

# **EFFICIENT HUMAN ACTIVITY RECOGNITION USING RANDOM PROJECTION FOREST**

## **A PROJECT REPORT**

*Submitted by*

**AUMRUDH LAL KUMAR T.J      (Regno. 201806007)**

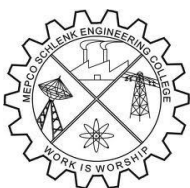
**SANJAI KUMAAR M                      (Regno. 201806040)**

*in partial fulfillment for the award of the degree  
of*

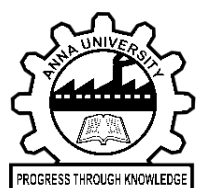
**BACHELOR OF TECHNOLOGY**

**in**

**INFORMATION TECHNOLOGY**



**DEPARTMENT OF INFORMATION TECHNOLOGY  
MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**  
(An Autonomous Institution affiliated to Anna University Chennai)



**APRIL 2022**

## **BONAFIDE CERTIFICATE**

Certified that this project report titled **EFFICIENT HUMAN ACTIVITY RECOGNITION USING RANDOM PROJECTION FOREST** is the bonafide work of **Mr. T.J AUMRUDH LAL KUMAR (Reg. No. 201806007), Mr. M. SANJAI KUMAAR (Reg. No. 201806040)** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

MRS. R. VENITTA RAJ, M.E., (Ph.D.),

**Internal guide**  
**Assistant Professor (Sl. Grade),**  
**Department of Information Technology,**  
**Mepco Schlenk Engineering College,**  
**Sivakasi.**

Dr. T. REVATHI, B.E., M.E., Ph.D.,

**Senior Professor,**  
**Head of the Department,**  
**Department of Information Technology,**  
**Mepco Schlenk Engineering College,**  
**Sivakasi.**

Submitted for viva-Voce Examination held at **MEPCO SCHLENK ENGINEERING COLLEGE (AUTONOMOUS), SIVAKASI** on .....

**Internal Examiner**

**External Examiner**

## **ABSTRACT**

---

## **ABSTRACT**

Humans do various activities in daily life such as walking, standing, laying, sitting, walking upstairs, and walking downstairs. K nearest neighbor is a famous classification and regression algorithm, used for prediction. It plays a critical and vital role in various big data and data science applications. Even though there are many classification algorithms such as logistic regression, Naive Bayes, support vector machine, and decision tree. Here we use the KNN algorithm for prediction of the human activity because it has better accuracy and runs faster and requires fewer resources and is easy to implement.

We here use a new approach for finding the nearest neighbor and for prediction. Random forest is an existing ensemble-based technique and supervised machine learning algorithm that is used for classification and regression. rpTree combines the tree-based technique's power of ensemble learning to predict the class label.

We use Arduino UNO R3 and ADXL335 (Accelerometer) as hardware components. We gather the data using this ADXL335 sensor which reads three-axes namely X, Y, and Z as input.

## **ACKNOWLEDGEMENT**

---

## ACKNOWLEDGEMENT

Apart from our efforts, the success of our project depends largely on the encouragement of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of our project. We would like to express our immense pleasure to thank our college management for giving us the required amenities for our project.

We would like to convey our sincere thanks to our respected Principal, **Dr.S.Arivazhagan, M.E., Ph.D.**, Mepco Schlenk Engineering College, for providing us with facilities to complete our project.

We extend our profound gratitude and heartfelt thanks to **Dr.T.Revathi, M.E., Ph.D.**, Senior Professor, and Head of the Department of Information Technology for providing us constant encouragement.

We are bound to thank our project coordinator **Dr.S Rajesh, ME, MBA, Ph.D.**, Associate Senior Professor of Information Technology. We sincerely thank our project guide **Mrs. R. Venitta Raj, M.E.,(Ph.D).**, Assistant Professor, Department of Information Technology, for his inspiring guidance and valuable suggestions to complete our project successfully.

The guidance and support received from all the staff members and lab technicians of our department who contributed to our project were vital for the success of the project. We are grateful for their constant support and help.

We would like to thank our parents and friends for their help and support in our project.

## TABLE OF CONTENTS

---

## TABLE OF CONTENTS

1. INTRODUCTION	2
2. LITERATURE STUDY	6
2.1 OVERVIEW	6
2.2 COVER TREES FOR NEAREST NEIGHBOR [1]	6
2.3 RANDOMIZED PARTITION TREES FOR NEAREST NEIGHBOR SEARCH [2]	6
2.4 RANDOM-PROJECTION ENSEMBLE CLASSIFICATION [3]	6
2.5 APPROXIMATE K-NEAREST NEIGHBOR BASED SPATIAL CLUSTERING USING K-D TREE [4]	7
2.6 SMARTPHONE-BASED ACTIVITY RECOGNITION USING K-NEAREST NEIGHBOR ALGORITHM [5]	7
3. SOFTWARE REQUIREMENT SPECIFICATION	10
3.1. INTRODUCTION	10
3.1.1 PURPOSE	10
3.1.2 OBJECTIVE	10
3.2. OVERVIEW	10
3.3 SPECIFIC REQUIREMENTS	12
3.3.1 FUNCTIONAL REQUIREMENTS	12
3.3.2 NON-FUNCTION REQUIREMENTS	12
3.3.3 WORKING ENVIRONMENT	13
3.4 DESIGN CONSTRAINTS	13
4. SYSTEM STUDY	15
4.1 OVERVIEW	15
4.2 PURPOSE	15
4.3 FUNCTIONAL REQUIREMENTS:	15
4.4 PYTHON LIBRARIES:	15
4.4.1 PANDAS	15
4.4.2 MATPLOTLIB	15
4.4.3 SEABORN	16
4.4.4 WARNINGS	16
4.4.5 OS	16
4.4.6 SKLEARN	16
4.4.7 COLLECTIONS	16
4.4.8 NUMPY	17
4.4.9 SYMPY	17
4.4.10 MATH	17
4.4.11 STATISTICS	17



4.4.12 SCIKIT	17
5. SYSTEM DESIGN	19
5.1 OVERVIEW	19
5.2 OVERALL ARCHITECTURE	19
5.3 MODULES	20
5.4 PROPOSED APPROACH	21
5.4.1 DATA COLLECTION	22
5.4.2 DATA PRE-PROCESSING	22
5.4.3 RANDOM DIRECTION GENERATION	22
5.4.4 PROJECTION OF DATA POINTS	23
5.4.5 CONSTRUCTION OF RPTREE	23
5.4.6 K NEAREST NEIGHBOR SEARCH	25
5.4.7 CONSTRUCTION OF RPFOREST	26
5.4.8 MAJORITY VOTING	27
6. IMPLEMENTATION METHODOLOGY	29
6.1 OVERVIEW	29
6.2 METHODOLOGY	29
7. CODING	36
8. EXPERIMENTAL RESULTS	48
8.1 OVERVIEW	48
8.2 SCREENSHOTS	48
9. EXPERIMENTAL COMPARISON	52
9.1 OVERVIEW	52
9.2 PERFORMANCE METRICS	52
10. CONCLUSION AND FUTURE WORK	56
10.1 CONCLUSION	56
10.2 FUTURE WORK	56
11. APPENDIX	58
11.1 REQUIREMENTS	58
11.2 WORKING ENVIRONMENT	58
11.2.1 HARDWARE REQUIREMENTS	58
11.2.2 SOFTWARE REQUIREMENTS	58
11.3 DATASET SPECIFICATION:	58
11.3.1. IMAGE SEGMENTATION DATASET	58
11.3.2. MUSK DATASET	59
11.3.3. HAR	59

11.4 PLAGIARISM REPORT	61
11.5 JOURNAL SUBMISSION PROOF	61
12. REFERENCES	63
12.1 WEB REFERENCES	63
12.2 REFERENCES	63

## **LIST OF FIGURES**

---

<b>FIGURE NO.</b>	<b>LIST OF FIGURES TOPIC</b>	<b>PAGE NO.</b>
4.1	System Architecture	27
4.2	Modules	28
4.3	Proposed Approach	29
5.1	Projection of Data points	33
5.2	Random Projection Tree	35
5.3	Arduino and Accelerometer Connection	37
7.1	Feature Importance	48
7.2	Random Data Selection	48
7.3	Random Direction Generation	49
7.4	Projected Data points	49
7.5	In order tree traversal	50
7.6	Input query and K value	50
7.7	Output	50
8.1	Performance Metrics	52
8.2	Human activity recognition - Accuracy	52
8.3	Comparison of training and testing accuracy with different datasets for $K = 5$	53

## **LIST OF ABBREVIATION**

---

## **LIST OF ABBREVIATIONS**

<b>S.No</b>	<b>ACRONYM</b>	<b>ABBREVIATION</b>
<b>1</b>	rpTree	Random Projection Tree
<b>2</b>	rpForest	Random Projection Forest
<b>3</b>	KNN	K Nearest Neighbour
<b>4</b>	HAR	Human Activity Recognition
<b>5</b>	USPS	The United States Postal Services

## **INTRODUCTION**

---

# CHAPTER 1

## 1. INTRODUCTION

Yesterday is Web 1.0. Today is Web 2.0. Tomorrow is Web 3.0. Web 2.0 mainly focused on the mobile, social, and cloud. While Web 3.0 is mainly focusing on Artificial Intelligence, decentralized data, and edge Computing. In Artificial Intelligence, Machine Learning plays a vital role. In Machine Learning the concept of K-Nearest Neighbour is applied in many places. K Nearest Neighbor (KNN) is a simple Machine learning algorithm utilized in an assortment of utilizations like money, medical care, handwriting recognition, image and video processing, and speech recognition. For Example, in the bank, we can check the credit score of the person and conclude whether to approve a credit card or not. In an email, we can classify whether the email is SPAM/HAM. KNN is a lazy learning algorithm. KNN is used for classification and regression. KNN algorithm is based on a feature-based similarity approach. KNN does not use any parameters as well as it doesn't make any assumptions, so it is popularly called a non-parametric algorithm. Next, we are going to use the concept of random projection forest which is a derived concept of a random forest algorithm. Ensemble learning is again part of machine learning. In ensemble learning, we combine many algorithms and then we take majority voting to conclude and find the result. It is unsupervised learning where we don't have a class label for classification. It is a process by which multiple models of the same kind or different kinds are trained separately and combined for more accurate results. Ensemble learning will improve the reliability and accuracy of prediction. We have many types of ensemble learning like bagging and boosting.

Random forest is a slightly modified version of bagging. It is better than decision trees but the accuracy is low while it removes the concept of overfitting. If we have more trees in the forest, accuracy is very much improved, but the time taken for the training is very long. Random Forest will select random observations and will build a decision tree and the result is taken by average or ranking or majority voting. Thus, there is no set of formulas for this. Random forest will give good results even if some of the data is missing and also it works well for large dimensional data.



Random projection forest is an upgraded version of the random forest where we use the concept of projection as an upgrade to it. We here use the concept of random projection. Thus, we achieve the ensemble methodology as well as tree-based methods. We construct the Ensemble of trees recursively on random projections. We Discover patterns that are useful for a variety of applications. Instead of optimizing concerning a metric in a random projection forest we randomly pick a split direction and then we randomly pick a split point along the direction. The split point is found where there is maximum stretched data (distance between data points). The contribution of our work is as follows; we combine KNN and random projection forest to reduce the running time complexity. we experiment on running time and also, we experiment on accuracy. we use Arduino and accelerometer sensor ADXL 335.

The Arduino UNO is the perfect board for beginners who want to learn about electronics and programming. The UNO is the most sturdy board you may start with if this is your first time working with the platform. The Arduino UNO is the most popular and well-documented board in the Arduino family.

The Arduino UNO is a microcontroller board that uses the ATmega328P microcontroller. There are 14 digital input/output pins (six of which can be used as PWM outputs), six analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button on the board. It comes with everything you'll need to get started with the microcontroller; simply plug it into a computer with a USB cable or power it with an AC-to-DC adapter or battery. You can experiment with your UNO without fear of making a mistake; in the worst-case situation, you can replace the chip for a few dollars and start over.

Arduino is a free and open-source hardware platform. An Atmel 8-bit AVR microcontroller with variable quantities of flash memory, pins, and functionality is used in most Arduino boards. The boot loader for Arduino microcontrollers is pre-programmed to make uploading programs to the on-chip flash memory easier. The Optiboot bootloader is the Arduino Uno's default bootloader. A serial connection to another computer is used to load software code onto the boards. Other Arduino variations, such as the unauthorized Boarduino, make use of a detachable USB-to-serial adaptor board or cable, Bluetooth, or other techniques. Instead of using the Arduino IDE, regular AVR in-system programming (ISP) programming is done with classic microcontroller tools.

An accelerometer is a device that measures acceleration forces using electromechanical means. Such forces might be static, such as gravity, or dynamic, such as in the case of mobile gadgets. The piezoelectric effect governs the operation of most accelerometer sensors. Because it is a low-power 3-axis sensor with a range of 3 g, the ADXL335 is used to measure dynamic accelerations such as motion, shock, and vibration, as well as static accelerations such as tilt or gravity, in the ADL series. The features of crystal deformation due by acceleration are exploited by the universal accelerometer sensor. Because this deformation generates voltage, the acceleration can be transformed into a voltage output as long as the relationship between the generated voltage and the applied acceleration is determined. A sensor that can measure acceleration is known as an accelerometer. Masses, dampers, elastic components, sensitive components, and adaptive circuits are common components.

The sensor obtains the acceleration value by utilizing Newton's second law and detecting the inertial force on the mass block during the acceleration operation. Common accelerometer sensors include capacitive, inductive, strain gauge, piezoresistive, piezoelectric, and so on, depending on the sensitive components of the sensor.

Depending on how the phone is tilted, the display rotates between landscape and portrait mode. Laptops have accelerometers that protect the hard drive from harm. If the laptop were to fall while in use, the accelerometer would detect the fall and turn off the hard drive instantly to prevent the reading heads from colliding with the platter. In cars, accelerometers are used to detect collisions and activate airbags as soon as possible.

Here we predict whether a person's orientation. This is achieved by training the reading taken from the accelerometer sensor. The reading will be taken from different positions and will be assigned with a class label as YES/No once the training is done, we will be running our algorithm to predict the person's orientation.



## CHAPTER 2

### 2. LITERATURE STUDY

#### 2.1 OVERVIEW

The existing methods used for predicting the class label are discussed here.

#### 2.2 COVER TREES FOR NEAREST NEIGHBOR [1]

**Authors:** Beygelzimer, A., Kakade, S., & Langford, J.(2006).

**Type:** ACM Press the 23rd international conference

**Methodology Used:**

The first step in the cover tree algorithm is finding the nearest neighbor. The cover tree keeps track of previous nodes. The next step is finding the approximating the nearest neighbor. Then we will insert and remove them based on our needs. The advantage is search efficiency is increased due to the introduction of tree data structure.

**Disadvantages:**

Accuracy is reduced because there is no forest and majority voting.

#### 2.3 RANDOMIZED PARTITION TREES FOR NEAREST NEIGHBOR SEARCH [2]

**Authors:** Dasgupta, S., & Sinha, K.(2015)

**Type:** A journal in Algorithmica.

**Methodology Used:**

There are two types: Randomized Spill Trees; Randomized Spill Trees. Here we do two splits and we use two data structures. We must split until we get at-most a single element in the leaf node. There are only two nodes inside each branch of the tree. The time complexity for nearest neighbour search query is  $O(n \log(n/n_0))$ . We have three splits namely Perturbed split, Overlapping split, and Median split.

**Disadvantages:**

Splitting must be done until the leaf node has at most one element.

#### 2.4 RANDOM-PROJECTION ENSEMBLE CLASSIFICATION [3]

**Authors:** Cannings, T.I., & Samworth, R.J.(2015).

**Type:** Journal of the Royal Statistical Society Series B.

**Methodology Used :**

Ensemble learning is again the part of machine learning. In ensemble learning, we combine many algorithms and then we take majority voting to conclude and find the result. It is unsupervised learning where we don't have a class label for classification. It is mainly used for the classification of high-dimensional data. The advantage is it is used for the classification of high-dimensional data

**Disadvantages:**

The disadvantage is it suffers interpretability and fails to determine the significance of each variable.

## 2.5 APPROXIMATE K-NEAREST NEIGHBOR BASED SPATIAL CLUSTERING USING K-D TREE [4]

**Authors:** Otair, M.(2013).

**Type:** International Journal of Database Management Systems.

**Methodology Used:**

The k-d tree was the first spatial data structure that was proposed for the nearest neighbor search. In high-dimensional spaces, its efficiency is reduced, but with randomization and using overlapping cells, it has been successful. A k-d tree, or k-dimensional tree, is a tree data structure used for organizing some number of points in a space with k dimensions. Spatial data has spatial characteristics and they are stored in spatial databases.

**Disadvantages:**

This method has No majority voting. Sometimes accuracy may be very low.

## 2.6 SMARTPHONE-BASED ACTIVITY RECOGNITION USING K-NEAREST NEIGHBOR ALGORITHM [5]

**Authors:** Mandong, A., & Munir, U.(2018)

**Type:** International Conference on Engineering Technologies.

**Methodology Used:**

Daily activities like walking, jogging, sitting, etc are recognized using the accelerometer sensor in the mobile phone. The accelerometer is typically a tri-axis sensor that detects the axis movements of humans. MATLAB is used for processing the data

gathered from the accelerometer sensor of the mobile phone. Then using WEKA, the data analysis software, the outcomes were analyzed. Here using the kNN algorithm with a k value of 3 an accuracy of 97.97% was achieved. A confusion matrix is used for showing the instances that are correctly and incorrectly classified.

**Disadvantages:**

This method consumes more power and resource.

## **SOFTWARE REQUIREMENT SPECIFICATION**

---

## CHAPTER 3

### 3. SOFTWARE REQUIREMENT SPECIFICATION

#### 3.1. INTRODUCTION

The purpose of this document is to give a detailed description of the requirements for “EFFICIENT HUMAN ACTIVITY RECOGNITION USING RANDOM PROJECTION FOREST” application. Using a random projection forest to improve accuracy. The accuracy of kNN searches is critical, and it must be improved in order to support Big Data. It combines the tree based methodology and power of ensemble learning.

##### 3.1.1 PURPOSE

The purpose of this project is to build an application that will predict the human activities. The prediction method must be efficient and the computational complexity must be low when compared to other existing methods. So, we use random projection forest and KNN algorithm. Also, to achieve better accuracy using random projection forest. kNN search accuracy is important and requires improvement to support big data. Random Projection Forest finds multiple kNN sensitive trees through random projection.

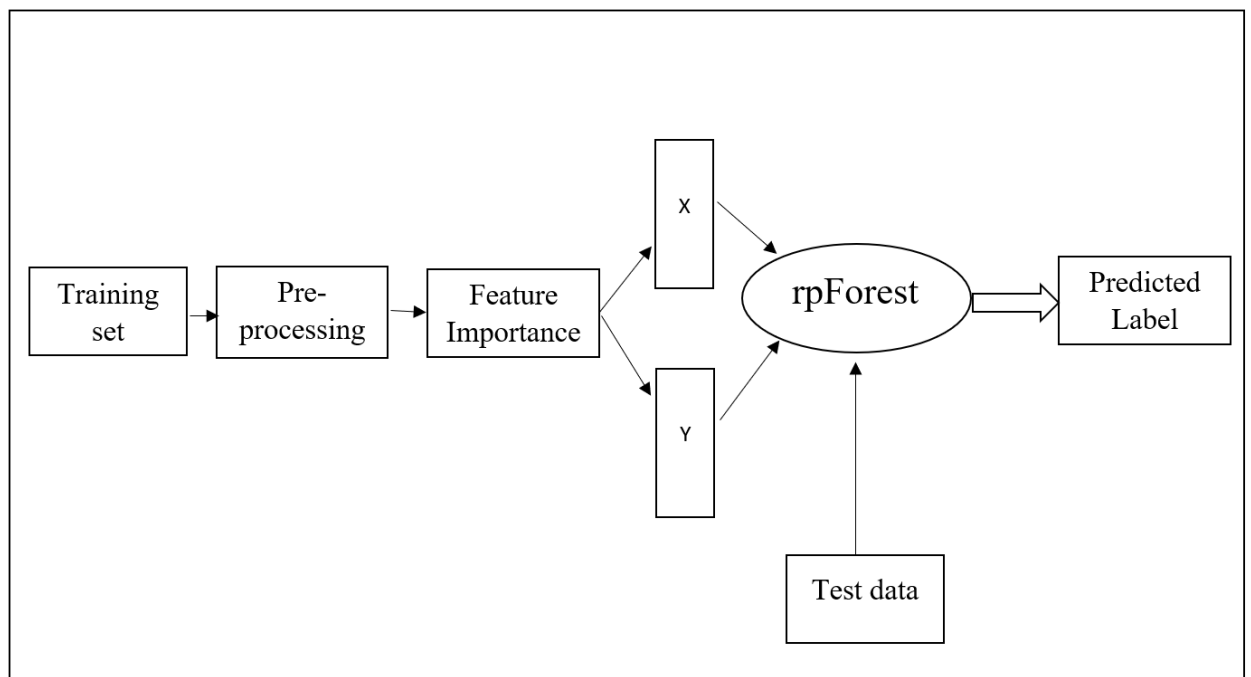
##### 3.1.2 OBJECTIVE

- To bring together the flexibility of tree-based technique with the power of ensemble methods.
- To develop a probability theory for nearby points separated by ensemble rpTrees.
- To assist in the selection of a random projection.

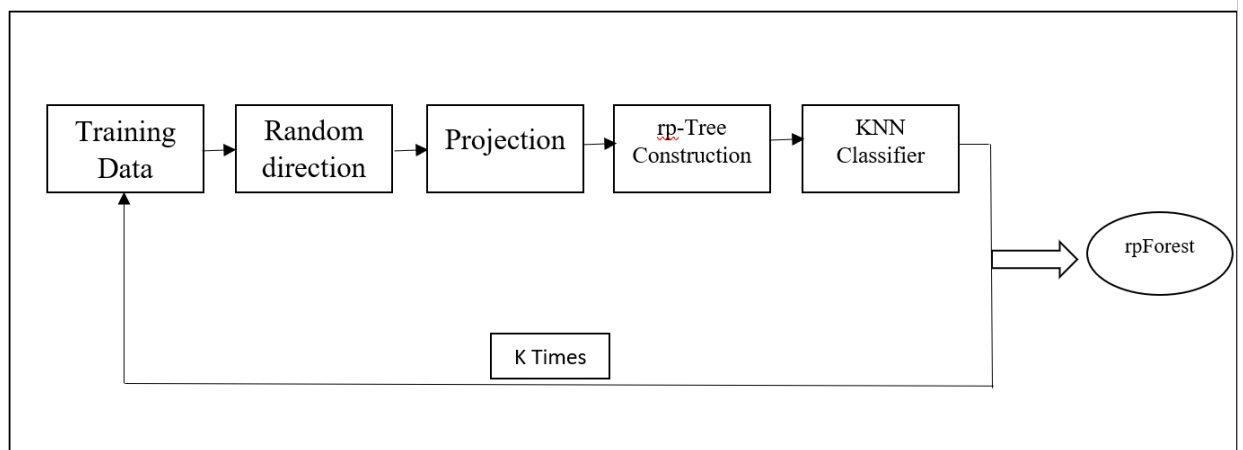
#### 3.2. OVERVIEW

Using an accelerometer sensor, we were able to recognize human activities efficiently. For various positions such as standing, sitting, walking upstairs, and walking downstairs, data is captured using an accelerometer sensor connected to an Arduino. The gathered data is then pre-processed. The features that change frequently are identified and are considered to be the most important qualities.





Two of these characteristics are identified. The data points should then be projected. After we've finished projecting, we'll start building the rpTree and use the k-nearest neighbor search algorithm to identify labels for the query points. We mix many rpTrees in rpForest Construction. Then we run for k times in a row, and we receive rpForests.



Finally, we use the majority voting method. We mix many rpTrees in rpForest Construction. We must project data points in order to construct the rpTree. The notion of orthogonal projection is used to project the data point. We create a new orthogonal line equation for each data point. We use  $m-1=1/m$  to calculate the slope of an orthogonal line. The intercept value is then calculated using the data point's x and y values. This is repeated for each data point. The projection is finished once all of the points have been completed. We place all of

the anticipated data points in the tree's root node. We determine the projection's maximum and minimum points. We get a random value by using the random.uniform function. The random range for this function will be between the minimum and maximum point x values. For b point, the lowest and maximum y values will be the same. This is where the line splits. This point can also be obtained using the median function. This point has now been passed, and the root node is split using it. The values in the root node that are less than this point will move to the left subtree, while those that are larger than this point will go to the right. This will be repeated until the leaf node has reached the desired number of data points. This is also determined at random by us.

### 3.3 SPECIFIC REQUIREMENTS

This section explains the requirements of Efficient Human Activity Recognition by Random Projection Forest application.

#### 3.3.1 FUNCTIONAL REQUIREMENTS

- Dataset must be pre-processed before training.
- Feature extraction methods like feature importance must be used.
- Only features that are extracted must be inserted into the tree.
- Theta value must be in between 90 degrees and 360 degrees.
- Theta value must be generated randomly.
- The ns value is predefined constant.
- The number of points(n) to be selected is generated randomly and n number of data points is selected from the dataset randomly.
- The number of points to be selected from dataset must be greater than ns.
- Using random value for a and b which is minimum and maximum of the data points, we calculate the median and choose it as splitting point, c.
- The points whose value is less than the splitting point goes to the left child of the rpTree and those greater go to the right child of the rpTree.

#### 3.3.2 NON-FUNCTION REQUIREMENTS

Non-functional requirements specify desired characteristics of the system to be designed and often have a greater impact on the system architecture than functional requirements.

- App tools: Python language is used for coding.
- Used Arduino and Accelerometer ADXL335 sensor for taking input dataset reading
- Must run for K number of times.
- GPU must be present.
- Input query point must lie inside the dataset. (ie) not far away from the dataset.

### 3.3.3 WORKING ENVIRONMENT

#### 3.3.3.1 HARDWARE REQUIREMENTS

Processor	AMD Ryzen 5 / Intel Core i5
Hard Disk	512 GB
RAM	8GB
Operating System	Windows 10 and Higher

#### 3.3.3.2 SOFTWARE REQUIREMENTS

- Python 3
- Accelerometer ADXL335
- Arduino IDE

### 3.4 DESIGN CONSTRAINTS

- Fast and accurate prediction.



## CHAPTER 4

### 4. SYSTEM STUDY

#### 4.1 OVERVIEW

System study involves studying the project's purpose and identifying the main functional requirements needed for the project and system requirements configurations needed for the project.

#### 4.2 PURPOSE

The purpose of this project is to achieve better accuracy using random projection forest. kNN search accuracy is important and requires improvement to support Big data. Random Projection forest finds multiple kNN sensitive trees through random projection.

#### 4.3 FUNCTIONAL REQUIREMENTS:

Python version 3 is required. Python Libraries including NumPy, Matplotlib, Pandas, random, sympy, collections, math, statistics, Scikit-Learn are required.

#### 4.4 PYTHON LIBRARIES:

##### 4.4.1 PANDAS

Pandas is one of the most widely used Python packages for dealing with large data sets. It has functions for analyzing, cleaning, exploring and manipulating data. Pandas is an open-source data analysis and manipulation tool that is quick, powerful, flexible, and simple to use.

##### 4.4.2 MATPLOTLIB

MATPLOTLIB is a Python data visualisation package that allows us to produce figures and plots, allowing us to easily create static raster or vector files without the use of any GUIs. Matplotlib is a free, open-source Python toolkit for plotting graphs. It is a graphical representation of the data. Scatter plots, bar graphs, histograms, area plots, and pie plots are examples of graphical forms. One of the most widely used Python packages for

data visualisation is Matplotlib. It's a cross-platform library that generates 2D charts from array data.

#### 4.4.3 SEABORN

Seaborn is a Python module for creating statistical visuals. It is based on matplotlib and is tightly connected with pandas data structures. Here are some of the features that seaborn has to offer:

#### 4.4.4 WARNINGS

Warnings are given to developers to alert them to circumstances that aren't necessarily exceptions. A warning is usually issued when particular programming pieces, such as a keyword, function, or class, have become obsolete.

#### 4.4.5 OS

In Python, the OS module has functions for dealing with the operating system. Python's standard utility modules include OS. This module allows you to use operating system-dependent functions on the go. Many functions to interface with the file system are included in the os and os.path modules.

#### 4.4.6 SKLEARN

It is a Python package with several supervised and unsupervised learning techniques. It is entirely made up of libraries such as pandas, Matplotlib, and NumPy. Regression, classification, clustering, model selection, and pre-processing are some of the features it offers.

#### 4.4.7 COLLECTIONS

Different types of containers are available in Python's collection module. A Container is a type of object that may be used to store several objects and provide a mechanism to access and iterate over them. Tuple, List, Dictionary, and other built-in containers are examples

#### 4.4.8 NUMPY

NumPy (numerical Python) is a library that consists of multidimensional array objects and a set of functions for manipulating those arrays. It is one of the most important Python packages for scientific computing. It allows you to conduct mathematical and logical operations on arrays, making it one of the most used Python tools for scientific computing. Random number generators are also included.

#### 4.4.9 SYMPY

SymPy is a symbolic mathematics Python package. Its goal is to develop into a full-featured computer algebra system (CAS) while keeping the code as basic as possible to make it comprehensible and extendable. SymPy is totally built in Python and does not rely on any third-party libraries.

#### 4.4.10 MATH

Python has a math module that can handle similar calculations. Basic operations such as addition(+), subtraction(-), multiplication(\*), and division(/) are covered by the Math module, as well as advanced operations such as trigonometric, logarithmic, and exponential functions.

#### 4.4.11 STATISTICS

Statistics is the method of collecting, tabulating, and interpreting numerical data in general. Python's statistics is a built-in descriptive statistics library. If your datasets aren't too large or you can't rely on importing other libraries, you can utilise it. This module contains functions for calculating numeric (Real-valued) data mathematical statistics.

#### 4.4.12 SCIKIT

In Python, Scikit-learn (Sklearn) is the most usable and robust machine learning library. It uses a Python consistency interface to give a set of efficient tools for machine learning and statistical modelling, such as classification, regression, clustering, and dimensionality reduction. NumPy, SciPy, and Matplotlib are the foundations of this package, which is mostly written in Python. Machine learning models are built with sklearn.





## CHAPTER 5

### 5. SYSTEM DESIGN

#### 5.1 OVERVIEW

This section gives an overview of the project. There are totally 7 steps involved in high level of this project.

#### 5.2 OVERALL ARCHITECTURE

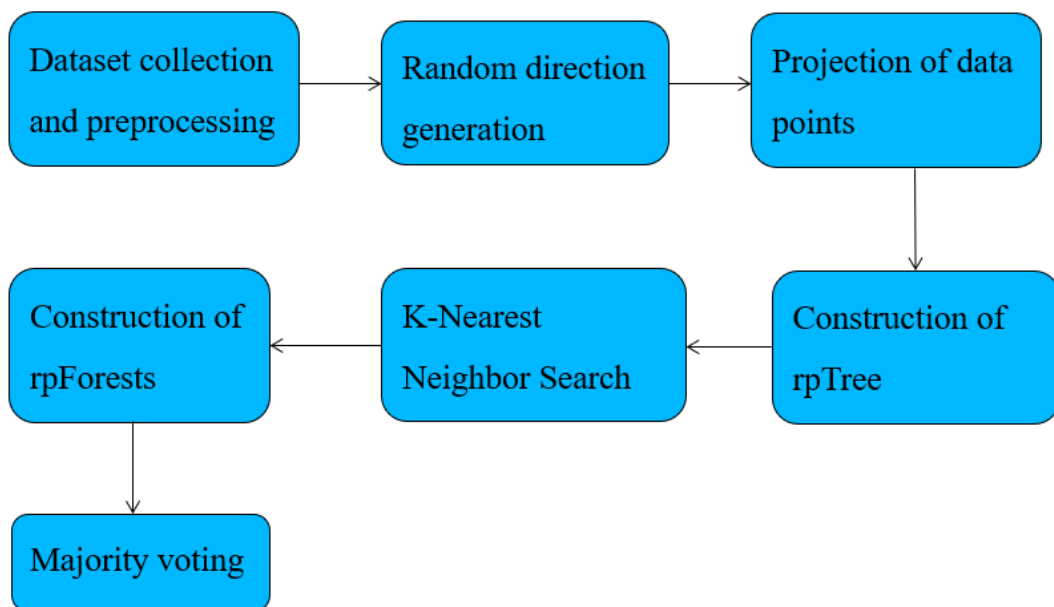


Figure 4.1 : System Architecture

The figure 4.1 depicts a high-level overview of the system architecture of the Efficient human activity recognition using random projection forest. The data is collected using accelerometer sensor using Arduino for various position such as standing, sitting, walking upstairs and walking downstairs. Then we pre-process the collected data. The features which change frequently are identified and they are the feature important feature. We identify two of such features. Then project the data points. Once projection is done, we start constructing the rpTree and then run the k- nearest neighbor search algorithm for the query points and find the label for them. Then we run continuously for k times and hence we get rpForests. Then finally we take the majority voting.

### 5.3 MODULES

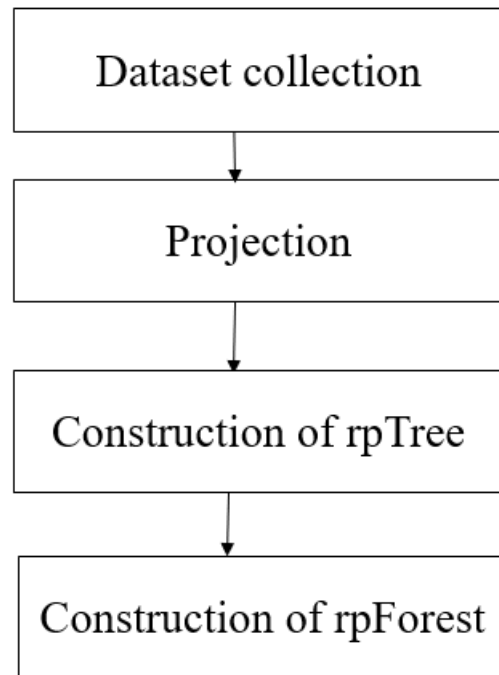


Figure 4.2 : Modules

The figure 4.2 depicts the modules or the workflow of the Efficient human activity recognition using random projection forest. The data is collected using accelerometer sensor using Arduino for various position such as standing, sitting, walking upstairs and walking downstairs. The data needs to be cleaned before any processing can be done with that data. The features which change frequently are identified and they are the feature important feature. Then the data is projected. For projection use orthogonal projection. Next, we start constructing the random projection tree. We run this multiple times and get random projection forest.

## 5.4 PROPOSED APPROACH

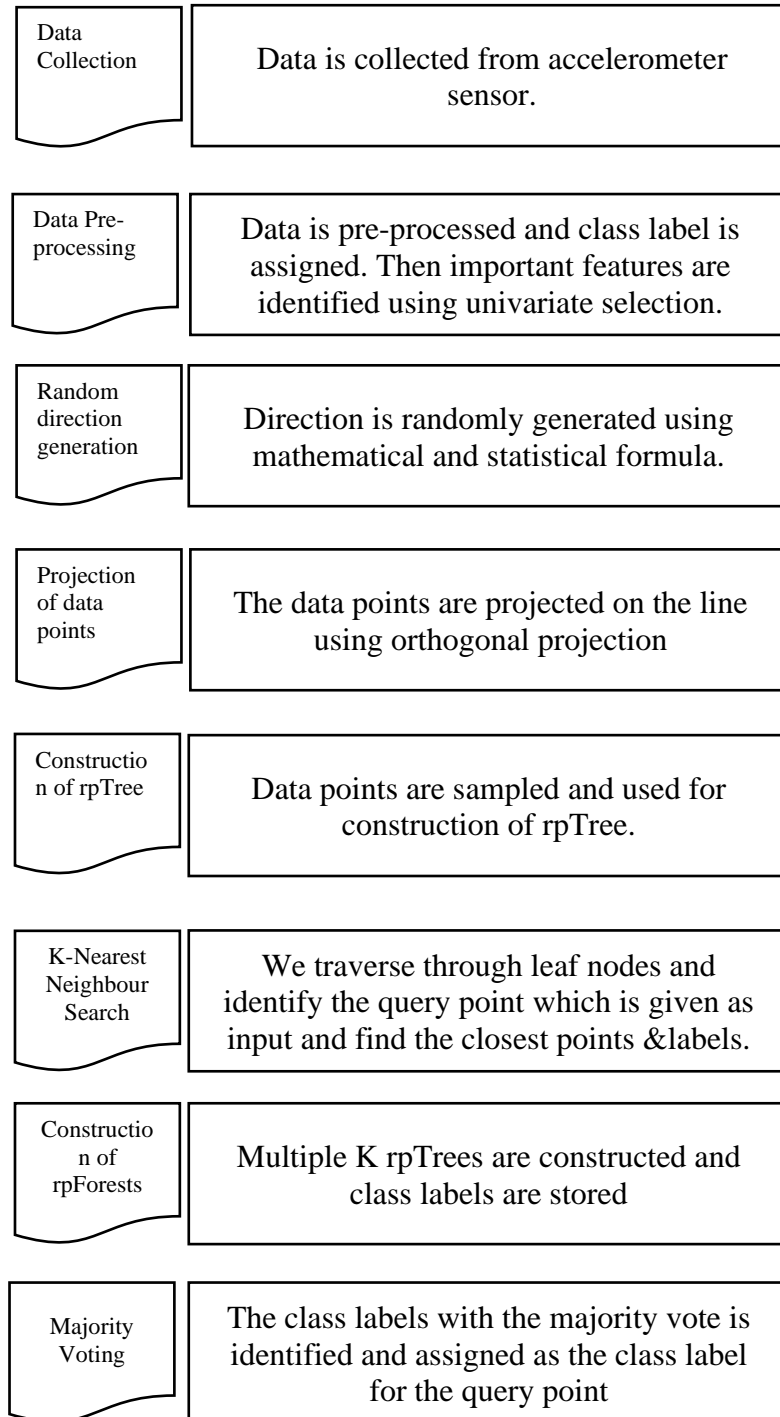


Figure 4.3: Proposed Approach

#### 5.4.1 DATA COLLECTION

The Efficient human activity recognition using random projection forest proposed approach has 7 steps involved. The first step is data collection. We here gather our own dataset using the hardware components which are Arduino UNO and accelerometer ADXL335 sensor. This gives three axes namely X, Y, Z as readings. We here use PuTTY, an open-source terminal emulator for storing the continuous data readings of the sensor in an excel. We now keep the sensor in one position, say Sitting. We store the output of this sensor which are readings in a csv file. Then we assign class label as “SITTING”. Similarly, we do the process for the other activity such as STANDING, WALKING UPSTAIRS, WALKING DOWNSTAIRS. Other than this we downloaded dataset from UCI Machine Learning repository. The datasets are Musk, USPS, GAS, Gisette, Image Segmentation, Arcene, Parkinson, Poker, Sensor, Smartphone, Wisconsin breast cancer. The first step data collection is over.

#### 5.4.2 DATA PRE-PROCESSING

Collected data may contain missing values or inconsistent. So, we need to pre-process the data. Also, the collected data may have many features. We can't use all the features. So, we need to use the concept of feature importance. For our collected data set we used feature importance and found out that Y and Z axis are the most important feature. Thus, we use the values of Y and Z axis only.

#### 5.4.3 RANDOM DIRECTION GENERATION

Once data pre-processing is done, we next need to take data points from the dataset randomly. We fix a constant  $n_s$ . We need to randomly generate a value, say  $n$ . We need to make sure that the randomly generated value is greater than  $n_s$  value. If not, we need to generate the random number again. If the condition is satisfied then we randomly select  $n$  indices and then take values of the indices from the dataset. Once this is done, we now have the randomly selected  $n$  data points. Next, generating random direction is the task. Direction means angle. So, generate random value ranging from 90 degree to 360 degree. Once this value is generated, we next need to find the slope ( $m$ ). We have the formula  $m = \tan \theta$ . Then we use median of selected data points as  $x$  and  $y$ . Using this we generate a random line equation.

#### 5.4.4 PROJECTION OF DATA POINTS

When a trained model is run on new data, it can produce training errors. It begins to produce incorrect outcomes. By the way, there's one logical assumption here: your training set won't contain the same training samples from separate courses, resulting in contradicting data. This characteristic may be present in some real-world datasets.

Random Projection is another decomposition technique for lowering the dimensionality of large datasets. These tactics have received a lot of attention for their robustness, simplicity, and unexpected effectiveness in achieving extremely low mistake rates. This is in contrast to a number of other decomposition approaches, including singular value decomposition, which we discussed before. The Johnson-Lindenstrauss lemma underpins the development of this approach as well as the majority of its uses.

Orthogonal projection methods use a different approach to lowering data dimensionality than transformation or variable selection methods. These methods have the advantage of more precisely identifying the dimensions of the subspace that describe the maximum number of variations in the multivariate space that are linked or unrelated to the item of interest,  $y$ . As a result, orthogonal projection methods have the advantage of making the regression model independent of the influence of variations in the data that aren't related to  $y$ , allowing for improved extraction of the most relevant information (as illustrated in the examples above). As a result, orthogonal projection methods improve data and model interpretability. It also indicates that the model's prediction ability improves outside of the calibration range and in the presence of numerous influencing factors that may impact the spectral signal.

To project the data point we use the concept of orthogonal projection. For each data points we construct new orthogonal line equation. For orthogonal line we use the slope using  $m^{-1}=1/m$ . Then we get the intercept value by using the  $x$  and  $y$  value of the data point. This is done for all the data points. Once done for all points the projection is completed.

#### 5.4.5 CONSTRUCTION OF RPTREE

##### K-D Tree

Because of the huge space and time complexity, the K-D Tree was designed to reduce the space and time complexity. A binary search tree in which the data at each node is a K-Dimensional point in space is known as a K-D Tree. In a nutshell, it's a data structure for arranging points in a K-Dimensional space that uses space partitioning. The  $\log_2 n$  depth of

the k-d tree is guaranteed, where  $n$  is the number of points in the set. Traditionally, k-d trees have been used to store points in  $d$ -dimensional space that are vectors in  $d$ -dimensional space.

#### Steps in K-D Tree

1. Choose an X-axis and project each point onto it, then compute the median and divide the data by it.
2. Then choose a Y-axis and project each point onto it, then compute the median and divide the data by it.
3. Build a tree by repeating the steps above and switching the axes.
4. In a K-D Tree, a non-leaf node divides the space into two halves, known as half-spaces.
5. The left subtree of that node represents points to the left of this space, whereas the right subtree represents points to the right of the space.

If there is only one point, use it to make a leaf. Otherwise, a line perpendicular to one of the axes might be used to divide the points in half. Construct k-d trees for both sets of points in a recursive manner. Points perpendicular to the axis with the greatest spread are divided.

#### Find the Nearest Neighbor using KD Tree

To discover the point in the same cell as the query, search recursively. On the way back, look through each subtree to see if there is a point that is closer to the one you already know about. Keep the closest point to the query point found variable. When the bounding boxes of subtrees state they can't contain any point closer than the query point, prune them. To increase the chances of pruning, search the subtrees.

#### Time Complexity of k-D Tree

In a suitable model, the K-D Tree takes  $O(\log n)$  average time per search. (Assume  $d$  is a tiny number.) The k-d tree has  $O$  storage ( $n$ ). If  $d$  is small, the pre-processing time is  $O(n \log n)$ .

## Limitations of k-D Tree

When  $d$  is not a small number, time complexity skyrockets. Only data that is consistently distributed and has a small dimension works well with the K-D Tree. However, most real-world data is not distributed evenly. The K-D Tree was created primarily for information retrieval and computer graphics, not for KNN.

Next step after projection is construction of the random projection tree. We insert all the projected data points into the root node of the tree. We find the maximum and minimum point of projection. Using `random.uniform` function we get a random value. The random range for this function will be minimum point  $x$  value and maximum  $x$ . Similarly for  $y$  point it will be minimum point  $y$  value and maximum  $y$ . This is the splitting point. We can also get this point using median function too. This is now passed and the root node is split using this point. The values in root node lesser than this point will be going to the left subtree and the greater values than this point will be going to the right side. This will be done recursively till the leaf node has specified number of data points. This is also specified by us randomly.

### 5.4.6 K NEAREST NEIGHBOR SEARCH

The K-Nearest Neighbour algorithm is based on the Supervised Learning technique and is one of the most basic Machine Learning algorithms. The KNN method assumes that the new case/data and existing cases are similar and places the new case in the category that is most similar to the existing categories. The K-NN method stores all available data and classifies a new data point based on its similarity to the existing data. This means that new data can be quickly sorted into a well-defined category using the K-NN method. The K-NN approach can be used for both regression and classification, but it is more commonly utilized for classification tasks. The KNN algorithm is a non-parametric algorithm, which means it makes no assumptions about the underlying data.

It's also known as a lazy learner algorithm since it doesn't learn from the training set straight away; instead, it saves the dataset and uses it to classify it later. The KNN method merely saves the information during the training phase, and when it receives new data, it classifies it into a category that is quite similar to the new data. KNN may be used to tackle both classification and regression forecasting problems. However, it is more typically used in the industry when classification problems arise.

Steps involved:

Step 1: Decide on the number of neighbours (K).

Step 2: Determine the Euclidean distance between K neighbours.

Step 3: Using the obtained Euclidean distance, find the K closest neighbours.

Step 4: Count the number of data points in each category among these k neighbours.

Step 5: Assign the new data points to the category with the greatest number of neighbours.

The rpTree is constructed and next we need to find the K-Nearest Neighbor. For this we need to traverse through the leaf node and find the query point. For the query point we find the KNN. We find the minimum distance point and take that point's class label and store it.

#### 5.4.7 CONSTRUCTION OF RPFORST

Random forest is a supervised machine learning technique for classification and regression issues. It uses the majority vote for classification and the average for regression to build decision trees from various samples.

There are two types of approaches used by Ensemble:

1. Bagging– It creates a new training subset with replacement from sample training data, and the final output is determined by majority voting.

Consider Random Forest, for example.

2. Boosting– It helps weak students become powerful. It's commonly accomplished by creating high-accuracy sequential models.

ADA BOOST and XG BOOST, for example, are two such examples.

#### Bagging

Bootstrap Aggregation, often known as Bagging, is a random forest ensemble method. Bagging takes a random sample of data from the complete set and puts it in a bag. As a result, row sampling is used to substitute the samples (Bootstrap Samples) provided by the Original Data in each model. Row sampling with replacement is referred to as the "bootstrap" step. The findings are generated after each model has been trained independently. The ultimate result is determined by majority voting once all of the models' findings have been combined. Aggregation is the process of combining all of the data and producing a result based on a majority vote.



The steps in the random forest method are as follows:

Step 1: Random forest selects  $n$  random records from a data set of  $k$  records at random.

Step 2: Individual decision trees are created for each sample.

Step 3: At the end of each decision tree, there is a result.

Step 4: The final result for classification and regression is based on Majority Voting or Averaging, as appropriate.

In conclusion, Random forest is a slightly modified version of bagging. It is better than decision trees but the accuracy is low while it removes the concept of overfitting. If we have more trees in the forest, accuracy is very much improved, but the time taken for the training is very long. Random Forest will select random observations and will build a decision tree and the result is taken by average or ranking or majority voting. Thus, there is no set of formulas for this. Random forest will give good results even if some of the data is missing and also it works well for large dimensional data.

Random projection forest is an upgraded version of the random forest where we use the concept of projection as an upgrade to it. We here use the concept of random projection. Thus, we achieve the ensemble methodology as well as tree-based methods. We construct the Ensemble of trees recursively on random projections. We Discover patterns that are useful for a variety of applications. Instead of optimizing concerning a metric in a random projection forest we randomly pick a split direction and then we randomly pick a split point along the direction. The split point is found where there is maximum stretched data (distance between data points). The contribution of our work is as follows; we combine KNN and random projection forest to reduce the running time complexity. we experiment on running time and also, we experiment on accuracy. we use Arduino and accelerometer sensor ADXL 335.

rpForest means multiple rpTree. Thus, we run the rpTree loop for  $n$  number of time and store the output.

#### 5.4.8 MAJORITY VOTING

Finally, after the rpForests construction is over we need to take majority voting. We then arrive the solution (ie) the class label for the query point

## **IMPLEMENTATION METHODOLOGY**

---

## CHAPTER 6

### 6. IMPLEMENTATION METHODOLOGY

#### 6.1 OVERVIEW

The implementation methodology of the project describes the techniques used in our project.

#### 6.2 METHODOLOGY

Our proposed method here is use combine the random projection Tree (rpTree) and K- Nearest neighbour (kNN). Firstly rpTree is a slightly modified and randomized version of kd-Tree. Using tree based methodology in data mining application is very common since it is very easy and it complexity is very low. The tree is constructed by splitting the data set into two randomly. Then we recursively apply it to the two nodes. It is continued till we are left with few data points. The selected split will be along random direction ( $\vec{r}$ ). If a point  $x$  which is a data point in dataset  $W$ , its projection is

$$\frac{r.(x)}{|r|^2} (\vec{r})$$

where  $.$  is dot product. It defines the projection coefficient along random direction ( $\vec{r}$ ). If the projection coefficient is denoted by  $c$ , then left node is given by  $WL = \{ x \in W : r.x < c \}$ . The right node is given by  $WR = \{ x \in W : r.x \geq c \}$ .

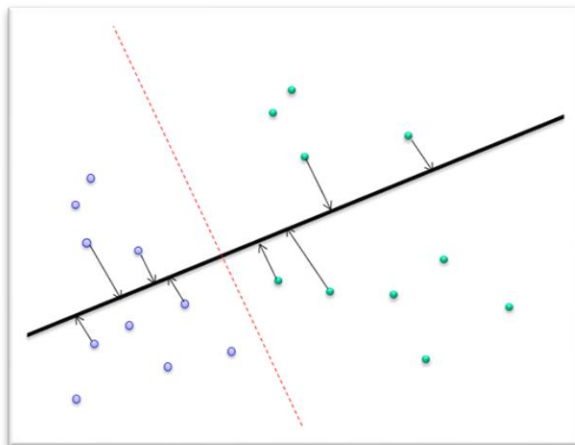


Figure 4.3: The Projection of the data points to the randomly generated line and split using the splitting point, the left and right child of the root node is assigned.

---

**Algorithm 1 - rpTree(U)**

---

```
1: Let U be the root node of tree t;  
2: Initialize the set of working nodes  $W \leftarrow \{U\}$ ;  
3: while W is not empty do  
4:   Randomly pick  $W \in W$  and set  $W \leftarrow W - \{W\}$ ;  
5:   if  $|W| < ns$  then  
6:     Skip to the next round of the while loop;  
7:   end if  
8:   Generate a random direction  $\vec{r}$   
9:   Project points in W onto  $\vec{r}$ ,  $W\vec{r} = \{x: x \in W\}$ ;  
10:  Let  $a = \min(W\vec{r})$  and  $b = \max(W\vec{r})$ ;  
11:  Generate a splitting point  $c \sim \text{runif}[a, b]$ ;  
12:  Split node W by  $WL = \{x : p\vec{r}(x < c)\}$  and  $WR = \{x : P\vec{r}(x \geq c)\}$ ;  
13:   $W.\text{left} \leftarrow WL$  and  $W.\text{right} \leftarrow WR$ ;  
14:  Update the working set by  $W \leftarrow W \cup \{WL, WR\}$ ;  
15: end while  
16: return(t);
```

---

The data set is denoted by U and t denotes the rpTree which is to be built from dataset U. Let working nodes be denoted by W. ns denotes a constant for the minimal number of data points in a tree for which we will split further.  $P\vec{r}$  denotes the projection coefficient of data point x onto line  $\vec{r}$ . Let neighborhoods be denoted by N.

The Efficient human activity recognition using random projection forest proposed approach has 7 steps involved. The first step is data collection. We here gather our own dataset using the hardware components which are Arduino UNO and accelerometer ADXL335 sensor. This gives three axes namely X, Y, Z as readings. We here use PuTTY, an open-source terminal emulator for storing the continuous data readings of the sensor in an excel. We now keep the sensor in one position, say Sitting. We store the output of this sensor which are readings in a csv file. Then we assign class label as “SITTING”. Similarly, we do the process for the other activity such as STANDING, WALKING UPSTAIRS, WALKING DOWNSTAIRS. Other than this we downloaded dataset from UCI Machine

Learning repository. The datasets are Musk, USPS, GAS, Gisette, Image Segmentation, Arcene, Parkinson, Poker, Sensor, Smartphone, Wisconsin breast cancer.

Collected data may contain missing values or inconsistent. So, we need to pre-process the data. Also, the collected data may have many features. We can't use all the features. So, we need to use the concept of feature importance. For our collected data set we used feature importance and found out that Y and Z axis are the most important feature. Thus, we use the values of Y and Z axis only. Once data pre-processing is done, we next need to take data points from the dataset randomly. We fix a constant  $n_s$ . We need to randomly generate a value, say  $n$ . We need to make sure that the randomly generated value is greater than  $n_s$  value. If not, we need to generate the random number again. If the condition is satisfied then we randomly select  $n$  indices and then take values of the indices from the dataset. Once this is done, we now have the randomly selected  $n$  data points. Next, generating random direction is the task. Direction means angle. So, generate random value ranging from 90 degree to 360 degree. Once this value is generated, we next need to find the slope ( $m$ ). We have the formula  $m = \tan \theta$ . Then we use median of selected data points as  $x$  and  $y$ . Using this we generate a random line equation. To project the data point we use the concept of orthogonal projection. For each data points we construct new orthogonal line equation. For orthogonal line we use the slope using  $m^{-1} = 1/m$ . Then we get the intercept value by using the  $x$  and  $y$  value of the data point. This is done for all the data points. Once done for all points the projection is completed.

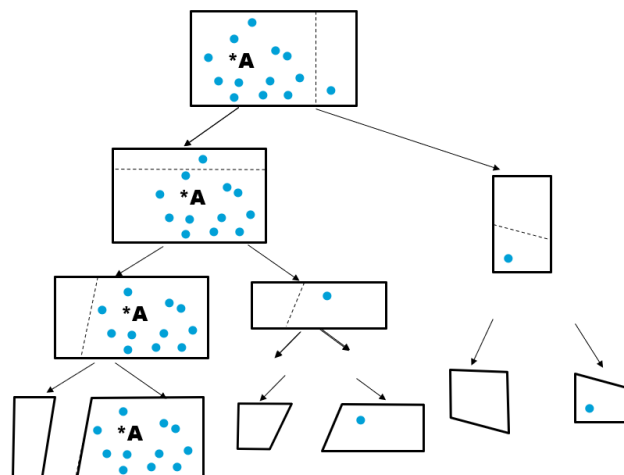


Figure 4.3: Block diagram of the rpTree with data points and query point

In the rpTree we initialize, the root node which consists of all the data points. Let's consider data point A, and all its kNNs are marked as blue. The root node is split and it causes one of the kNN of point A to lie at right child node. Further growth of tree will make other additional kNN of A to be separated as shown. Finally, most of its kNNs and point A would land in the same leaf node, with an exception of a few of its data points of kNNs in different leaf nodes which causes error in kNN search. So, to eliminate this problem our algorithm combines the power of ensemble methods with use of rpTree which will reduce the mis-match in kNN search. For single rpTree we will have one leaf node. Thus, it grows individually and is independent. Now taking union will be eliminating the error.

We insert all the projected data points into the root node of the tree. We find the maximum and minimum point of projection. Using random.uniform function we get a random value. The random range for this function will be minimum point x value and maximum x. Similarly for b point it will be minimum point y value and maximum y. This is the splitting point. We can also get this point using median function too. This is now passed and the root node is split using this point. The values in root node lesser than this point will be going to the left subtree and the greater values than this point will be going to the right side. This will be done recursively till the leaf node has specified number of data points. This is also specified by us randomly.

Second algorithm is combining k-nearest neighbors using the ensemble of rpTrees. Let  $Q \subseteq U$  be the set of data points for which we use k-nearest neighbors in U. Q can be U itself. rpTree resembles random forest and hence we call the algorithm as Random Projection Forest (rpForests). rpForest finds multiple kNN sensitive trees through random projection. The objective of the random projection forest is as follows; To combine the flexibility of the tree-based methodology and power of ensemble methods; To develop a theory on probability of neighboring points separated by ensemble rpTrees; To guide the choice of random projection.

The rpTree is constructed and next we need to find the K-Nearest Neighbor. For this we need to traverse through the leaf node and find the query point. For the query point we find the KNN. We find the minimum distance point and take that point's class label and store it.

rpForest means multiple rpTree. Thus, we run the rpTree loop for n number of time and store the output.

---

**Algorithm 2 - kNNrpForests(Q, U)**

---

```
1: for i = 1 to T do
2:     Build the i-th rp Tree by  $t_i \leftarrow \text{rpTree}(U)$ ;
3: end for
4: for each data point  $q \in Q$  do
5:     for i = 1 to T do
6:         Let q fall through  $t_i$  and arrive at leaf node  $N_i$ ;
7:     end for
8:     Set the union of neighbor sets of q by  $N_q \leftarrow \bigcup_{i=1}^T N_i$  ;
9:     Compute distance between q and each point in  $N_q$ ;
10:    The K points in  $N_q$  with smallest distance to q are its kNNs;
11: end for
```

---

Thus, the overall overflow is Construct rpTree then Use probability of neighbor point to separate rpTree and finally Combine kNN and rpTree to construct kNN-rpForest Algorithm. We now implement this into one of application which is Human activity recognition. This is done by the use of accelerometer sensor. We here use Arduino UNO and the accelerometer sensor and take the readings for different actions(motions) such as sitting, standing, walking upstairs and downstairs, etc. We assign class label to it and hence we have created our own dataset.

---

**Algorithm 3 - readData from sensor**

---

```
1: initialize the three data pins of accelerometer sensor.
2: Keep the accelerometer sensor in a horizontal position.
3: Read value from the xpin, ypin, and zpin.
3: Now repeat the step 3 for different positions.
```

Now in real-time we can use accelerometer sensor and detect or predict what the human is doing.

---

Arduino is a free and open-source hardware platform. An Atmel 8-bit AVR microcontroller with variable quantities of flash memory, pins, and functionality is used in most Arduino boards. The boot loader for Arduino microcontrollers is pre-programmed to make uploading programs to the on-chip flash memory easier. The Optiboot bootloader is the Arduino Uno's default bootloader.

An accelerometer is a device that measures acceleration forces using electromechanical means. Such forces might be static, such as gravity, or dynamic, such as in the case of mobile gadgets. The piezoelectric effect governs the operation of most accelerometer sensors. Because it is a low-power 3-axis sensor with a range of 3 g, the ADXL335 is used to measure dynamic accelerations such as motion, shock, and vibration, as well as static accelerations such as tilt or gravity, in the ADL series. The features of crystal deformation due by acceleration are exploited by the universal accelerometer sensor. Because this deformation generates voltage, the acceleration can be transformed into a voltage output as long as the relationship between the generated voltage and the applied acceleration is determined. A sensor that can measure acceleration is known as an accelerometer. Masses, dampers, elastic components, sensitive components, and adaptive circuits are common components.

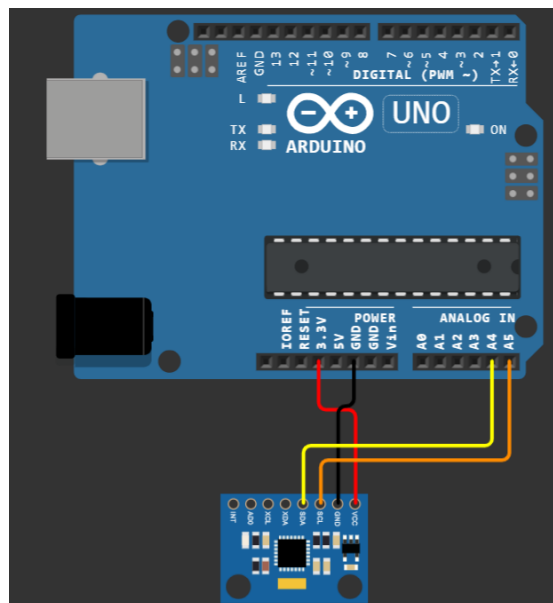


Fig.3. The simulation of the working of the accelerometer sensor using Arduino.





## CHAPTER 7

### 7. CODING

**# importing libraries**

import numpy as np

import pandas as pd

import random

import math

import statistics

from sympy import symbols, Eq, solve

from matplotlib import pyplot as plt

from collections import defaultdict

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

dataset = pd.read\_csv("OwnDataSet.csv")

dataset.head()

%store dataset

# dataset.shape

data = dataset.values[:]

data.shape

**# Feature Importance**

from sklearn.ensemble import ExtraTreesClassifier

#data = pd.read\_csv("usps.csv")

X = dataset.iloc[:,1:] #independent columns

y = dataset.iloc[:,0] #target column i.e price range

model = ExtraTreesClassifier()

model.fit(X,y)

#print(model.feature\_importances\_) #use inbuilt class feature\_importances of tree based classifiers

#plot graph of feature importances for better visualization

```

feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
fs = feat_importances.nlargest(2) # pandas.series
fd = fs.to_frame() # series to dataframe
dx = fd.index[0] # important feature 1
dy = fd.index[1] # important feature 2
print(dx,dy)
df = pd.DataFrame(dataset)
xindex = df.columns.get_loc(dx)
yindex = df.columns.get_loc(dy)
if xindex > yindex:
    temp = xindex
    xindex = yindex
    yindex = temp
print(xindex,yindex)
df.shape
xdata = data[:,xindex] # fetch values of a particular column in the dataset
ydata = data[:,yindex]
#print(xdata)
xdatalist = xdata.tolist() # convert np array to a list
ydatalist = ydata.tolist()
xlis = [[i] for i in xdatalist] # convert elemets in list to list
ylis = [[i] for i in ydatalist]
xnpparray = np.array(xlis)
ynpparray = np.array(ylis)
dset = np.concatenate((xnpparray,ynpparray),axis = 1)
dlis = dset.tolist() # dataset to be given as a input to algorithm as a root node
#dset
def solute(m,ms,c,bs):
    a, b = symbols('a,b')
    eq1 = Eq((a-(b/m)), (-c/m))
    # print("Equation 1:")

```

```

eq2 = Eq((a-(b/ms)), (-bs/ms))
answer=solve((eq1, eq2), (a, b))
return list(answer.values())

#Defining rpTree
class Node:
    leaf = 2
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.val)
        if self.right:
            self.right.PrintTree()

    def insert(self, p):
        l=[]
        r=[]
        for i in self.val:
            if i<p:
                l.append(i)
            else:
                r.append(i)
        le=Node(l)
        ri=Node(r)
        self.left=le
        self.right=ri
        lmid=np.median(le.val,axis=0).tolist()
        rmid=np.median(ri.val,axis=0).tolist()
        Tree1=self
        if(len(Tree1.left.val))>self.leaf:

```

```

        Tree1 = Tree1.left
        Tree1.insert(np.median(Tree1.val,axis=0).tolist())
    if(len(Tree1.right.val))>self.leaf:
        Tree1 = Tree1.right
        Tree1.insert(np.median(Tree1.val,axis=0).tolist())
    Tree1=self
    if(len(Tree1.right.val))>self.leaf:
        Tree1 = Tree1.right
        Tree1.insert(np.median(Tree1.val,axis=0).tolist())
    if(len(Tree1.left.val))>self.leaf:
        Tree1 = Tree1.left
        Tree1.insert(np.median(Tree1.val,axis=0).tolist())
def printLeafNodes(root: Node) -> None:
    if (not root):
        return
    if (not root.left and
        not root.right):
        l.append(root.val)
        return
    if root.left:
        printLeafNodes(root.left)
    if root.right:
        printLeafNodes(root.right)
knn=defaultdict(int)
query = list(map(int, input("Enter the query point: ").split()))
k = int(input("K :"))
for i in range(k):
    Root = Node(dset)
    ns = 5          # predefined constant for the minimal number of data points in the node
    drows = dset.shape[0]
    print("Total rows in the dataset :",drows)      # Total rows in the dataset

    projected_notProjected_dictionary={ }

```

```

while drows!=0:
    rand_no = random.randint(0,10)
    # Random indices in the datase
    random_indices = np.random.choice(drows, size=rand_no, replace=False)
    if len(random_indices) < ns:
        continue
    else:
        print("index :",random_indices)
        print("Length :",len(random_indices))
        root_node = dset[random_indices, :] # Random elements in the dataset
        print("Root node :",root_node.tolist())

# Random Direction
maxcol = np.max(root_node,axis=0)
mincol = np.min(root_node,axis=0)
print("Maximum in column :",maxcol)
print("Minimum in column :",mincol)
origin = []
x1 = maxcol[0]
y1 = maxcol[1]
x2 = mincol[0]
y2 = mincol[1]
origin.append((x1+x2)//2)
origin.append((y1+y2)//2)
print("Origin Point :",origin)
#y = mx +c
theta = random.randint(180,270) # Generate a random degree
print("Theta :",theta)
m = math.tan(math.radians(theta)) # Convert degree to angle
#yinter = m * xinter + c
m = round(m,2) # Rounding to two decimals
#print(m)

```

```

c = origin[1] - m * origin[0]
c = round(c,2)
print("c :",c) #6.928
print("Random Line Equation : " + "Y = " + str(m) + " X + " + str(c))
# m,c = random_direction(root_node)
ms = -1/m
ms = round(ms,2)
projlis = []
lix=[]
liy=[]
for i in root_node:
    ls = i
    bs = ls[1] - ms * ls[0]
    bs = round(bs,2)
    # solve
    #print("Orthogonal Line Equation : " + "Y = " + str(ms) + " X + " + str(bs))
    li = solute(m,ms,c,bs)
    lix.append(round(li[0],2))
    liy.append(round(li[1],2))
    rounded_list=[]
    rounded_list.append(round(li[0],2))
    rounded_list.append(round(li[1],2))
    projected_notProjected_dictionary[str(i.tolist())]=(rounded_list)
    projlis.append(rounded_list)
print("Projected List :",projlis)
maxx = np.max(projlis,axis=0)
minn = np.min(projlis,axis=0)
print("Maximum projected point :",maxx)
print("Minimum projected point :",minn)
# amin = np.max(projlis)
# bmin = np.min(projlis)
# resx = statistics.median(lix)      # Splitting using median of the projected lists
# resy = statistics.median(liy)

```

```

splitx=round(random.uniform(round(minn[0],2),round(maxx[0],2)),2)
print("splitx :",splitx) #x
splits=round(random.uniform(round(minn[1],2),round(maxx[1],2)),2)
print("splits :",splits) #y
split_points = splitx,splits
split_points = list(split_points)
%%store split_points
Tree = Node(projlis)
# mid=np.median(projlis,axis=0).tolist()
Tree.insert(split_points)
print("splitting point random :",split_points)
print("Tree inorder traversal :")
Tree.PrintTree()
l=[]
printLeafNodes(Tree)
dis={}
sub = []
for i in l:
    ll=[]
    for j in i:
        x,y=(j[0]-query[0]),(j[1]-query[1])
        sublist=[]
        sublist.append(x)
        sublist.append(y)
        ll.append(sublist)
        dis[str(j)]=sublist
    sub.append(ll)
#     print(ll)
print("Distance list",sub)
print("minimum",min(min(sub)))
print(dis)
result=[]
for key, value in dis.items():

```



```

        if value == min(min(sub)):
            result=(key)
    print("Result",result)

    final_output=[]
    final=[]
    print(projected_notProjected_dictionary)
    for key, value in projected_notProjected_dictionary.items():
        if str(value) == result:
            print("inside validation")
            final_output=list(key)
            final_output.remove('[')
            final_output.remove(']')
            joining_final_op=".".join(final_output)
            final=joining_final_op.split(",")
            final[0]=int(final[0])
            final[1]=int(final[1])
            print("final",final)
            datafr = df.loc[(df[dy]==final[0]) & (df[dx]==final[1])]
            al = datafr["LABEL"].where(((datafr[dy]==final[0]) &
(datafr[dx]==final[1]))).tolist()
            output = (list(set(al)))
            print("output",output)
            knn[str(output)]+=1
        break
    knn
    max(knn,key=knn.get)
    xaxis = [[i[0]] for i in root_node] # convert elemets in list to list
    yaxis = [[i[1]] for i in root_node]
    print(root_node)
    print(xaxis)
    print(yaxis)
    graph_set = df.loc[df["LABEL"]=="SITTING"]

```

```

al = df["LABEL"],df["Y"],df["Z"].where(df["LABEL"]=="SITTING")
sit = df.loc[(df["LABEL"]=="SITTING")]
sit.pop('X')
stand = df.loc[(df["LABEL"]=="STANDING")]
stand.pop('X')
stand
# graph_set
print(sit)
xsit = sit['Y'].tolist()
ysit = sit['Z'].tolist()
xstand = stand['Y'].tolist()
ystand = stand['Z'].tolist()
plt.scatter(xsit,ysit, c='b',marker="D")
plt.scatter(query[0],query[1],c='g',marker="*") #334,256
plt.scatter(xstand,ystand, c='r',marker="+")
plt.title("K-Nearest neighbor")
plt.xlabel('Horizontal Position')
plt.ylabel('Vertical Position')
plt.legend(["Sitting","Query","Standing"])
plt.show()
X_train = pd.read_csv("HAR/X_train.csv")
X_test = pd.read_csv("HAR/X_test.csv")
y_train = pd.read_csv("HAR/y_train.csv")
y_test = pd.read_csv("HAR/y_test.csv")
X_train = X_train.iloc[:,:].values
X_test = X_test.iloc[:,:].values
y_train = y_train.iloc[:,:].values
y_test = y_test.iloc[:,:].values
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=5)

```

```

classifier.fit(X_train, y_train.ravel())
y_pred = classifier.predict(X_test)
print("Training accuracy :",round(classifier.score(X_train,y_train),4))
print("Testing accuracy :",round(classifier.score(X_test,y_test),4))
classifier = KNeighborsClassifier(n_neighbors=10)
classifier.fit(X_train, y_train.ravel())
y_pred = classifier.predict(X_test)
print("Training accuracy :",round(classifier.score(X_train,y_train),4))
print("Testing accuracy :",round(classifier.score(X_test,y_test),4))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train.ravel())
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.figure(figsize=(12, 7))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')

acc = []
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(X_train,y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',

```

```

        marker='o',markerfacecolor='red', markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))
X = ['OwnDataset','USPS','Musk','HAR','Segmentation']
#k=5
train = [99.07,97.63,93.68,98.29,88.10]
test = [96.3,96.24,87.5,88.36,85.71]
X_axis = np.arange(len(X))
plt.bar(X_axis - 0.2, train, 0.4, label = 'Train')
plt.bar(X_axis + 0.2, test, 0.4, label = 'Test')
plt.ylim(ymax=150,ymin=50)
plt.xticks(X_axis, X)
plt.xlabel("Datasets")
plt.ylabel("Accuracy")
plt.title("Training and Testing accuracy for K = 5")
plt.legend()
plt.show()
X = ['OwnDataset','USPS','Musk','HAR','Segmentation']
#k = 10
train = [95.3,96.45,90.26,97.66,88.10]
test = [92.59,95.75,84.38,89.28,83.33]
X_axis = np.arange(len(X))
plt.bar(X_axis - 0.2, train, 0.4, label = 'Train')
plt.bar(X_axis + 0.2, test, 0.4, label = 'Test')
plt.ylim(ymax=150,ymin=50)
plt.xticks(X_axis, X)
plt.xlabel("Datasets")
plt.ylabel("Accuracy")
plt.title("Training and Testing accuracy for K = 10")
plt.legend()
plt.show()

```

## **EXPERIMENTAL RESULTS**

---

## CHAPTER 8

### 8. EXPERIMENTAL RESULTS

#### 8.1 OVERVIEW

This chapter details our project results and for each step screenshots of output are attached and explained.

#### 8.2 SCREENSHOTS

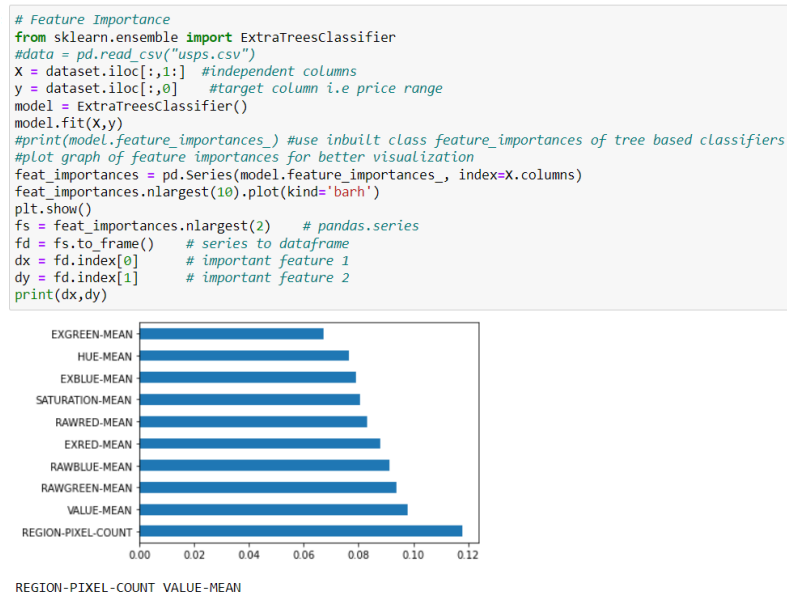


Figure 7.1 : Feature Importance

In Figure 7.1, The input dataset has many features. So we need to pre-process it before using. For that, the two important features of data is taken.

```
while draws!=0:
    rand_no = random.randint(0,10)
    random_indices = np.random.choice(draws, size=rand_no, replace=False) # Random indices in the dataset
    if len(random_indices) < ns:
        continue
    else:
        print("index :",random_indices)
        print("Length :",len(random_indices))
        root_node = dset[random_indices, :] # Random elements in the dataset
        print("Root node :",root_node.tolist())
    break
```

index : [36 29 14 19 0]  
Length : 5  
Root node : [[260, 333], [260, 334], [325, 268], [333, 268], [335, 269]]

Figure 7.2 : Random Data Selection

In Figure 7.2, We randomly choose a number between 0 to 10 and keep it as rand\_no. Then we choose rand\_no of random indices. Finally we fetch those indecies data points from the dataset.

```

: #y = mx + c
theta = random.randint(180,360)    # Generate a random degree
print("Theta :",theta)
m = math.tan(math.radians(theta))   # Convert degree to angle
#yinter = m * xinter + c
m = round(m,2) # Rounding to two decimals
#print(m)
c = origin[1] - m * origin[0]
c = round(c,2)
print("c :",c) #6.928
print("Random Line Equation : " + "Y = " + str(m) + " X + " + str(c))

# tan(248) = 2.47 = m
# 298 - (2.47)331

Theta : 217
c : 78.5
Random Line Equation : Y = 0.75 X + 78.5

```

Figure 7.3 : Random Direction Generation

In Figure 7.3, We randomly generate a theta value and then we find the slope. Using this slope and the origin point we find the intercept value. We now got the random line equation.

```

ms = -1/m
ms = round(ms,2)
projlis = []
for i in root_node:
    ls = i
    bs = ls[1] - ms * ls[0]
    bs = round(bs,2)
    # solve
    li = solute(m,ms,c,bs)
    proj=[]
    proj.append(round(li[0],2))
    proj.append(round(li[1],2))
    projlis.append(proj)
# print("ls :",ls)
# print("xp,yp :",xp,yp)
# print("li :",li)
# print("len(li) :",len(li))
print("projeccted List :",projlis)
# print("len(projlis) :",len(projlis))
# %store projlis

projeccted List : [[288.61, 294.95], [289.09, 295.31], [298.92, 302.69], [304.03, 306.53], [305.79, 307.84]]

```

Figure 7.4 : Projected Data points

In Figure 7.4, We find the orthogonal line for each data point to the random line which was generated earlier. This completes the projection

```

root = projlis
Tree = Node(root)
import numpy as np
mid=np.median(root,axis=0).tolist()
#print(mid)
#Tree.insert(mid)
Tree.insert(split_points)
Tree.PrintTree()

[[288.61, 294.95], [289.09, 295.31]]
[[288.61, 294.95], [289.09, 295.31], [298.92, 302.69], [304.03, 306.53], [305.79, 307.84]]
[[298.92, 302.69]]
[[298.92, 302.69], [304.03, 306.53], [305.79, 307.84]]
[[304.03, 306.53], [305.79, 307.84]]

```

Figure 7.5 : In order tree traversal

In Figure 7.5 we traverse through the rpTree and display in ascending order or in tree terms we use inorder traversal.

```

query = list(map(int, input("Enter the query point: ").split()))
# print(query)
k = int(input("K :"))

```

```

Enter the query point: 334 256
K :5

```

Figure 7.6 : Input query and K value

In Figure 7.6, We get the input query point and K value which is number of rpTree.

```

: 1 max(knn,key=knn.get)
: 'Sitting'

```

Figure 7.7 : Output

In Figure 7.7 the output is displayed by taking majority voting.



## **EXPERIMENTAL COMPARISON**

---

## CHAPTER 9

### 9. EXPERIMENTAL COMPARISON

#### 9.1 OVERVIEW

This section compares our proposed project system with the existing research results. The next section compares the F1, Recall and Precision of human activity recognition.

#### 9.2 PERFORMANCE METRICS

	precision	recall	f1-score	support
LAYING	0.99	0.96	0.97	537
SITTING	0.89	0.82	0.86	491
STANDING	0.84	0.93	0.88	532
WALKING	0.82	0.98	0.89	496
WALKING_DOWNSTAIRS	0.98	0.75	0.85	420
WALKING_UPSTAIRS	0.89	0.89	0.89	471
accuracy			0.89	2947
macro avg	0.90	0.89	0.89	2947
weighted avg	0.90	0.89	0.89	2947

Figure 8.1 Shows the classification report

Fig 8.1 shows classification report of the human activity recognition using a smartphone dataset. Using the classification report we can understand many terms such as *f1-score*, *precision*, *recall*, and *support*. *Precision* measures how many positive predictions made are correct. *Recall* measures how many positive predictions were made overall positive cases in data. It is also called sensitivity. The *f1-score* is the mean of precision and recall. These indicate the accuracy of classifying data points in one class compared to other classes. The *support* is the number of data points of the true response that belongs to that class.

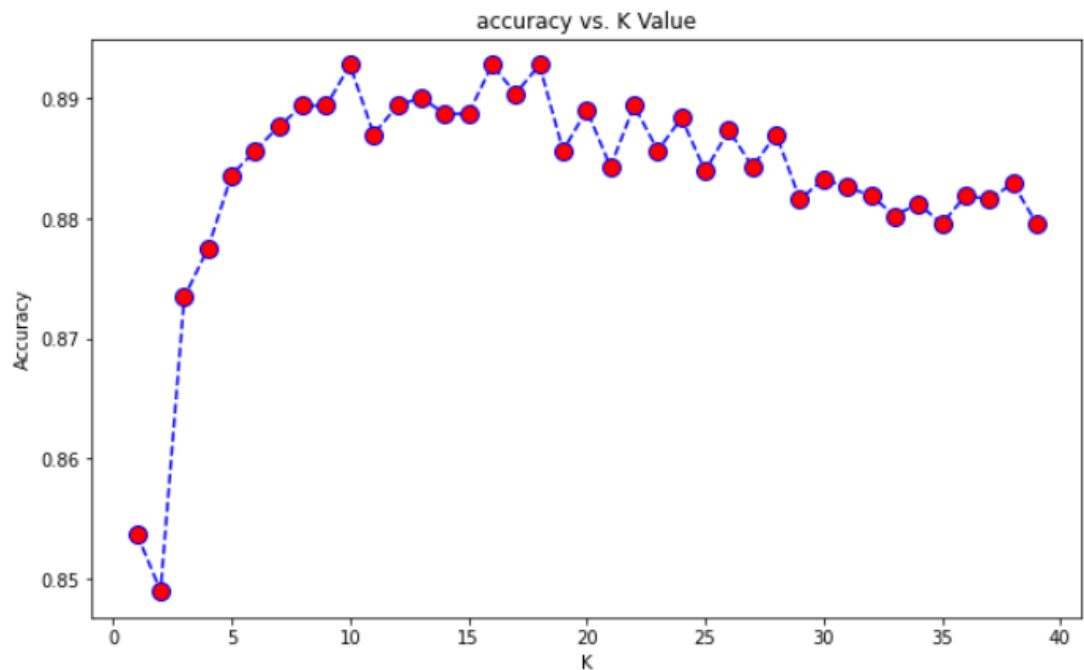


Figure 8.2 Human activity recognition – Accuracy

The accuracy decreases at the first, but eventually when the value of K increases, the accuracy starts to rise, and at  $K = 9$ , the classification model receives the maximum accuracy of 89.27%.

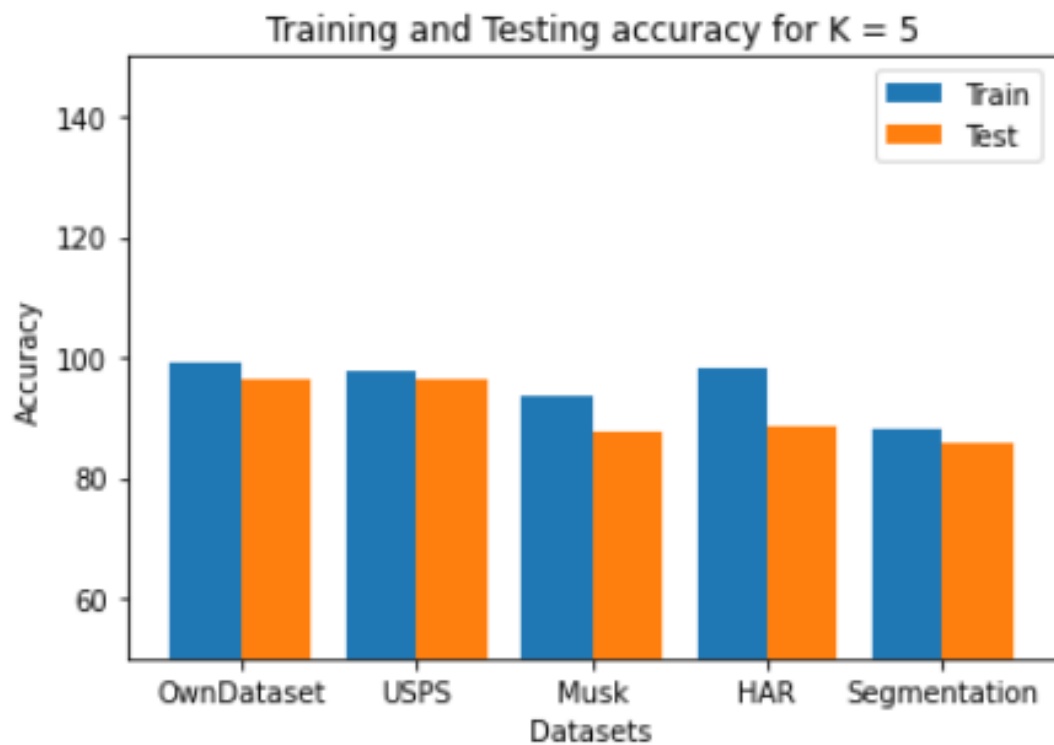


Figure 8.3 Comparison of training and testing accuracy with different datasets for  $K = 5$

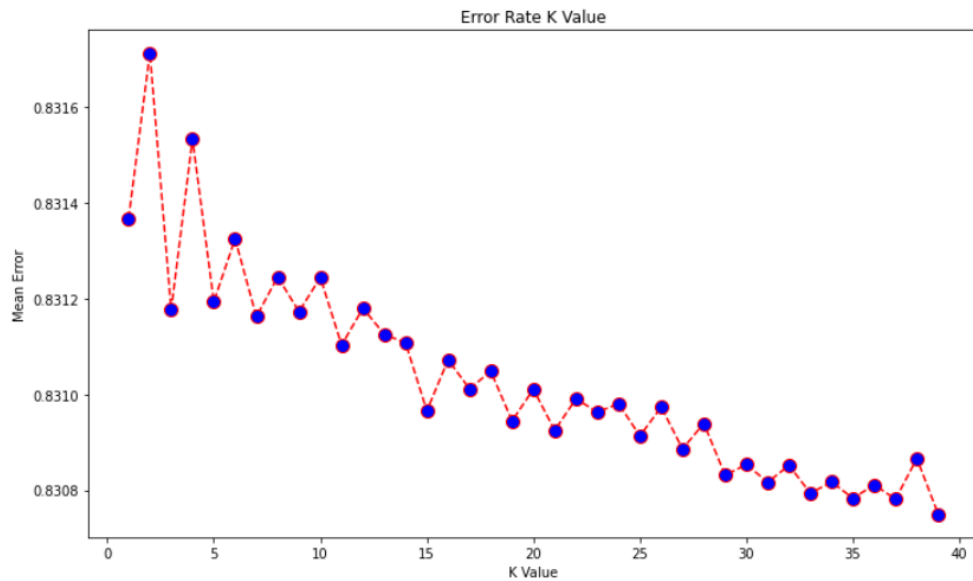


Figure 8.4 The line plot depicts the variation of error rate and K-value for the HAR dataset.

## Training error

When a trained model is run on new data, it can produce training errors. It begins to produce incorrect outcomes. By the way, there's one logical assumption here: your training set won't contain the same training samples from separate courses, resulting in contradicting data. This characteristic may be present in some real-world datasets.

## Cross-validation error

The problem arises when Cross-validation is used to select the best  $K$ . When the train error is low and the validation error is high, we have an overfitting problem, as indicated in the diagram above. When train error and validation error are both high, we have a problem called underfitting, as indicated in the diagram above. Our model was chosen because several train and validation mistakes are close to each other, as illustrated in the above graphic. Fig 8.4 shows how the mean error rate varies when the  $k$  value is changed. When the  $k$  value increases the mean error rate decreases. Thus, we need to set the  $k$  value high to decrease the error.

## **CONCLUSION AND FUTURE WORK**

---

## CHAPTER 10

### 10. CONCLUSION AND FUTURE WORK

#### 10.1 CONCLUSION

rpForests is another tree-based algorithm that uses the combination of both tree-based methodologies as well as ensemble learning. The rpForests receives the maximum accuracy even for the high dimensional datasets with data's ranging from small to n-dimension data when using more trees. This algorithm can be used for other predications too.

#### 10.2 FUTURE WORK

Our future work includes developing a mobile application that would alert the person who is using it if the mobile phone is about to fall. The app will first give a warning if there is a 50% possibility of the mobile fall. Then will give a continuous alert sound if there is a possibility of fall lies greater than 85%.

## **APPENDIX**

---

# CHAPTER 11

## 11. APPENDIX

### 11.1 REQUIREMENTS

This chapter explains our project requirements in detail and the modules which we used to run the experimentation.

### 11.2 WORKING ENVIRONMENT

#### 11.2.1 HARDWARE REQUIREMENTS

Processor	AMD Ryzen 5 / Intel Core i5
Hard Disk	512 GB
RAM	8GB
Operating System	Windows 10 and Higher

#### 11.2.2 SOFTWARE REQUIREMENTS

- Python 3
- Accelerometer ADXL335
- Arduino IDE

### 11.3 DATASET SPECIFICATION:

#### 11.3.1. IMAGE SEGMENTATION DATASET

Title: Image Segmentation data

Description: The instances were drawn randomly from a database of 7 outdoor images. The images were hand segmented to create a classification for every pixel.

Number of Instances: Training data: 210 Test data: 2100

Number of Attributes: 19 continuous attributes

Missing Attribute Values: None

Classes: brick face, sky, foliage, cement, window, path, grass.

30 instances per class for training data.



300 instances per class for test data.

### 11.3.2. MUSK DATASET

Title: MUSK database

Description:

This dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks. The goal is to learn to predict whether new molecules will be musks or non-musks. However, the 166 features that describe these molecules depend upon the exact shape, or conformation, of the molecule. Because bonds can rotate, a single molecule can adopt many different shapes. To generate this data set, the low-energy conformations of the molecules were generated and then filtered to remove highly similar conformations. This left 476 conformations. Then, a feature vector was extracted that describes each conformation. This many-to-one relationship between feature vectors and molecules is called the "multiple instance problem". When learning a classifier for this data, the classifier should classify a molecule as "musk" if ANY of its conformations is classified as a musk. A molecule should be classified as "non-musk" if NONE of its conformations is classified as a musk.

Number of Attributes: 169.

Number of Instances: 476

Missing Attribute Values: none.

Class Distribution:

Musks: 47

Non-musks: 45

### 11.3.3. HAR

Title: Human Activity Recognition Using Smartphones Dataset

Description:

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING,

WALKING\_UPSTAIRS, WALKING\_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers were selected for generating the training data and 30% for the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low-frequency components, therefore a filter with a 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

Number of Attributes: 561.

Number of Instances:

X\_train : 7352

X\_test : 2947

Y\_test : 2947


Y\_train : 7352

Missing Attribute Values: none.

Classes :

WALKING, WALKING\_UPSTAIRS, WALKING\_DOWNSTAIRS, SITTING, STANDING, LAYING.

## 11.4 PLAGIARISM REPORT



---

**Document Information**





---

<b>Analyzed document</b>	Final Year Project Report draft.pdf (D133819551)
<b>Submitted</b>	2022-04-17T20:51:00.0000000
<b>Submitted by</b>	
<b>Submitter email</b>	venittaraj@mepcoeng.ac.in
<b>Similarity</b>	3%
<b>Analysis address</b>	venittaraj.mepco@analysis.orkund.com

---


**Sources included in the report**

---

	URL: <a href="https://www.its203.com/article/kelly_fumiao/106458251">https://www.its203.com/article/kelly_fumiao/106458251</a> Fetched: 2022-04-17T20:52:37.0630000	 2
	URL: <a href="https://www.uj5u.com/qita/118343.html">https://www.uj5u.com/qita/118343.html</a> Fetched: 2022-04-17T20:52:34.4600000	 3

## 11.5 JOURNAL SUBMISSION PROOF

# Submission Confirmation



---

Thank you for your submission

---

**Submitted to**  
IET Computer Vision

**Manuscript ID**  
CVI-2022-04-0149

**Title**  
EFFICIENT HUMAN ACTIVITY RECOGNITION USING RANDOM PROJECTION FOREST

**Authors**  
R, Venitta Raj  
KUMAR T J, AUMRUDH LAL  
Kumaar M, Sanjai

**Date Submitted**  
19-Apr-2022

## REFERENCES

---

## CHAPTER 12

### 12. REFERENCES

#### 12.1 WEB REFERENCES

- [http://www.math.umassd.edu/~dyan/Slides\\_rpForests.pdf](http://www.math.umassd.edu/~dyan/Slides_rpForests.pdf)
- <https://www.youtube.com/watch?v=HbySATqmGnw>
- <https://towardsdatascience.com/random-forest-in-python-24d0893d51c0>
- <https://ccrma.stanford.edu/~jos/st/Projection.html>
- <https://math.stackexchange.com/questions/62633/orthogonal-projection-of-a-point-onto-a-line>

#### 12.2 REFERENCES

- [1] Yan, D., Wang, Y., Wang, J., Wang, H., & Li, Z.(2021)." K-nearest Neighbors Search by Random Projection Forests", in IEEE Transactions on Big Data.
- [2] Mandong, A., & Munir, U.(2018)." Smartphone-Based Activity Recognition using K-Nearest Neighbor Algorithm", in International Conference on Engineering Technologies.
- [3] Beygelzimer, A., Kakade, S., & Langford, J.(2006)." Cover Trees for Nearest Neighbor", ACM Press the 23rd international conference.
- [4] Otair, M.(2013). "Approximate K-Nearest Neighbour Based Spatial Clustering Using K-D Tree", in International Journal of Database Management Systems.
- [5] Andoni, A., & Indyk, P.(2008)." Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions", in Communication of the ACM.
- [6] Dasgupta, S., & Sinha, K.(2015)."Randomized Partition Trees for Nearest Neighbor Search", a journal in Algorithmica.
- [7] Cannings, T.I., & Samworth, R.J.(2015)." Random-projection ensemble classification", Journal of the Royal Statistical Society Series B.
- [8] Nussey, J.(2013).[Book]."Arduino for Dummies".
- [9] Starlino. (2009)."A Guide to using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications", retrieved from:  
[http://www.starlino.com/imu\\_guide.html](http://www.starlino.com/imu_guide.html).
- [10] Arduino,"ADXL3xx Accelerometer", retrieved from:  
<https://docs.arduino.cc/built-in-examples/sensors/ADXL3xx>.