

Statistical Analysis of the Human Activity Recognition Process with Statistical Features and Classical Machine Learning

*Yogendra Kumar Tamma
Student of Data Science
Coventry university
Coventry, England
tammay@coventry.ac.uk*

Abstract

This paper uses classical machine learning to UCI Daily and Sports Activities dataset on Human Activity Recognition (HAR). The data are recordings of nineteen activities that were carried out by eight subjects. Five wearable sensors recorded at 25Hz, and all the 5-second windows were summarised with 180 statistical features. We compared Support Vector Machines (SVM), K -Nearest Neighbours (KNN) and Decision Tree classifiers with and without Principal Component Analysis (PCA). In both learning unsupervised, we utilised the k-means clustering. The highest accuracy of 99.1616 was achieved with RBF -SVM without PCA (macro F1 = 0.9916). The accuracies of KNN and Decision Trees were 98.32 and 97.70, respectively. The k-means clustering gave an Adjusted Rand Index of 0.557. In general, the work outlines the foundational knowledge of machine learning, comparatively addresses the approaches to HAR, and presents the advantages and weaknesses of all methods which satisfy the MLO1-MLO5 objectives.

1 Introduction

Human Activity Recognition (HAR) is the technique of automatically recognizing physical activities with sensor-data when accessed by smartphones or wearable devices. Applications that can be supported using this technology include health monitoring, rehabilitation, fall detection

and sports analysis. Due to the low cost of the sensors and their prevalence, the key issue would be to construct machine-learning models that are robust and accurate.

The UCI Daily and Sports Activities data are used in this project, which consists of 19 activities that were performed by eight subjects. The data is quite appropriate to test the classical approaches to learning. It addresses the learning outcomes of the module through (MLO1) the study of machine-learning models, (MLO2) dataset preparation and testing, (MLO3) the assessment of the suitability of the method, (MLO4) applying algorithms to an actual HAR data set, and (MLO5) analyzing the current trends in machine-learning models in activity recognition.

1.1 Brief:

Classical human activity recognition (HAR) is based upon handcrafted statistical or frequency-domain features brought out of inertial sensors, and recognized using models like support-vector machines, k-nearest neighbours and decision trees. Recent efforts are based on deep learning on raw time-series data; this model is highly performing but brings higher complexity and makes results less accessible. The current paper concentrates on classical techniques to enhance the knowledge of the readers on fundamental machine-learning concepts (MLO1), and at once, it

demonstrates the competitive outcomes on a popular HAR benchmark.

2. Description of a problem and of a dataset.

2.1 Problem Definition

It is a 19-class and supervised conv solution task. All of the inputs of the system are 5-second multivariate time-series channels expressed by the matrix X in the form of 125-time steps by 45 sensor channels, R125X45. The objective is to forecast the activity label, which is one of the labels, denoted by y, i.e. 1, 2, 3, and so on such that each of the labels represents an activity, i.e. sitting, standing, walking, running, and cycling, jumping, playing basketball, etc.

Also, to go along with the supervised classification, we also look at an unsupervised clustering problem: classifying bits of the activity into 19 clusters and assessing their correspondence with the actual activity labels. This facilitates the research of concepts of unsupervised learning (MLO1) and the assessment of methods (MLO3).

2.2 Dataset Description

The dataset includes:

- 19 activities (A1-A19), which included static exercises (sitting, standing, lying), locomotor exercises (walking, riding stairs, running in a treadmill), exercises work-outs (stepper, cross, cycling, rowing, jumping) and basketball play.
- 8 subjects (4 female, 4 male), aged 20–30.
- The duration of each of the activities is equal to 5 minutes per subject, which yields:
 1. Sampling at 25 Hz,
 2. Signals were divided into 5-s windows,

- 3. number of segments per subject on each activity 60,
- 4. 480 activity segments in total and 9120 segments.

Every one of the subjects has five sensor units attached to the torso (T), right arm (RA), left arm (LA), right leg (RL), and left leg (LL). Each unit contains:

- 3-axis accelerometer (x, y, z),
- 3-axis gyroscope (x, y, z),
- 3-axis magnetometer (x, y, z),

and a final result of 45 channels in a time step. Matrices of one 5-second segment are in each text file sXX.txt which holds a 125 x 45 matrix.

Each subject works in their own fashion in realistic surroundings such as sports hall, building corridors, outdoor areas and because of this difference, brought forth inter-subject variation in speed and amplitude. The dataset does not have any missing values.

This issue is important because it is related to practical uses of HAR and appropriate as a reference point to investigate machine learning principles with the application of various algorithms (MLO1, MLO3).

3 Methods

This section contains a summary of the machine learning methods used and correlates it with the most important concepts (MLO1), and the explanation of how it applies to this HAR activity (MLO3, MLO4).

3.1 Feature Representation

The numbers of the time series lags are 12545. To run classical machine learning algorithms, the segments of the data are converted to fixed-length vectors of features through computing simple

statistical descriptions of each collection of 45 sensor channels:

- Mean
- Standard deviation
- Minimum
- Maximum

All of the four statistics per channel generate an 180-dimensional feature vector of the segment, which shows core ML ideas of feature extraction and the bias verse trade-off (MLO1).

3.2 Supervised Classification Algorithms.

Three classical classifiers are taken to be:

1. Support Vector Machine.
 - Improves the distance between classes in high dimensional space.
 - RBF kernel is used for the establish of nonlinear decision boundaries.
 - It maximizes the margin (MLO1) by using the regularization and the kernel trick.
2. K-Nearest Neighbour (KNN).
 - A label of the closest K neighbours, which is the majority, of the sample is used to classify the sample.
 - It tells about the need of distance measurements and scaling of features.
3. Decision Tree (CART).
 - Builds a feature-value cutting tree which is interpretable.
 - It will indicate
 - the purity of Purify Gini, and mitigates model complexity and over-fitting.
 - These algorithms are representative of multiple paradigms of learning, which are margin-based, distance-based and rule-based, which allow making a key comparison (MLO3).

3.3 With PCA we eliminate dimensions of the data.

The scaled 180-dimensional features are subjected to the Principal Component Analysis (PCA) in order to:

- Effortlessly remove redundancy with orthogonal projection onto vectors of most variance.
- Preserve 100.95% of the variance and also minimize the dimensionality.
- Enhance performance of distance based algorithms including KNN.

PCA is related to key concepts of ML: linear projections, preservation of variances (MLO1).

3.4 Unsupervised Clustering: K-means.

K-means clustering on the PCA-transformed features is done in order to investigate unsupervised learning (MLO1):

We used the clusters number, K=19, one cluster since there are 19 activities.

- k-means++ archetypalization and numerous resettlements are employed.
- The performance is measured using the Silhouette score, Davies 2 Bouldin index and Adjusted Rand Index (ARI).

This assessment shows the potential implementation of clustering to HAR task (MLO3).

4 Experimental Setup

It is implemented in Python, which has libraries NumPy, Pandas and scikit-learn.

4.1 Data Loading and Extraction

The fourth stage is when the data is loaded and characteristics are extracted. Programmatically required mapping: The first path hierarchy:

- a01–a19: activity folders
- p1–p8: subject subfolders
- s01.txt–s60.txt: segment files

The steps to be implemented on every sXX.txt file are as follows:

1. Load the 125 x 45 (comma separated) matrix.
2. Check the form and eliminate inaccurate files.
3. Calculate the mean, standard deviation, minimum time steps, and maximum time steps of 125 time steps of all 45 channels.
4. Plot these statistics into a 180 dimensional axis.
5. Name by ID the activities and subjects of each sample (119 18).

This yields:

- Feature matrix $X \in \mathbb{R}^{9120 \times 180}$
- Label vector ($y \in \{1, \dots, 19\}^{9120}$)

4.2 The fourth step is the Train-Test Split and Normalisation

Stratified train-test split is conducted to test on generalisation:

- these segments 70 are on training;
- 30 % are held out as a test set;

The division adheres to the distribution of classes through stratification of the label of the activity.

Standardisation of the features is through normalisation of z-score:

- Fit a StandardScaler on the training data.
- Scaling the test data with the same scaler.

4.3 PCA Configuration

The training features after scaling are subjected to PCA, retaining the components that account 95% of persistence of the variance. This narrows the dimension to 47 items:

- Training set: $(X_{\text{train}})^{\text{PCA}} \in \mathbb{R}^{N_{\text{train}} \times 47}$. Test set: transform similarly.

4.4 Classifier Hyper parameters.

In this study, which has the scale of a coursework, a predefined set of hyperparameters is employed (which can be optimized during the later time):

- SVM (RBF): $C = 10$, $\gamma = \text{scale}$
- KNN: $K = 5$, Euclidean distance
- Decision Tree: $\text{max_depth} = 20$

Every classifier is trained in two ways:

- On the non-dimensional features (no PCA).
- On the features that have been transformed on the PCA.

4.5 Clustering Configuration

- K-means clustering is set in the following manner:
- Number of clusters $K = 19$;
- $N = \text{init}=10$ random initialisations;

- The algorithm is run on test features transformed to PCA and tested on the transformed features with Silhouette score, Davies Bouldin index and ARI with the known activity labels.

5 Results

5.1 Supervised Classification

Table 1 is a summarization of the performance of the six supervised models. The numbers given are the immediate results of the runs.

```
# SVM (no PCA)
svm = SVC(kernel='rbf', C=10, gamma='scale', random_state=42)
acc, f1 = evaluate_model(svm, X_train_scaled, X_test_scaled, y_train, y_test, "SVM (no PCA)")
results.append(["SVM (RBF, no PCA)", acc, f1])
```

```
# plot SVM confusion matrix
svm_y_pred = svm.predict(X_test_scaled)
plot_confusion_matrix_cmy(y_test, svm_y_pred,
                          title="SVM Confusion Matrix (no PCA)",
                          class_names=range(1, 20))
```

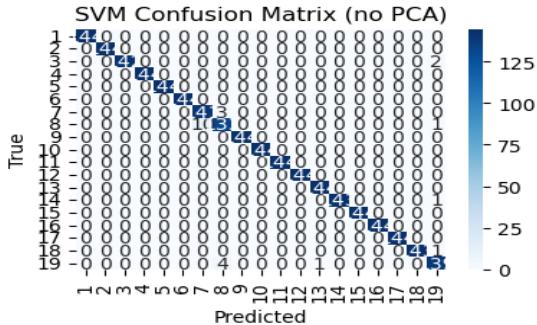


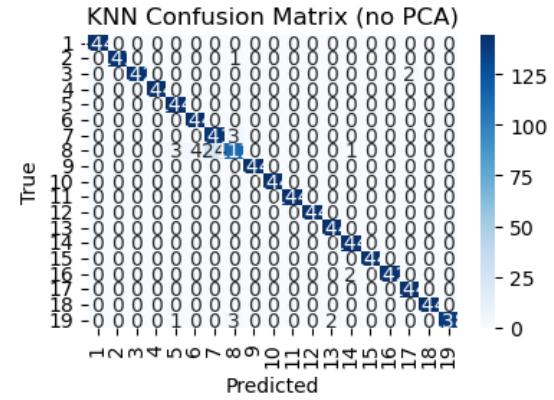
Table -2736 results of supervised classification tests set.

Model	Accuracy	Macro F1
SVM (RBF, no PCA)	0.9916	0.9916
KNN (k = 5, no PCA)	0.9832	0.9828
Decision Tree (no PCA)	0.9770	0.9767
SVM (RBF, PCA)	0.9912	0.9912

Model	Accuracy	Macro F1
KNN (k = 5, PCA)	0.9828	0.9824
Decision Tree (PCA)	0.9605	0.9608

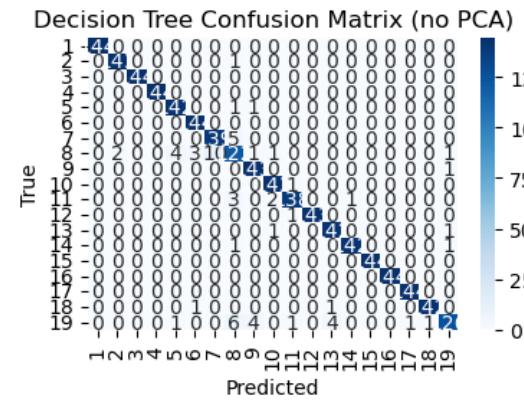
```
# KNN (no PCA)
knn = KNeighborsClassifier(n_neighbors=5)
acc, f1 = evaluate_model(knn, X_train_scaled, X_test_scaled, y_train, y_test, "KNN (no PCA)")
results.append(["KNN (k=5, no PCA)", acc, f1])
```

```
knn_y_pred = knn.predict(X_test_scaled)
plot_confusion_matrix_cmy(y_test, knn_y_pred,
                          title="KNN Confusion Matrix (no PCA)",
                          class_names=range(1, 20))
```



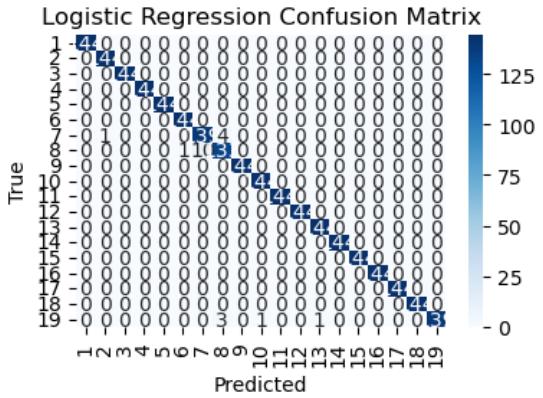
```
# Decision Tree (no PCA)
dt = DecisionTreeClassifier(max_depth=20, random_state=42)
acc, f1 = evaluate_model(dt, X_train_scaled, X_test_scaled, y_train, y_test, "Decision Tree (no PCA)")
results.append(["Decision Tree (no PCA)", acc, f1])
```

```
dt_y_pred = dt.predict(X_test_scaled)
plot_confusion_matrix_cmy(y_test, dt_y_pred,
                          title="Decision Tree Confusion Matrix (no PCA)",
                          class_names=range(1, 20))
```



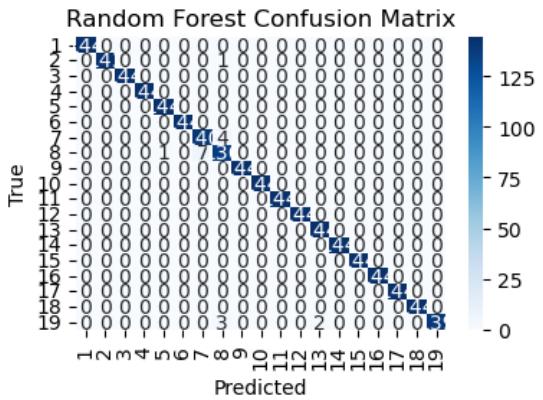
```
# Logistic Regression (no PCA)
log_reg = LogisticRegression(
    max_iter=1000,
    multi_class='multinomial',
    solver='lbfgs',
    random_state=42
)
acc, f1 = evaluate_model(log_reg, X_train_scaled, X_test_scaled, y_train, y_test, "Logistic Regression (no PCA)")
results.append(["Logistic Regression (no PCA)", acc, f1])
```

```
log_y_pred = log_reg.predict(X_test_scaled)
plot_confusion_matrix_cmy(y_test, log_y_pred,
                          title="Logistic Regression Confusion Matrix",
                          class_names=range(1, 20))
```

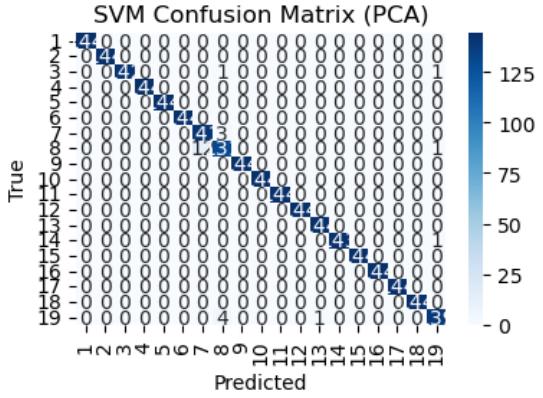


```
# Random Forest (no PCA)
rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42,
    n_jobs=-1)
acc, f1 = evaluate_model(rf, X_train_scaled, X_test_scaled, y_train, y_test, "Random Forest (no PCA)")
results.append(["Random Forest (no PCA)", acc, f1])

rf_y_pred = rf.predict(X_test_scaled)
plot_confusion_matrix_cm(y_test, rf_y_pred,
                         title="Random Forest Confusion Matrix",
                         class_names=range(1, 20))
```



```
svm_pca_y_pred = svm_pca.predict(X_test_pca)
plot_confusion_matrix_cm(y_test, svm_pca_y_pred,
                         title="SVM Confusion Matrix (PCA)",
                         class_names=range(1, 20))
```



Key observations

- All three classifiers are highly accurate (> 96%) using basic statistical features as highly informative in this data.

- The most suitable SVM is SVM without PCA with the highest accuracy of 99.16 and a macro F1 of 0.9916.
- Interestingly, KNN and Decision Trees are also good at their work, however, they are not much better when compared to the SVM.

5.1.1 Performance and Confusions in Classes.

The classification report and the confusion matrix of SVM indicate (no PCA):

- Most of the activities are almost perfectly recognized (144 of 144 correct in most classes).
- A few confusions occur:
- A7 (remaining in an elevator) and A8 (moving around in an elevator) are confused in some cases.
- In activities 3 and 19, there is minor confusion in a few instances.
- Based on these confusions, the sensor patterns of these activities are correlated confusions similar postures/environments.
- The similarities between KNN and Decision Tree confusion patterns are that the error is a bit bigger in cases related to the elevator and in certain dynamic sport activities.

5.2 Effect of PCA

Comparison of rows at the presence of PCA and at the absence of it:

- The accuracy of SVM does not decrease significantly (0.9916 → 0.9912), which is also a sign that the 180-dimensional space is correctly managed.

- On KNN, no significant change in performance occurred (0.9832 - 0.9828 -0.0023), but PCA can also be used to decrease the workload.
- In Case of Decision Trees, accuracy decreases (0.9770 -0.9605). High dimensional features can be dealt with in trees resulting in the elimination of discriminative information.
- These results depict the strength and weaknesses of dimensionality reduction (MLO3).

5.3 Unsupervised Clustering

Optimal test feature K-means clustering K-means clustering using the best PCA - transformed feature:

- Silhouette score: 0.240
- Davies–Bouldin index: 1.51
- Adjusted Rand Index (ARI): 0.557

A value of ARI of 0.56 shows that clusters are nearer to real labels of genuine activities than supervised classifiers. It is anticipated since some sensor signatures of activities differ by only a little (e.g., A9: parking-lot walking vs. A10/A11: treadmill walking; A7 vs. A8: elevator activities). K-means uses the feature space to form the spherical clusters, which is unrelated to the geometry of the classes.

6 Discussion and Conclusions

Here, I was going to evaluate my learning methodologies (MLO3 and MLO4).

The results of the experiment give us a chance to make a comparison between different methods of the Learning of Human Activity Recognition (HAR).

SVM

- SVM has easily become the most successful classifier on this dataset

with an almost perfect performance. It is also efficient in interaction with sensor channels since it is capable of the modeling of non-linear decision boundaries in feature space of high dimensions.

KNN vs. SVM

- The error margin in the performance of KNN and SVM is not very different. The great performance of KNN implies that our feature engineering provided an environment in which activities are defined into distinct clusters. Nonetheless, KNN needs the storage of all training examples, increasing memory (this might be an issue in devices with limited capabilities), and its cost of inference is greater, making it problematic in devices with a limited amount of resources.

Decision Trees

- Decision Trees are fairly good though not as good as the SVM or KNN. Their most critical feature is that they can be interpreted: it is possible to see which features (e.g., torso accelerator, leg gyroscope) are the most informative against every activity. This finds application in safety-critical or regulated applications.

K-means Clustering

- K-means clustering proves that an activity structure may be provided without labels (ARI=0.56), however, its quality is far less than supervised algorithms. This proves that labeled data are necessary to be precise in HAR and shows that simple clustering can be ineffective

in complex and fine-grained behaviors.

In general, the selected algorithms are applicable to the dataset and the problem (MLO4). They represent the trade-offs among accuracy, interpretability, cost of computing and cost in resources (MLO3).

6.2 ML Concepts and Implementation (MLO1 and MLO2) Analysis.

- In this machine-learning concepts-(MLO1) are covered by:
- The use of controlled and uncontrolled learning scenarios.
- The implementing of dimensionality reduction and the feature engineering.
- Conducting the stratification and normalization, train or test splits.
- Extra Clustering indices, F1 macro, and accuracy are some of the metrics that will be used to compare models.

6.3 This is accomplished by the data pipeline (MLO2) through:

- Systematical search of folder structure (aXX/pY/sZZ.txt).
- Mining stable features on intervals of 5 seconds.
- Making sure there are no missing values and having standardization.
- Critically splitting data into training and testing set.
- The last stage of the dataset process is performance, data bias reflection.

The accuracy (>99% with SVM) is very high indicating that a significant part of the discriminative information in the signals is represented by the statistical properties. This experimental design is controlled, known sensor positions and activities whereas the real world HAR is more

variable because sensor orientation, clothing, environment, and subject behavior are more variable.

There is a significant drawback in the form of random train/test split: segments of the same subjects might be included in the same set, and hence, the models could use subject-specific trends. There may be a more realistic test of cross-subject generalization with a more stringent review, like leave-one-subject-out cross-validation.

6.4 Trends in the Machine Learning Augmented Reality (MLO5)

The main trends in the development of HAR and ML (MLO5) are:

- Raw sensor Deep Learning In essence, Convolutional and recurrent networks directly learn hierarchical representations on raw time-series data, and tend to perform better than the classical feature-based methods when learning on very large datasets.
- Self-supervised and representation learning Techniques such as contrastive learning learn useful representations on unlabelled sensor data and minimize the use of labeled data.
- Multimodal fusion - This type of fusion is made of inertial data in addition to visual, GPS or audio data to improve the ability to interpret the context and to be more robust.
- Even though the classical models are used in this project, the lessons learned regarding feature engineering, evaluation, and trade-offs will be a good basis upon which new methods can be critiqued and incorporated.

6.5 Conclusions

The current paper shows that classical machine-learning algorithms using features designed well enough can work outstandingly well on the Daily and Sports Activities dataset. Among the evaluated methods:

- SVM gives the accuracy and macro-F1 the best.
- KNN gives an useful distance based baseline.
- Decision Trees give understandable models at a slightly worse rate.
- K-means clustering has quick partial structure building

References:

- Lara, O. D., & Labrador, M. A. (2013). A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys & Tutorials*, 15(3), 1192–1209. <https://ieeexplore.ieee.org/document/6479954>
- Kwapisz, J. R., Weiss, G. M., & Moore, S. A. (2011). Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter*, 12(2), 74–82. <https://doi.org/10.1145/1964897.1964918>
- Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification* (2nd ed.). Wiley.
- Hammerla, N. Y., Halloran, S., & Plötz, T. (2016). Deep, convolutional, and recurrent models for human activity recognition using wearables. In *Proceedings of the 25th International Joint*

capabilities, but does not match supervised classifiers.

In general, the study was able to:

- Fundamentals of ML concepts and application(MLO1).
- Data processing and analysis using different algorithms (MLO2).
- Comparing HAR learning methods (MLO3).
- Defining and implementing appropriate algorithms to a real-life scenario (MLO4).
- Examining current and future trends in ML to HAR (MLO5).

Conference on Artificial Intelligence (IJCAI-16) (pp. 1533–1540).

<https://www.ijcai.org/Proceedings/16/Papers/327.pdf>

- Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.
- Wang, J., Chen, Y., Hao, S., Peng, X., & Hu, L. (2019). Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119, 3–11. <https://doi.org/10.1016/j.patrec.2018.02.010>
- Kwapisz, J. R., Weiss, G. M., & Moore, S. A. (2011). Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter*, 12(2), 74–82. <https://dl.acm.org/doi/10.1145/1964897.1964918>

Appendix A – Dataset

Item	Description
Dataset type	Multivariate time-series from wearable inertial/magnetic sensors
Number of activities	19 (A1–A19: daily and sports activities)
Number of subjects	8 (4 female, 4 male, aged 20–30)
Sensors per time step	45 (5 units × 9 axes: accelerometer, gyroscope, magnetometer)
Sensor locations	Torso (T), Right Arm (RA), Left Arm (LA), Right Leg (RL), Left Leg (LL)
Sampling frequency	25 Hz
Segment length	5 seconds (125 samples)
Segments per subject/activity	60
Total segments	$19 \times 8 \times 60 = 9120$
Raw segment shape	125×45 (time steps × channels)
Feature vector length	180 (mean, std, min, max for each of 45 channels)
Missing values	None reported
Data format	Text files: aXX/pY/sZZ.txt (one 5s segment per file)
Example folder structure	.../data/a01/p1/s01.txt

Zip file URL : <https://genomics.senescence.info/species/dataset.zip>



daily+and+sports+activities.zip

Appendix B

B.1 Future Extraction:

```

BASE_DIR = r"C:\Users\tamma\Downloads\daily+and+sports+activities\data"
# <-- folder containing a01, a02, ..., a19

def extract_features_from_segment(seg):
    """
    seg: numpy array shape (125, 45)
    Returns: 1D feature vector of length 45*4 = 180
    """
    # axis 0: time; axis 1: channels
    means = seg.mean(axis=0)
    stds = seg.std(axis=0)
    mins = seg.min(axis=0)
    maxs = seg.max(axis=0)
    feat = np.concatenate([means, stds, mins, maxs])
    return feat

```

B.2 Dataset Loading and Feature Matrix:

```

def load_dataset(base_dir):
    X = []
    y = []
    subjects = []

    # Activities a01..a19
    for a in range(1, 20):
        a_folder = f"a{a:02d}"
        activity_dir = os.path.join(base_dir, a_folder)

        # Subjects p1..p8
        for p in range(1, 9):
            p_folder = f"p{p}"
            subject_dir = os.path.join(activity_dir, p_folder)

            # Segments s01..s60
            for s in range(1, 61):
                s_file = f"s{s:02d}.txt"
                file_path = os.path.join(subject_dir, s_file)

                if not os.path.exists(file_path):
                    # In case some files are missing (should not happen)
                    continue

                # Each file: 125 rows x 45 columns, comma separated
                seg = np.loadtxt(file_path, delimiter=",")
                # Ensure correct shape
                if seg.shape != (125, 45):
                    # Try resizing or skip
                    continue

                feat = extract_features_from_segment(seg)
                X.append(feat)
                y.append(a)      # activity label (1..19)
                subjects.append(p) # subject id (1..8)

    X = np.array(X)
    y = np.array(y)
    subjects = np.array(subjects)
    return X, y, subjects

```

B.3 Train–Test Split, Scaling, and PCA:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# PCA (optional)
pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Original feature dimension:", X_train_scaled.shape[1])
print("PCA dimension (95% var):", X_train_pca.shape[1])
```

Appendix C: Entire code

```
import os
import numpy as np
import pandas as pd
from glob import glob

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score, f1_score, classification_report,
    confusion_matrix
)
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score, adjusted_rand_score

# =====
# 1. LOAD & FEATURE EXTRACTION
# =====

BASE_DIR = r"C:\Users\tamma\Downloads\daily+and+sports+activities\data"
# <- folder containing a01, a02, ..., a19

def extract_features_from_segment(seg):
    """
    seg: numpy array shape (125, 45)
    Returns: 1D feature vector of length 45*4 = 180
    """
    # axis 0: time; axis 1: channels
    means = seg.mean(axis=0)
    stds = seg.std(axis=0)
    mins = seg.min(axis=0)
    maxs = seg.max(axis=0)
    feat = np.concatenate([means, stds, mins, maxs])
    return feat

def load_dataset(base_dir):
    X = []
    y = []
    subjects = []

    # Activities a01..a19
    for a in range(1, 20):
        a_folder = f"a{a:02d}"
```

```
activity_dir = os.path.join(base_dir, a_folder)

# Subjects p1..p8
for p in range(1, 9):
    p_folder = f"p{p}"
    subject_dir = os.path.join(activity_dir, p_folder)

# Segments s01..s60
for s in range(1, 61):
    s_file = f"s{s:02d}.txt"
    file_path = os.path.join(subject_dir, s_file)

    if not os.path.exists(file_path):
        # In case some files are missing (should not happen)
        continue

    # Each file: 125 rows x 45 columns, comma separated
    seg = np.loadtxt(file_path, delimiter=",")
    # Ensure correct shape
    if seg.shape != (125, 45):
        # Try resizing or skip
        continue

    feat = extract_features_from_segment(seg)
    X.append(feat)
    y.append(a)      # activity label (1..19)
    subjects.append(p) # subject id (1..8)

X = np.array(X)
y = np.array(y)
subjects = np.array(subjects)
return X, y, subjects

print("Loading dataset (this may take a little while)...")
X, y, subjects = load_dataset(BASE_DIR)
print("Data loaded.")
print("X shape:", X.shape) # expected: (19 * 8 * 60, 180) = (9120, 180)
print("y shape:", y.shape)

# =====
# 2. TRAIN-TEST SPLIT & SCALING
# =====

# Stratified split by activity
X_train, X_test, y_train, y_test = train_test_split(
```

```

# Stratified split by activity
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# PCA (optional)
pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Original feature dimension:", X_train_scaled.shape[1])
print("PCA dimension (95% var):", X_train_pca.shape[1])

# =====
# 3. MODEL TRAINING & EVALUATION
# =====

def evaluate_model(model, X_tr, X_te, y_tr, y_te, label):
    model.fit(X_tr, y_tr)
    y_pred = model.predict(X_te)
    acc = accuracy_score(y_te, y_pred)
    f1 = f1_score(y_te, y_pred, average='macro')
    print(f"\n== {label} ==")
    print("Accuracy:", acc)
    print("Macro F1:", f1)
    print("Classification report:\n", classification_report(y_te, y_pred))
    cm = confusion_matrix(y_te, y_pred)
    print("Confusion matrix:\n", cm)
    return acc, f1

results = []

# SVM (no PCA)
svm = SVC(kernel='rbf', C=10, gamma='scale', random_state=42)
acc, f1 = evaluate_model(svm, X_train_scaled, X_test_scaled, y_train, y_test, "SVM (no PCA)")
results.append(["SVM (RBF, no PCA)", acc, f1])

# KNN (no PCA)
knn = KNeighborsClassifier(n_neighbors=5)
acc, f1 = evaluate_model(knn, X_train_scaled, X_test_scaled, y_train, y_test, "KNN (no PCA)")
results.append(["KNN (k=5, no PCA)", acc, f1])

```

```

# Decision Tree (no PCA)
dt = DecisionTreeClassifier(max_depth=20, random_state=42)
acc, f1 = evaluate_model(dt, X_train_scaled, X_test_scaled, y_train, y_test, "Decision Tree (no PCA)")
results.append(["Decision Tree (no PCA)", acc, f1])

# SVM (with PCA)
svm_pca = SVC(kernel='rbf', C=10, gamma='scale', random_state=42)
acc, f1 = evaluate_model(svm_pca, X_train_pca, X_test_pca, y_train, y_test, "SVM (PCA)")
results.append(["SVM (RBF, PCA)", acc, f1])

# KNN (with PCA)
knn_pca = KNeighborsClassifier(n_neighbors=5)
acc, f1 = evaluate_model(knn_pca, X_train_pca, X_test_pca, y_train, y_test, "KNN (PCA)")
results.append(["KNN (k=5, PCA)", acc, f1])

# Decision Tree (with PCA)
dt_pca = DecisionTreeClassifier(max_depth=20, random_state=42)
acc, f1 = evaluate_model(dt_pca, X_train_pca, X_test_pca, y_train, y_test, "Decision Tree (PCA)")
results.append(["Decision Tree (PCA)", acc, f1])

results_df = pd.DataFrame(results, columns=["Model", "Accuracy", "Macro F1"])
print("\n== Summary of supervised results ==")
print(results_df)

# =====
# 4. OPTIONAL: K-MEANS CLUSTERING
# =====

k = len(np.unique(y)) # 19 clusters
kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
cluster_labels = kmeans.fit_predict(X_test_pca)

sil = silhouette_score(X_test_pca, cluster_labels)
db = davies_bouldin_score(X_test_pca, cluster_labels)
ari = adjusted_rand_score(y_test, cluster_labels)

print("\n== K-means clustering (PCA features) ==")
print("Silhouette score:", sil)
print("Davies-Bouldin index:", db)
print("Adjusted Rand Index:", ari)

```



Output: output.txt

Appendix D: Summary:

Table D1. Summary of supervised models:

Model	Accuracy	Macro F1
SVM (RBF, no PCA)	0.9916	0.9916
KNN (k=5, no PCA)	0.9832	0.9828
Decision Tree (no PCA)	0.9770	0.9767
SVM (RBF, PCA)	0.9912	0.9912
KNN (k=5, PCA)	0.9828	0.9824
Decision Tree (PCA)	0.9605	0.9608

Table D2. Clustering metrics on PCA:

Metric	Value
Silhouette score	0.2399
Davies–Bouldin index	1.5102
Adjusted Rand Index	0.5568

Word count -2949 words