

Machine Learning Project Report
on
HUMAN ACTIVITY RECOGNITION USING SMARTPHONE

Submitted by

G.V.V. Lalith Krishna (22501A0563)
G. Saethu Sai (22501A0561)
G. Devi Priya Anjali (22501A0558)
A. Sai Anil Kumar (22501A0516)

Under the Esteemed Guidance of

Dr. G. Lalitha Kumari M.Tech.,Ph.D.

Senior Assistant Professor

Department of Computer Science and Engineering



Department of Computer Science and Engineering

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

(Permanently affiliated to JNTU: Kakinada, Approved by AICTE)

(An NBA & NAAC accredited and ISO 9001:2015 certified institution)

Kanuru, Vijayawada-520007

2024-2025

Signature of the Guide

Dr. G.Lalitha Kumari,

Senior Assistant Professor.

Signature of the HOD

Dr. A. Jaya Lakshmi,

Professor & HOD

TABLE OF CONTENTS

S.NO	CONTENT	PAGE NO
1	INTRODUCTION 1.Problem Statement 2.Objective 3.Proposed Model	1 2 3
2	MATERIALS AND METHODS 1.Requirements 2.Dataset 3.Supervised Machine Learning Methods 4.Evaluation Parameters	4 5 6 7 - 8
3	METHODOLOGY	9 - 15
4	IMPLEMENTATION	16-26
5	RESULT	27
6	CONCLUSION	28
7	FUTURE SCOPE	29
8	REFERENCES	30

ABSTRACT

Human Activity Recognition (HAR) using smartphones leverages machine learning and sensor data analysis to classify and predict human activities based on motion and orientation data. With the widespread use of smartphones equipped with accelerometers, gyroscopes, and magnetometers, HAR has become a valuable tool in healthcare, fitness tracking, and human-computer interaction. This project focuses on developing an intelligent HAR system that utilizes real-time sensor data to recognize activities such as walking, running, sitting, standing, and lying down.

A supervised machine learning approach is adopted, utilizing algorithms such as Support Vector Machines (SVM), Random Forest, and Logistic Regression . The system is rigorously evaluated using metrics such as accuracy, precision, recall, and F1-score to ensure reliable performance. This HAR system has broad applications, including fitness tracking, elderly care, rehabilitation, and security monitoring. By integrating smartphone sensors with machine learning techniques, the project aims to deliver an efficient, scalable, and real-time activity recognition system with high accuracy.

Keywords

1. Human Activity Recognition (HAR)
2. Smartphone Sensors (Accelerometer, Gyroscope, Magnetometer)
3. Machine Learning Algorithms (SVM, Random Forest, KNN)
4. Feature Extraction & Data Preprocessing
5. Classification Metrics
6. Real-Time Activity Classification
7. Healthcare & Fitness Applications

1. INTRODUCTION

1.1 Problem Statement

The rise of smartphones equipped with motion sensors like accelerometers and gyroscopes presents an opportunity for intelligent **Human Activity Recognition (HAR)** systems. Traditional activity monitoring methods are often intrusive or costly, making smartphone-based HAR a convenient alternative. However, challenges such as **noisy sensor data, user variability, and real-time processing constraints** affect accuracy. This project aims to develop a **machine learning-based HAR system** that efficiently processes smartphone sensor data to classify activities like walking, running, sitting, and standing. The solution has applications in **healthcare, fitness tracking, and human-computer interaction**, providing a scalable and non-intrusive method for real-time activity recognition.

1.2 Objective

The main objective of this project is to develop an efficient **Human Activity Recognition (HAR) system** using smartphone sensor data. The system aims to accurately classify activities such as **walking, running, sitting, standing, and lying down** by leveraging machine learning techniques.

The specific objectives are:

- **Accurate Activity Classification** – Recognize and classify activities like **walking, running, sitting, standing, and lying down** with high accuracy.
- **Sensor Data Processing** – Preprocess raw data from **accelerometers and gyroscopes**, removing noise and extracting meaningful features.
- **Machine Learning Model Development** – Train and evaluate models like **SVM, Random Forest, KNN**, for optimal performance.
- **Real-time Recognition** – Ensure the system works in **real-time** while handling variations in user behavior and smartphone placement.
- **Scalability & Efficiency** – Design a solution that is **lightweight, scalable, and non-intrusive**, making it easy to deploy on smartphones.
- **Applications in Various Domains** – Implement the system for use cases like **healthcare (elderly monitoring), fitness tracking**.

1.3 Proposed Model

The proposed model follows a structured pipeline that processes raw sensor data from a smartphone and classifies human activities using machine learning techniques. The system consists of the following key components:

1. Data Collection

- Sensor data (accelerometer & gyroscope) is collected from a smartphone.
- Activities such as **walking, running, sitting, standing, and lying down** are recorded.
- The dataset is labeled to train the model effectively.

2. Data Preprocessing

- **Noise Removal:** Filters are applied to reduce sensor noise.
- **Feature Extraction:** Extract time-domain and frequency-domain features like mean, standard deviation, and entropy.
- **Normalization:** Scales the data to ensure uniformity across different smartphone placements.

3. Machine Learning Model Training

- Train models like **Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors (KNN)**.
- Evaluate models using **accuracy, precision, recall, and F1-score**.

4. Real-time Activity Recognition

- The trained model is deployed on a smartphone application.
- Sensor data is continuously processed to classify activities in real-time.

5. Performance Evaluation & Optimization

- Test the system on different users and device placements.
- Optimize the model for efficiency, ensuring **low computational cost and real-time processing**.

6. Applications & Deployment

- The final model can be used in **healthcare (elderly monitoring), fitness tracking, and human-computer interaction**.
- The system can be integrated into **mobile applications** for real-world use.

2. MATERIALS AND METHODS

2.1 Requirements

1. Hardware Requirements

- Smartphone with built-in accelerometer and gyroscope sensors for data collection.
- Computer/Laptop for model training and development.

2. Software Requirements

- Operating System: Windows, macOS, or Linux.
- Programming Language: Python (preferred).
- Development Environment: Jupyter Notebook, Google Colab, or PyCharm.
- Libraries & Frameworks:
 - Data Handling & Preprocessing: NumPy, Pandas, Scikit-learn
 - Data Visualization: Matplotlib, Seaborn
 - Machine Learning Models: Scikit-learn, TensorFlow/Keras, PyTorch (if deep learning is used)

3. Dataset Requirements

- Publicly Available Dataset: Example – UCI HAR dataset (if not collecting own data).

4. Functional Requirements

- Data Collection Module: To capture real-time accelerometer and gyroscope data.
- Data Preprocessing Module: To clean, normalize, and extract features from raw sensor data.
- Model Training & Evaluation Module: To build and optimize machine learning models.
- Real-time Prediction Module: To classify activities based on live sensor input.
- Deployment Module: To integrate the model into a mobile or web application

2.2 Dataset

A commonly used dataset for **HAR** is the **UCI Human Activity Recognition Dataset**, which contains motion sensor data collected from smartphones while users perform various activities.

Important Columns in the Dataset:

1. **Activity** – The target label (e.g., Walking, Sitting, Standing, Lying Down, Walking Upstairs, Walking Downstairs).
2. **Subject ID** – Identifies the user performing the activity.
3. **Timestamp** – Records the time of each sensor reading.
4. **Accelerometer Readings (X, Y, Z)** – Measures acceleration in three directions.
5. **Gyroscope Readings (X, Y, Z)** – Captures angular velocity in three directions.
6. **Body Acceleration (X, Y, Z)** – Extracted from the accelerometer after removing gravity effects.
7. **Gravity Acceleration (X, Y, Z)** – Represents the gravitational force component from the accelerometer.
8. **Jerk Signals (X, Y, Z)** – Measures sudden changes in motion.
9. **Magnitude Features** – Overall magnitude of acceleration and gyroscope data.
10. **Statistical Features** – Mean, standard deviation, entropy, etc., computed for each sensor signal.

Dataset Source:

UCI Machine Learning Repository

Link : <https://archive.ics.uci.edu/dataset/240/human+activity+recognition+using+smartphones>

Dataset Shape :

- (7352, 564) → Training Set
 - 7352 → Number of rows (data samples) in the training dataset.
 - 564 → Number of columns (features + target label).
- (2947, 564) → Test Set
 - 2947 → Number of rows (data samples) in the test dataset.
 - 564 → Number of columns (features + target label).

2.3 Supervised Machine Learning Methods

1. Logistic Regression

Logistic Regression serves as a baseline model for classifying human activities based on smartphone sensor data. By establishing a linear decision boundary between activity classes, it predicts the likelihood of a given activity (e.g., walking, sitting, or running) based on extracted features such as acceleration, angular velocity, and jerk signals. Despite its simplicity, Logistic Regression provides an interpretable classification model for basic HAR tasks.

2. Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a powerful supervised learning algorithm used for activity classification. It works by finding the optimal hyperplane that best separates different human activities in a high-dimensional space using sensor data. SVMs are highly effective in handling complex activity patterns and can be enhanced with kernel functions to manage non-linearity, making them suitable for HAR applications with structured and well-separated sensor features.

3. Random Forest

Random Forest, an ensemble learning technique, improves activity recognition accuracy by combining multiple decision trees. Each tree is trained on a random subset of the dataset, reducing overfitting and enhancing model robustness. The final classification is determined by majority voting across all trees, ensuring better generalization and adaptability to variations in sensor readings due to different smartphone placements and user behaviors.

4. K-Nearest Neighbors (KNN)

KNN classifies human activities by comparing incoming sensor readings with past instances of labeled activity data. It identifies the most similar recorded activity patterns based on features like acceleration and gyroscope signals, assigning the most frequent label among the nearest neighbors. This approach is particularly useful for recognizing activities with subtle differences and is effective in capturing sensor data variations without requiring an explicit model training phase.

2.4 Evaluation Parameters

To assess the performance of different machine learning models in classifying human activities, we use the following evaluation metrics:

1. Accuracy

Accuracy measures the overall correctness of the model by calculating the proportion of correctly classified activities out of all activity instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where:

- **TP (True Positives):** Correctly classified activity instances.
- **TN (True Negatives):** Correctly classified non-occurrences.
- **FP (False Positives):** Incorrectly classified instances.
- **FN (False Negatives):** Missed activity classifications.

Example: If the model correctly identifies 90 out of 100 activities, its accuracy is **90%**.

2. Precision

Precision measures the proportion of correctly predicted activity instances among all instances predicted as that activity. It helps evaluate how many of the identified activities are actually correct.

•

$$Precision = \frac{TP}{TP + FP}$$

Importance: High precision is important when false positives (misclassification of one activity as another) need to be minimized.

3. Recall (Sensitivity or True Positive Rate)

Recall measures the ability of the model to correctly identify all instances of a particular activity. It is useful when false negatives (missed activities) need to be minimized.

$$Recall = \frac{TP}{TP + FN}$$

Example: If the model correctly detects 80 out of 100 walking instances, recall is **80%**.

4.F1-Score

The F1-score is the harmonic mean of precision and recall, providing a balanced measure when there is an **imbalance** in activity classes.

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

Importance: Helps in scenarios where both false positives and false negatives should be minimized.

5.Confusion Matrix

A confusion matrix is a table used to evaluate the performance of classification models. It provides a detailed breakdown of correct and incorrect predictions for each activity class.

Structure:

Actual ↓ / Predicted →	Walking	Running	Sitting
Walking	TP	FP	FN
Running	FN	TP	FP
Sitting	FP	FN	TP

Use: Helps analyze which activities are being misclassified and improve the model accordingly.

3. METHODOLOGY

3.1 Data Collection

Data collection is a crucial step in Human Activity Recognition (HAR), as it directly impacts the model's accuracy and reliability. The process involves gathering sensor data from smartphones while users perform different activities.

1. Data Sources

Using Pre-existing Datasets:

UCI HAR Dataset (Widely used benchmark dataset)

- Contains data from 30 participants performing 6 activities (Walking, Sitting, Standing, etc.).
- Collected using smartphones with accelerometer and gyroscope sensors.

2. Sensor Data Collection Process

To collect data from a smartphone, use built-in motion sensors such as:

1. **Accelerometer** → Measures acceleration in X, Y, Z axes.
2. **Gyroscope** → Captures angular velocity (rotation) in X, Y, Z axes.
3. **Magnetometer** → Measures orientation based on Earth's magnetic field.

3.2 Data Preprocessing

Data preprocessing is a crucial step in preparing raw sensor data for machine learning models. It ensures the data is clean, structured, and suitable for accurate classification.

1. Data Cleaning – Handles missing values, removes duplicate entries, and ensures data consistency.
2. Noise Filtering – Sensor readings often contain noise, which is reduced using signal processing techniques like low-pass filtering.
3. Feature Extraction – Converts raw time-series data into meaningful features such as statistical measures (mean, standard deviation) and frequency-domain features (Fourier Transform).
4. Normalization – Standardizes data values to a common scale using techniques like Min-Max scaling or Z-score normalization to improve model performance.
5. Data Segmentation – Since HAR involves time-series data, it is divided into fixed-length windows to capture temporal dependencies.
6. Train-Test Splitting – The dataset is divided into training and testing sets to evaluate model performance effectively.

Preprocessing ensures that the HAR model learns from high-quality, well-structured data, improving its accuracy and efficiency in recognizing human activities.

3.3 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is essential in understanding the structure, distribution, and relationships within the dataset before applying machine learning models. EDA helps identify patterns, anomalies, and trends in the sensor data collected from smartphones.

1. Understanding the Dataset

- The UCI HAR Dataset contains **10,299 samples**, each representing a **3-second window** of recorded activity.
 - Each sample has **561 features** derived from accelerometer and gyroscope readings.
 - The target variable (**Activity Label**) represents **six activities**:
 1. Walking
 2. Walking Upstairs
 3. Walking Downstairs
 4. Sitting
 5. Standing
 6. Laying
-

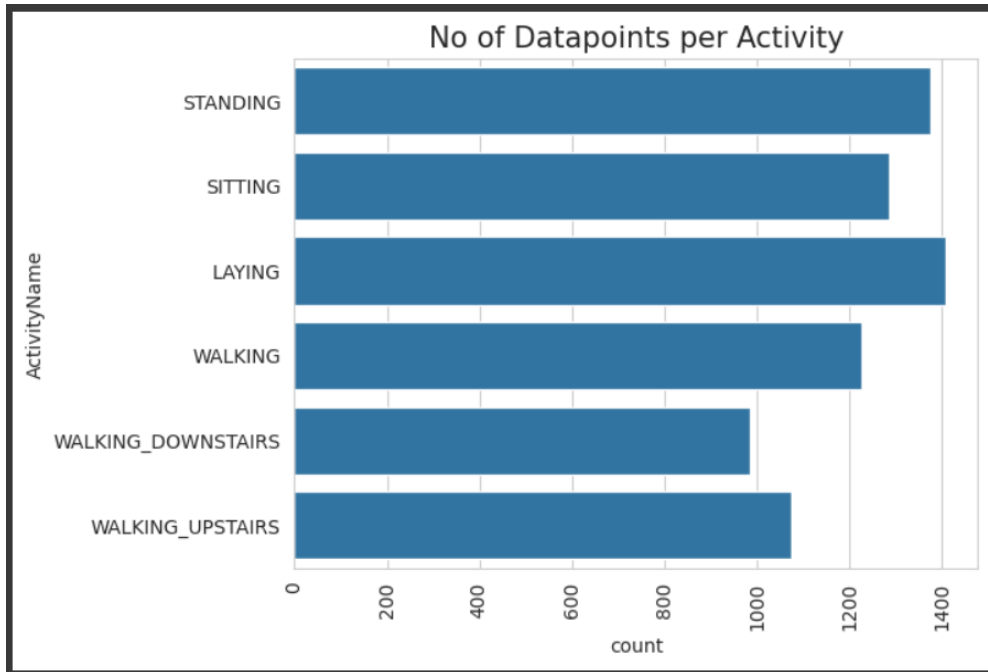
2. Statistical Summary of Features

- Checking the basic statistics like **mean, median, standard deviation, and range** for sensor readings.
 - Identifying whether data is skewed or normally distributed.
-

3. Data Visualization

A. Class Distribution of Activities

- A bar plot can show how many samples are available for each activity to check if the dataset is balanced.



B. Correlation Heatmap

- A heatmap can identify relationships between different sensor features, helping in feature selection.

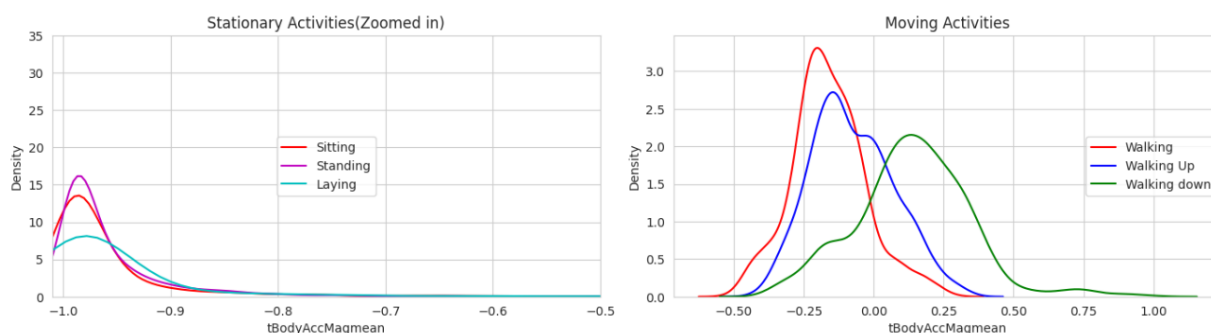
C. Sensor Data Trends Over Time

- Line plots of accelerometer and gyroscope readings can show variations in sensor signals for different activities.

D. Pairwise Feature Relationships

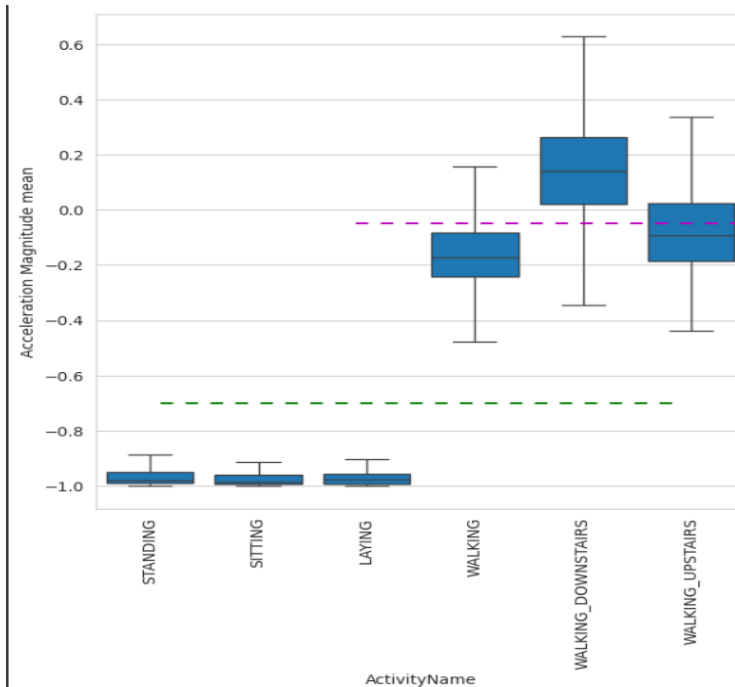
- Scatter plots can help visualize how different sensor readings relate to each other and distinguish activities.
- Static and Dynamic Activities**
 - In static activities (sit, stand, lie down) motion information will not be very useful.
 - In the dynamic activities (Walking, WalkingUpstairs, WalkingDownstairs) motion info will be significant.

Stationary and Moving activities are completely different



4. Feature Importance Analysis

- Using techniques like Principal Component Analysis (PCA) or Feature Correlation to identify the most significant features contributing to activity classification.



Observations:

- If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying.
- If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or WalkingUpstairs.
- If tAccMean > 0.0 then the Activity is WalkingDownstairs.
- We can classify 75% the Activity labels with some errors.

5. Identifying Outliers

- Box plots can help detect anomalies in sensor readings that might need further processing or removal.

3.4. Splitting Data into Training & Testing Sets

To train and evaluate machine learning models effectively, the dataset must be divided into training and testing sets. This ensures that the model learns from one portion of the data and is tested on unseen data to measure its performance.

1. Standard Data Splitting Ratio

- Training Set (70%) – Used to train the machine learning model.
- Testing Set (30%) – Used to evaluate model performance on unseen data.

Since HAR involves multiple activity classes, **stratified splitting** ensures that each activity has proportional representation in both training and testing sets. This prevents class imbalance issues

3.5. Model Selection & Training

Multiple supervised machine learning models are trained to optimize marine routes:

1. Logistic Regression – Baseline Model

- A simple classification algorithm that assumes a **linear relationship** between input features and activity labels.
- Uses the **sigmoid function** to assign probabilities to each class and selects the class with the highest probability.
- Works well if the sensor data features are **linearly separable**, but HAR data is often complex and nonlinear.

2. Support Vector Machine (SVM) – High-Dimensional Classification

- SVM finds the optimal hyperplane that separates different activity classes in a high-dimensional space.
- Uses kernel functions (e.g., RBF kernel) to handle non-linearly separable data.

- HAR data consists of 561 features from accelerometer and gyroscope signals, making SVM a strong candidate due to its ability to handle high-dimensional data.

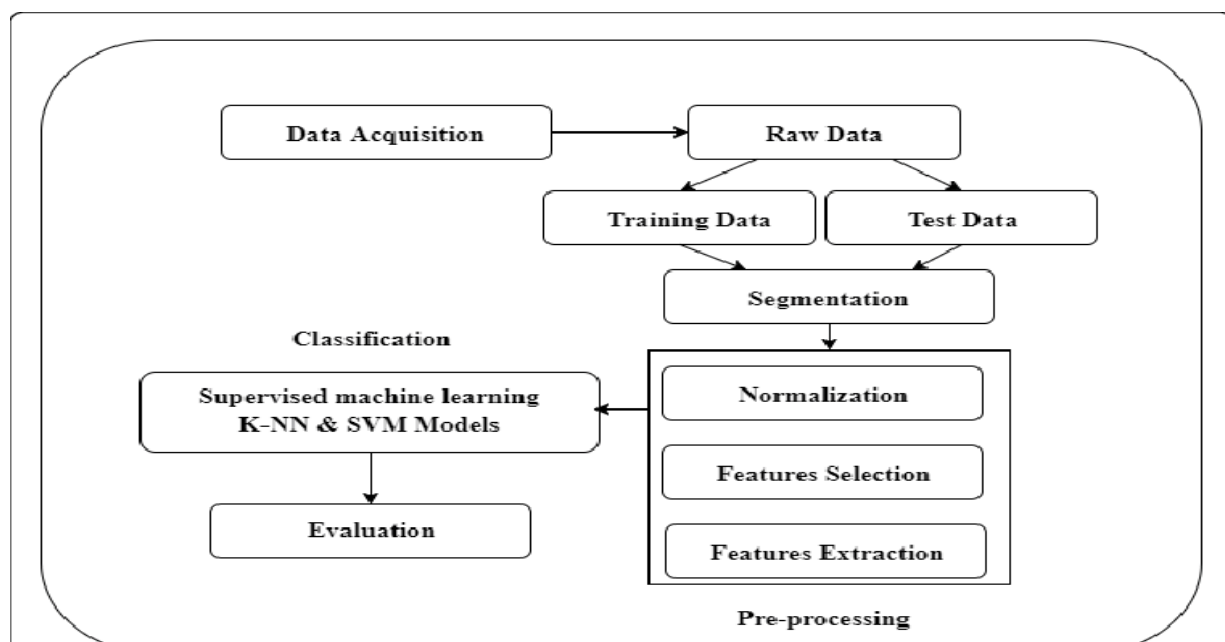
4. K-Nearest Neighbors (KNN) – Instance-Based Learning

- A distance-based algorithm that assigns labels to new samples by looking at the K most similar (nearest) training samples.
- Works well if activities have distinct motion patterns (e.g., walking vs. sitting)
- Best suited for small-scale HAR applications but not ideal for real-time classification due to high latency.

3.6. Model Evaluation

Key Evaluation Metrics:

1. **Accuracy** – Measures how often the model makes correct predictions overall.
2. **Precision** – Measures how many of the predicted activities were actually correct.
3. **Recall (Sensitivity)** – Measures how well the model correctly identifies actual activities.
4. **F1-Score** – A balance between precision and recall, useful when classes are imbalanced.
5. **Confusion Matrix** – A table that shows where the model makes correct and incorrect predictions for each activity.



4.IMPLEMENTATION

The implementation of the Human Activity Recognition (HAR) system involves several key stages, from data collection to model evaluation. The process begins with sensor data acquisition from smartphone accelerometers, gyroscopes, and magnetometers, capturing movement patterns for activities like walking, running, sitting, standing, and lying down.

The collected raw data undergoes Exploratory Data Analysis (EDA), including data cleaning, missing value handling, feature extraction, and normalization to improve accuracy. Traditional machine learning algorithms such as Support Vector Machines (SVM), Random Forest, and K-Nearest Neighbors (KNN) are trained on labeled activity data. The models are evaluated using classification metrics like accuracy, precision, recall, and F1-score to ensure optimal performance. Finally, the system is fine-tuned and tested to classify real-time activity inputs, making it suitable for applications in fitness tracking, healthcare monitoring, and security systems

Importing the Required Libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import itertools
from datetime import datetime
from sklearn import linear_model, metrics
from sklearn.model_selection import GridSearchCV
import warnings
from sklearn.ensemble import Random Forest Regression
from sklearn.linear_model import Linear Regression
from sklearn.neighbors import K Neighbors Classifier
from sklearn.svm import Linear SVC
```

Load the Data

```
from google.colab import drive
drive.mount('/content/drive')
train = pd.read_csv('/content/drive/My Drive/Project/Train.csv')
test = pd.read_csv('/content/drive/My Drive/Project/TestP.csv')
```

Dataset Size:

```
print(train.shape, test.shape)
```

```
(7352, 564) (2947, 564)
```

Get the Basic Information of the Dataset:

```
train.head(5)
```

meangravityMean	angletBodyGyroJerkMeangravityMean	angleXgravityMean	angleYgravityMean	angleZgravityMean	subject	Activity	ActivityName
-0.464761	-0.018446	-0.841247	0.179941	-0.058627	1	5	STANDING
-0.732626	0.703511	-0.844788	0.180289	-0.054317	1	5	STANDING
0.100699	0.808529	-0.848933	0.180637	-0.049118	1	5	STANDING

Printing data info:

```
print(train.info())
```

```
print(train.info());  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7352 entries, 0 to 7351  
Columns: 564 entries, tBodyAccmeanX to ActivityName  
dtypes: float64(561), int64(2), object(1)  
memory usage: 31.6+ MB  
None
```

```
print(train.columns)
```

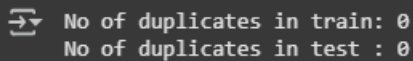
```
print(train.columns)  
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',  
       'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',  
       'tBodyAccmadZ', 'tBodyAccmaxX',  
       ...,  
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',  
       'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',  
       'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',  
       'subject', 'Activity', 'ActivityName'],  
      dtype='object', length=564)
```

Exploratory Data Analysis (EDA)

Data Preprocessing:

Checking for Duplicates:

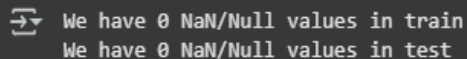
```
print('No of duplicates in train: {}'.format(sum(train.duplicated())))  
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```



```
➦ No of duplicates in train: 0  
No of duplicates in test : 0
```

Checking for NaN Values:

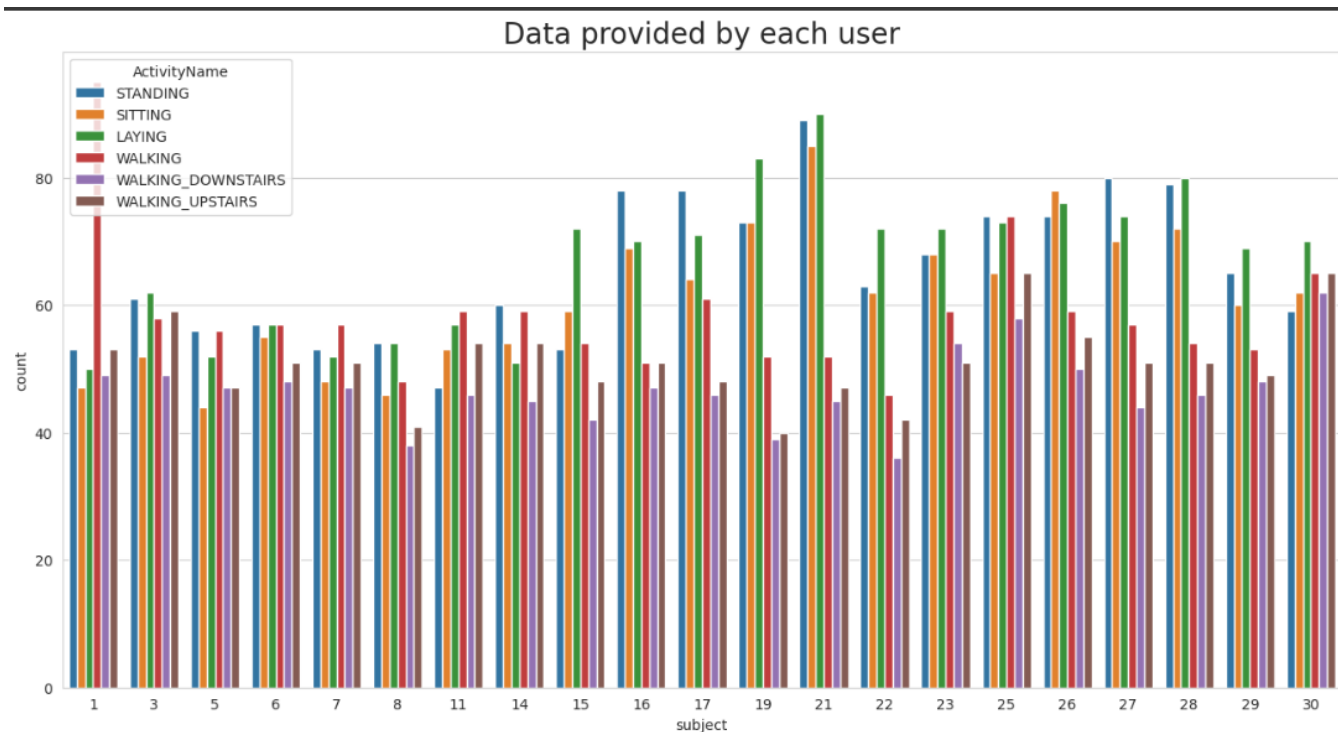
```
print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))  
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```



```
➦ We have 0 NaN/Null values in train  
We have 0 NaN/Null values in test
```

Check for data imbalance:

```
import matplotlib.pyplot as plt  
  
import seaborn as sns  
  
sns.set_style('whitegrid')  
  
plt.rcParams['font.family'] = 'Dejavu Sans'  
  
plt.figure(figsize=(16,8))  
  
plt.title('Data provided by each user', fontsize=20)  
  
sns.countplot(x='subject',hue='ActivityName', data = train)  
  
plt.show()
```

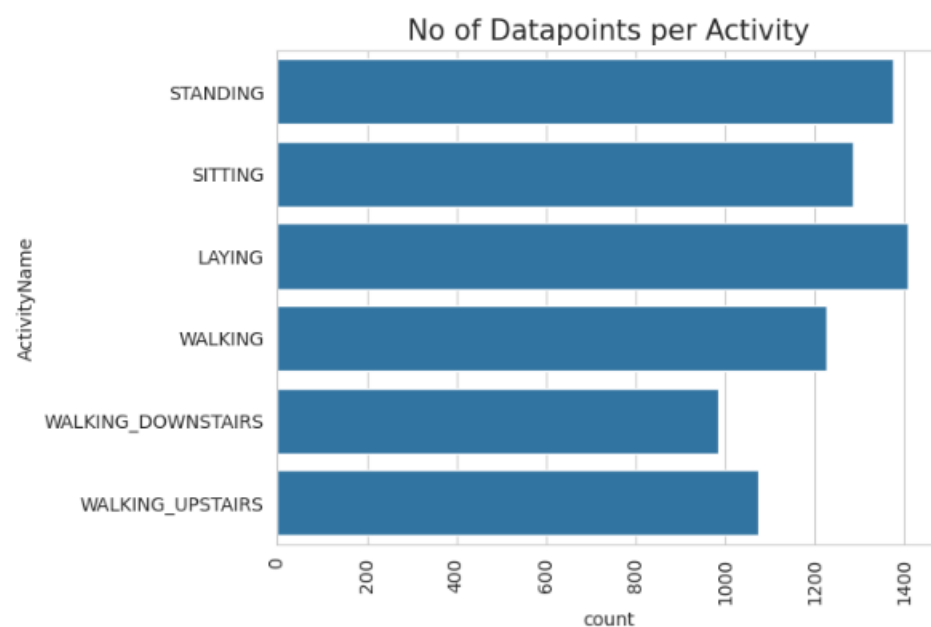


```
plt.title('No of Datapoints per Activity', fontsize=15)
```

```
sns.countplot(train.ActivityName)
```

```
plt.xticks(rotation=90)
```

```
plt.show()
```



Observation : Our data is well balanced (almost)

Separating features (X) and labels (y) from the dataset.

```
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
```

```
y_train = train.ActivityName
```

```
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
```

```
y_test = test.ActivityName
```

```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
```

```
print('X_test and y_test : ({},{})'.format(X_test.shape, y_test.shape))
```

```
X_train and y_train : ((7352, 561), (7352,))
X_test and y_test : ((2947, 561), (2947,))
```

Function to plot the confusion matrix:

```
import itertools
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import confusion_matrix
```

```
plt.rcParams["font.family"] = 'DejaVu Sans'
```

```
def plot_confusion_matrix(cm, classes,
                           normalize=False,
                           title='Confusion matrix',
                           cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    ax = plt.gca()
    ax.set_ylim(-.5, 5.5)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
```

```
horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black")
```

```
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Generic function to run any model specified:

```
from datetime import datetime
def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=True, \
    print_cm=True, cm_cmap=plt.cm.Red):

    # to store results at various phases
    results = dict()

    # time at which model starts training
    train_start_time = datetime.now()
    print('training the model..')
    model.fit(X_train, y_train)
    print('Done \n \n')
    train_end_time = datetime.now()
    results['training_time'] = train_end_time - train_start_time
    print('training_time(HH:MM:SS.ms) - {}'.format(results['training_time']))

    # predict test data
    print('Predicting test data')
    test_start_time = datetime.now()
    y_pred = model.predict(X_test)
    test_end_time = datetime.now()
    print('Done \n \n')
    results['testing_time'] = test_end_time - test_start_time
    print('testing_time(HH:MM:SS.ms) - {}'.format(results['testing_time']))
    results['predicted'] = y_pred

    # calculate overall accuracy of the model
    accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
    # store accuracy in results
    results['accuracy'] = accuracy
```



```

print('-----')
print('|   Accuracy   |')
print('-----')
print('\n  {}\n\n'.format(accuracy))

# confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
results['confusion_matrix'] = cm
if print_cm:
    print('-----')
    print('| Confusion Matrix |')
    print('-----')
    print('\n {}'.format(cm))

# plot confusion matrix
plt.figure(figsize=(8,8))
# Corrected line: Changed 'b=False' to 'visible=False'
plt.grid(visible=False)
plot_confusion_matrix(cm, classes=class_labels, normalize=True, title='Normalized confusion matrix',
cmap = cm_cmap)
ax = plt.gca()
ax.set_ylim(-.5,5.5)
plt.show()

# get classification report
print('-----')
print('| Classification Report |')
print('-----')
classification_report = metrics.classification_report(y_test, y_pred)
# store report in results
results['classification_report'] = classification_report
print(classification_report)

# add the trained model to the results
results['model'] = model

return results

```

Model Selection & Training:

1. Logistic Regression with Grid Search

```
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1'], 'solver': ['liblinear']}
```

```
log_reg = linear_model.LogisticRegression()
```

```
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-1)
```

```
log_reg_grid_results = perform_model(log_reg_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

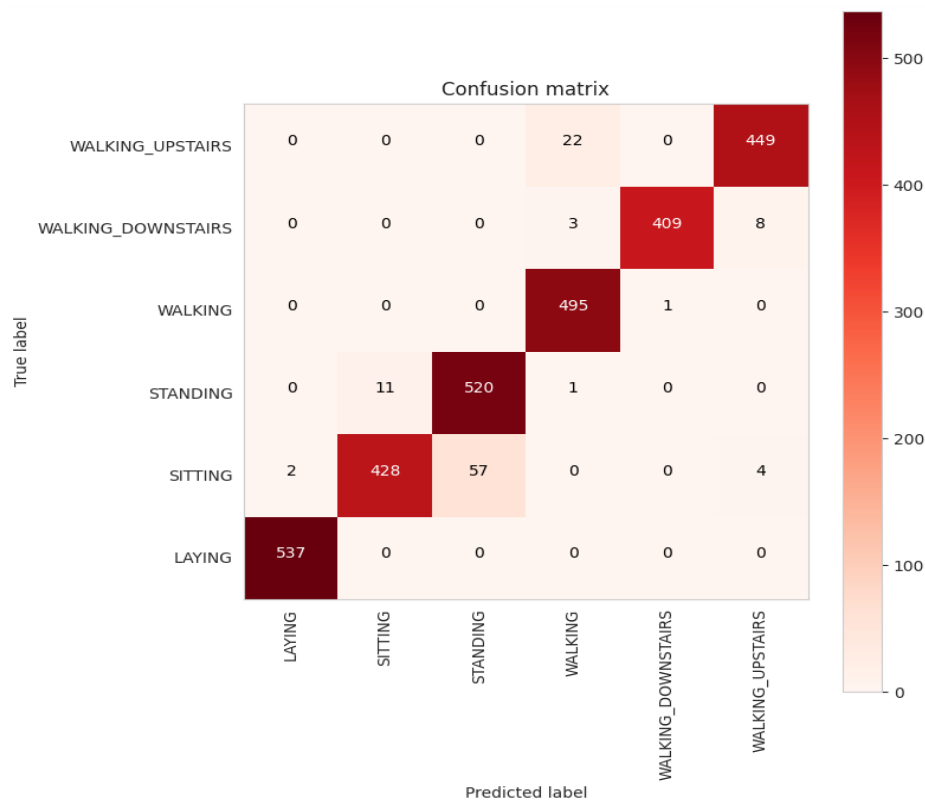
```
training_time(HH:MM:SS.ms) - 0:03:04.360798

Predicting test data
Done

testing time(HH:MM:SS.ms) - 0:00:00.014832

-----
|      Accuracy      |
-----

0.9630132337970818
```



| Classification Report |

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537
SITTING	0.97	0.87	0.92	491
STANDING	0.90	0.98	0.94	532
WALKING	0.95	1.00	0.97	496
WALKING_DOWNSTAIRS	1.00	0.97	0.99	420
WALKING_UPSTAIRS	0.97	0.95	0.96	471
accuracy			0.96	2947
macro avg	0.97	0.96	0.96	2947
weighted avg	0.96	0.96	0.96	2947

2.K Nearest Neighbour (KNN) with Grid Search

```
parameters = {'n_neighbors': [1, 10, 11, 20, 30]}
log_knn = KNeighborsClassifier(n_neighbors=6)
log_knn_grid = GridSearchCV(log_knn, param_grid=parameters, cv=3, verbose=1, n_jobs=-1)
log_knn_grid_results = perform_model(log_knn_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

training_time(HH:MM:SS.ms) - 0:00:11.648908

Predicting test data
Done

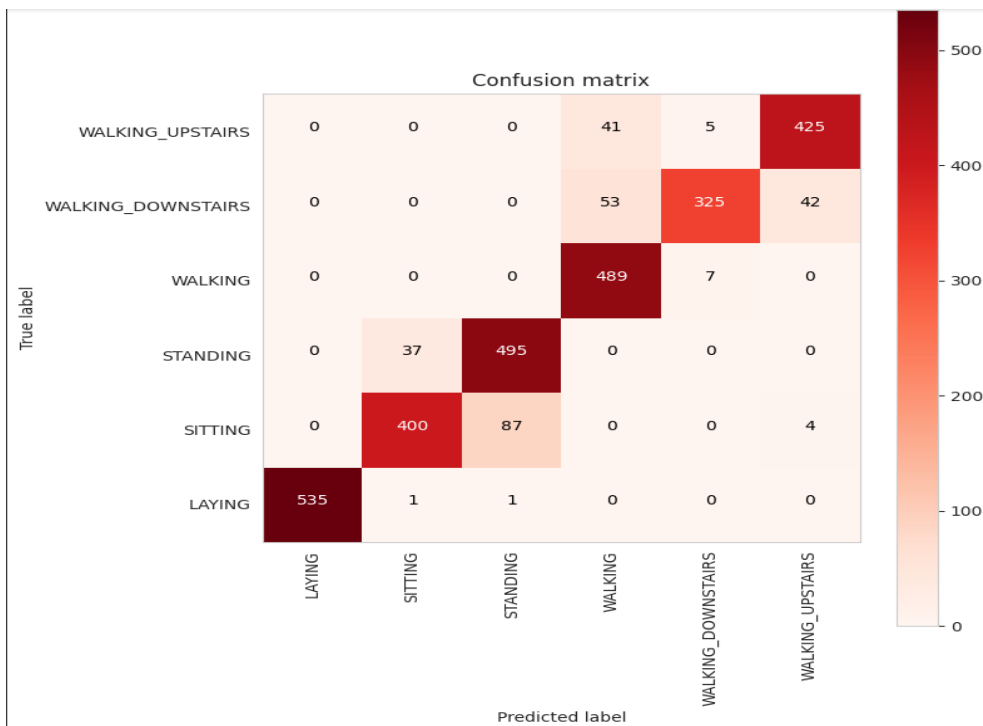
testing time(HH:MM:SS.ms) - 0:00:01.174165

| Accuracy |

0.9056667797760435

| Classification Report |

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537
SITTING	0.91	0.81	0.86	491
STANDING	0.85	0.93	0.89	532
WALKING	0.84	0.99	0.91	496
WALKING_DOWNSTAIRS	0.96	0.77	0.86	420
WALKING_UPSTAIRS	0.90	0.90	0.90	471
accuracy			0.91	2947
macro avg	0.91	0.90	0.90	2947
weighted avg	0.91	0.91	0.90	2947



3.Linear Support Vector Machine (SVC) with Grid Search

```
parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
lr_svc = LinearSVC(tol=0.00005)
lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

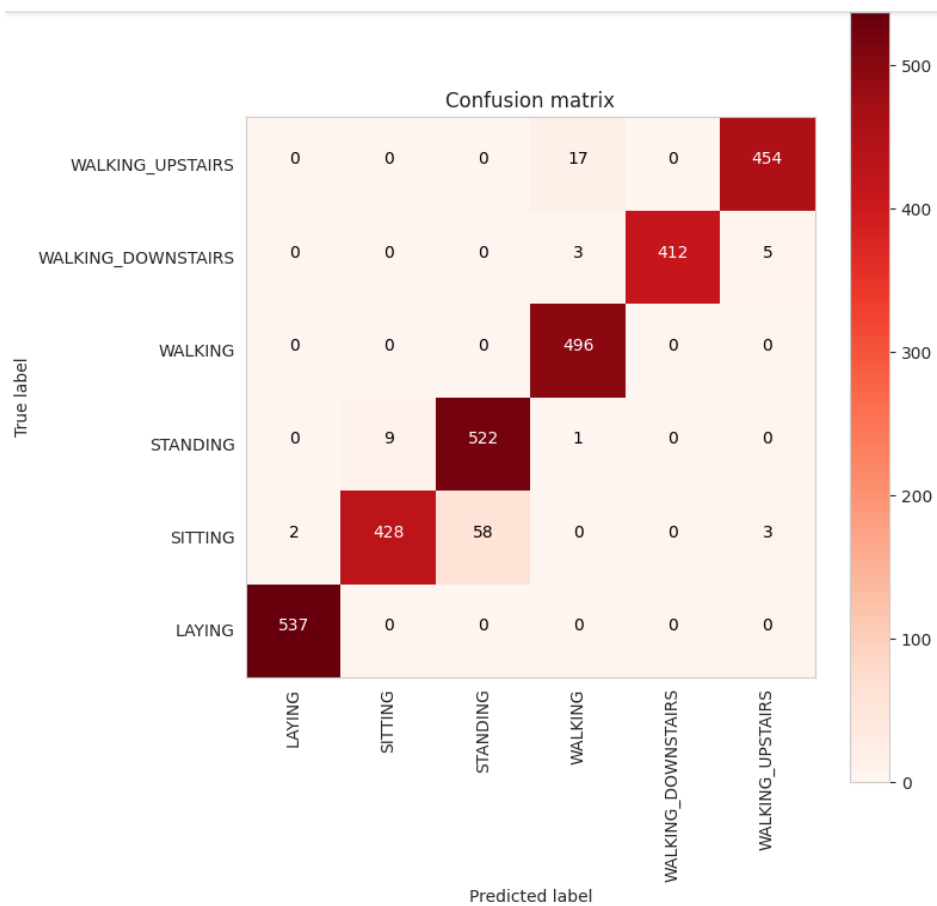
```
training_time(HH:MM:SS.ms) - 0:01:13.531077

Predicting test data
Done

testing time(HH:MM:SS.ms) - 0:00:00.011622

-----
|      Accuracy      |
-----

0.9667458432304038
```



| Classification Report |

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537
SITTING	0.98	0.87	0.92	491
STANDING	0.90	0.98	0.94	532
WALKING	0.96	1.00	0.98	496
WALKING_DOWNSTAIRS	1.00	0.98	0.99	420
WALKING_UPSTAIRS	0.98	0.96	0.97	471
accuracy			0.97	2947
macro avg	0.97	0.97	0.97	2947
weighted avg	0.97	0.97	0.97	2947

5. RESULTS

Comparing three Algorithms:

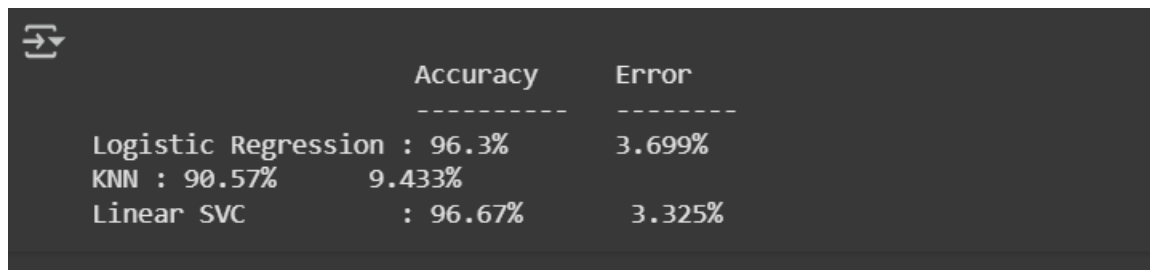
```
print('\nAccuracy  Error')

print('-----  -----')

print('Logistic Regression : {:.04}%    {:.04}%'.format(log_reg_grid_results['accuracy'] * 100,\
                                                         100-(log_reg_grid_results['accuracy'] * 100)))

print('KNN : {:.04}%    {:.04}% '.format(log_knn_grid_results['accuracy'] * 100,\
                                           100-(log_knn_grid_results['accuracy'] * 100)))

print('Linear SVC      : {:.04}%    {:.04}% '.format(lr_svc_grid_results['accuracy'] * 100,\
                                                       100-(lr_svc_grid_results['accuracy'] * 100)))
```



	Accuracy	Error
	-----	-----
Logistic Regression :	96.3%	3.699%
KNN : 90.57%	9.433%	
Linear SVC	: 96.67%	3.325%

Technique	Accuracy	Error
Logistic Regression	96.3%	3.699%
K-Nearest Neighbours (KNN)	90.57%	9.433%
Linear SVC	96.67%	3.325%

6.CONCLUSION

The **Human Activity Recognition (HAR)** using Smartphones project successfully demonstrates the application of machine learning techniques for classifying human activities based on sensor data. By leveraging accelerometer, gyroscope, and magnetometer readings, the system effectively distinguishes between different activities such as walking, running, sitting, standing, and lying down.

Through data preprocessing, feature extraction, and model selection, various machine learning algorithms, including **K-Nearest Neighbors (KNN)**, **Support Vector Machines (SVM)**, and **Logistic Regression** , were trained and evaluated. Hyperparameter tuning using GridSearchCV improved model accuracy, ensuring optimal classification performance. The system was assessed using classification metrics like accuracy, precision, recall, and F1-score, confirming its reliability.

This project has significant applications in fitness tracking, healthcare, rehabilitation, and security. Future improvements can include real-time deployment, deep learning models, and integration with IoT devices for enhanced performance. Overall, the HAR system provides a scalable and efficient solution for activity recognition, contributing to advancements in smart healthcare and human-computer interaction.

7.FUTURE SCOPE

The **Human Activity Recognition (HAR) using Smartphones** project has demonstrated the effectiveness of machine learning in classifying human activities. However, there is significant potential for improvement and expansion in various areas:

1. Real-Time Activity Recognition

- Implement the model in a **mobile application** for **live activity tracking**.
- Optimize computational efficiency for **on-device processing**.

2. Integration with IoT & Wearable Devices

- Extend the system to **smartwatches, fitness bands, and IoT sensors**.
- Improve **health monitoring and elderly care applications**.

3. Deep Learning Implementation

- Explore advanced **Deep Learning models** like **LSTMs, CNNs, and Transformers** for better accuracy.
- Utilize **transfer learning** for improved feature extraction.

4. Multimodal Data Fusion

- Combine smartphone sensor data with **GPS, heart rate, and environmental data** for enhanced predictions.

5. Activity Prediction & Anomaly Detection

- Extend the system to **predict future activities** based on behavior patterns.
- Detect **abnormal activities** for security and healthcare applications (e.g., fall detection for elderly monitoring).

6. Scalability & Cloud Deployment ☁

- Deploy the model on **cloud platforms (AWS, Google Cloud, Azure)** for large-scale usage.
- Develop a **web-based dashboard** for real-time monitoring and analytics.

7. Personalized & Adaptive Learning Models

- Train models on **individual user data** for personalized activity recognition.
- Use **reinforcement learning** to adapt to unique movement patterns over time.

By implementing these improvements, the HAR system can become a **more robust, scalable, and real-world-ready solution** for applications in **healthcare, fitness, security, and smart environments**.

8. REFERENCES

1. **Scikit-Learn Documentation**

Available at: <https://scikit-learn.org/stable/>

Description: Provides information on **classification algorithms** like **SVM, KNN, and Random Forest** used in HAR.

2. **UCI Machine Learning Repository - HAR Dataset**

Available at:

<https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones>

Description: One of the most widely used **HAR datasets**, collected from smartphone sensors.

3. **Bishop, C. M. (2006).** *Pattern Recognition and Machine Learning*
[Springer Link](#)

Covers **classification algorithms**, including **SVM, Random Forest, and KNN**.

4. **Raschka, S., & Mirjalili, V. (2019).** *Python Machine Learning*
<https://dl.ebooksworld.ir/books/Machine.Learning.with.PyTorch.and.Scikit-Learn.Sebastian.Raschka.Packt.9781801819312.EBooksWorld.ir.pdf>

Hands-on **machine learning implementation guide** in Python.

5. **Feature Engineering:** <https://www.analyticsvidhya.com/blog/2021/10/a-beginners-guide-to-feature-engineering-everything-you-need-to-know/>