
    for alpha in [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]:
        mlp_model = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes, alpha=alpha,
                                  max_iter= 1000, random_state=22001758)
        mlp_model.fit(X_train, y_train)
        val_preds = mlp_model.predict(X_val)
        val_acc = accuracy_score(y_val, val_preds)
        print(f"MLP (hidden_layer_sizes={hidden_layer_sizes}, alpha={alpha}) - Validation Accuracy: {val_acc:.4f}")
        if val_acc > bestValAccuracyMLP:
            bestValAccuracyMLP = val_acc
            bestMLP = mlp_model

# Evaluate the best MLP on the test set.
MLPTestPredictions = bestMLP.predict(X_test)
MLPTestAccuracy = accuracy_score(y_test, MLPTestPredictions)
print(f"\nBest MLP classifier test accuracy: {MLPTestAccuracy:.4f}")
```

```
print("MLP Classification Report:")
print(classification_report(y_test, MLPTestPredictions, target_names=cl))
print("\nBest MLP parameters:")
print(bestMLP.get_params())
```