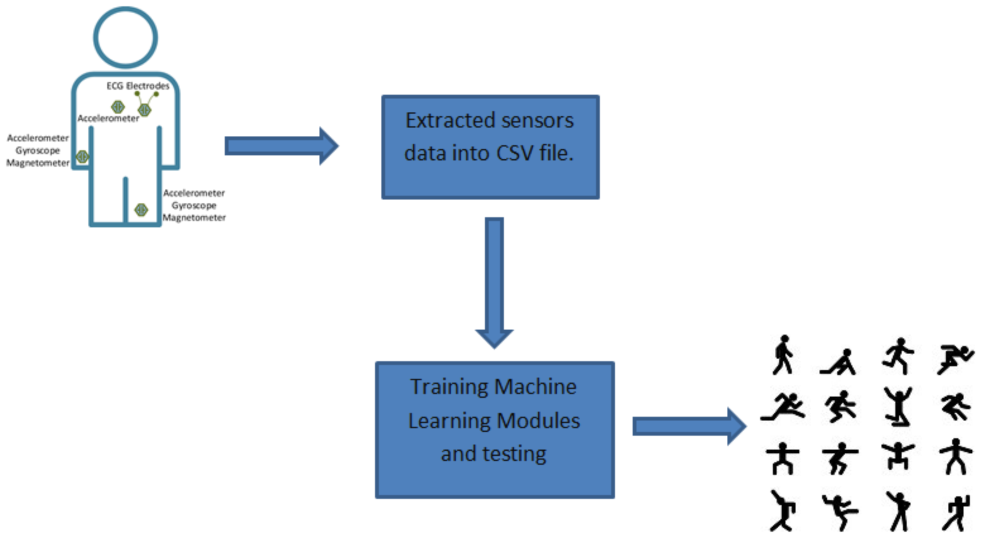
Multiclass classification: Human Physical Activities Recognition using wearable Sensors data.

Module Leader: Dr. Alessandro Di Stefano



Machine Learning: Technical report

MSc Data Science
School of Computing, Engineering & Digital Technologies
Teesside University
Middlesbrough
Tees Valley, TS1 3BX
UK

Author: Konda Reddy Thokala
Student Id: C2994587

Multiclass classification: Human Physical Activities Recognition using wearable sensors data.

Author: Konda Reddy Thokala
Mail id: C2994587@live.tees.ac.uk

MSc Data Science
School of Computing, Engineering & Digital Technologies
Teesside University
Middlesbrough
Tees Valley, TS1 3BX

Abstract:

Human Physical activity recognition using Machine learning (ML) and Deep Learning (DL) has been one of the popular research areas in recent times. There are so many methodologies described in papers to recognize human activities with Machine Learning. As so many smart devices got evolved with sensors on board, we can use accumulated data to apply Machine learning (ML) / Deep Learning (DL) algorithms to describe daily human activities and can provide the necessary tips to maintain their fitness. In this artifact and report, I tried to explain how popular machine-learning classification algorithms can be used to recognize human activities based on a dataset that was collected for a novel framework for the agile development of mobile health applications [1] [2]. The algorithms are then compared with one another to justify which methodology and Algorithm can be the best fit to recognize the Human activities for the collected data.

1. Introduction:

Human physical activity recognition technology has a wide range of applications in various fields such as healthcare, robotics, sports, etc... In healthcare, it can be used to monitor patient movement, track progress during rehabilitation, and identify potential health problems. This technology can be used in Robotics to help develop robots that can perform tasks that require human-like dexterity, precision, and movements. In sports, it helps coaches analyse and improve an athlete's performance by tracking their movement and identifying areas for improvement. Similarly, there are so many other advantages to this technology in day-to-day human life. Overall, human physical activity recognition technology has the potential to revolutionize the way we approach healthcare, robotics, and sports by providing accurate and efficient insight into human movement and behaviour.

2. About Dataset

The MHEALTH dataset is taken from the website and the dataset includes recordings of ten participants' bodily movements and vital signs as they engaged in a variety of physical activities. The subject's chest, right wrist, and left ankle are fitted with sensors that record the acceleration, rate of turn, and magnetic field orientation of various body parts. The sensor, which is placed on the chest, also generates 2-lead ECG readings, which may be utilised for routine cardiac monitoring, screening for different arrhythmias, or examining how exercise affects the ECG.

The sensors used for the recordings are acceleration, magnetometer, and gyroscope and each sensor will produce the data in 3 different plans (X, Y, Z) [fig. 1].

- Acceleration is used to measure the rate of change of velocity of an object and it is measured in (m/s^2).
- Magnetometer is used to measure a magnetic field's direction, intensity, or relative change.
- Gyroscope is a tool for determining or maintaining direction and angular velocity and it is measured in (deg/s).

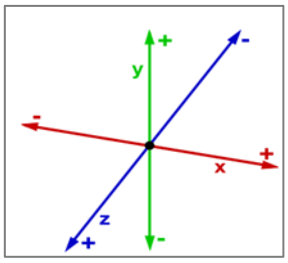
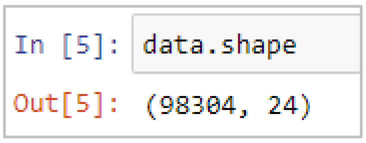


Fig. 1: *x, y, z axis*

Apart from these sensors, an electrocardiogram (ECG) is also used to determine the heart's rhythm and electrical activity.

2.1. Data Exploration and Preparation:

The original dataset contains the samples collected from 10 volunteers, but to reduce the size of the dataset, I have tried on only 1 volunteer data. From the combined data, there the dimension of the dataset is 98304, 24 [Fig 2].

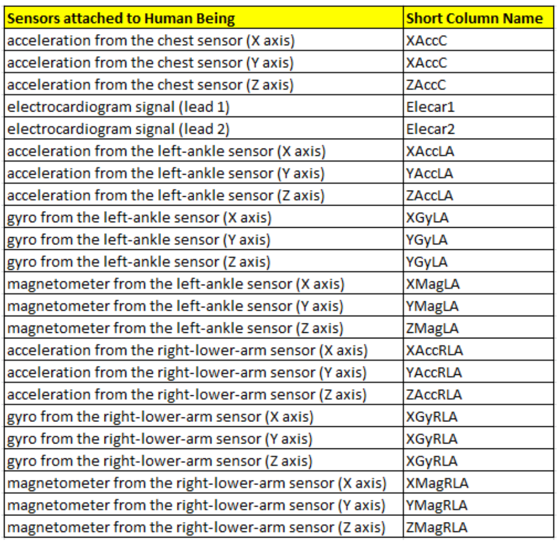


```
In [5]: data.shape

Out[5]: (98304, 24)
```

Fig. 2*: Shape of the dataset from code.*

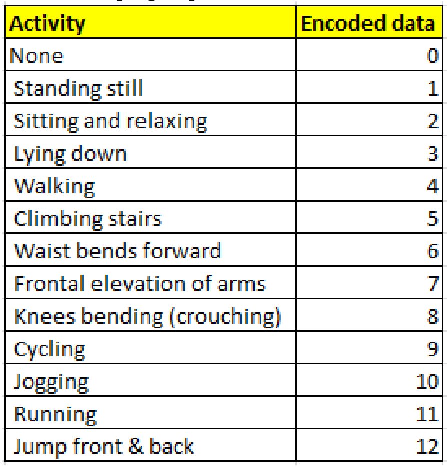
Below is the list of sensors used to measure the data and respective short column names which are used in my dataset[Fig 3].



Sensors attached to Human Being	Short Column Name
acceleration from the chest sensor (X axis)	XAccC
acceleration from the chest sensor (Y axis)	XAccC
acceleration from the chest sensor (Z axis)	ZAccC
electrocardiogram signal (lead 1)	Elecar1
electrocardiogram signal (lead 2)	Elecar2
acceleration from the left-ankle sensor (X axis)	XAccLA
acceleration from the left-ankle sensor (Y axis)	YAccLA
acceleration from the left-ankle sensor (Z axis)	ZAccLA
gyro from the left-ankle sensor (X axis)	XGyLA
gyro from the left-ankle sensor (Y axis)	YGyLA
gyro from the left-ankle sensor (Z axis)	YGyLA
magnetometer from the left-ankle sensor (X axis)	XMagLA
magnetometer from the left-ankle sensor (Y axis)	YMagLA
magnetometer from the left-ankle sensor (Z axis)	ZMagLA
acceleration from the right-lower-arm sensor (X axis)	XAccRLA
acceleration from the right-lower-arm sensor (Y axis)	YAccRLA
acceleration from the right-lower-arm sensor (Z axis)	ZAccRLA
gyro from the right-lower-arm sensor (X axis)	XGyRLA
gyro from the right-lower-arm sensor (Y axis)	XGyRLA
gyro from the right-lower-arm sensor (Z axis)	XGyRLA
magnetometer from the right-lower-arm sensor (X axis)	XMagRLA
magnetometer from the right-lower-arm sensor (Y axis)	YMagRLA
magnetometer from the right-lower-arm sensor (Z axis)	ZMagRLA

Fig. 3*: List of Sensors and their short names.*

Similarly below is the list of activities measured from the collected data and how it is encoded in the dataset [Fig: 4].



Activity	Encoded data
None	0
Standing still	1
Sitting and relaxing	2
Lying down	3
Walking	4
Climbing stairs	5
Waist bends forward	6
Frontal elevation of arms	7
Knees bending (crouching)	8
Cycling	9
Jogging	10
Running	11
Jump front & back	12

Fig. 4*: List of Activities and Encoded labels*

As the data is already partially pre-processed, we don’t have any missing values and duplicate values to drop from the dataset. Along with this, we have only float64 and int64 data types in the dataset. Hence no action is required on the dataset.

To compare the correlation between the columns, I have used a heat map to determine the relation as shown in the below figure. [Fig. 5].

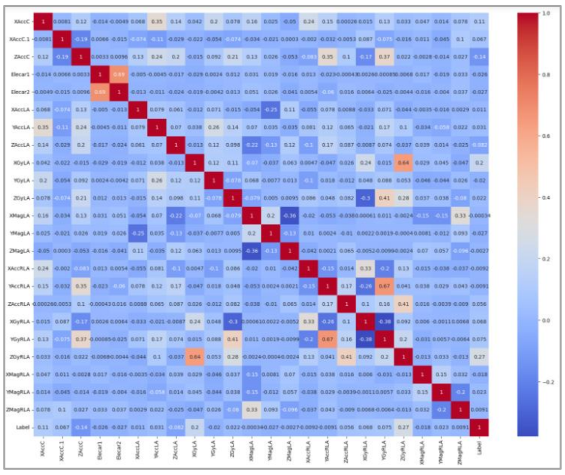


Fig. 5*: Correlation matrix between columns*

From the above correlation matrix, we can see there is no correlation more than 70%, Hence we are not dropping any column from the dataset.

We can see there are outliners in the data, Hence we need to remove the outliner in the data to increase the accuracy and decrease the error rate [Fig. 6].

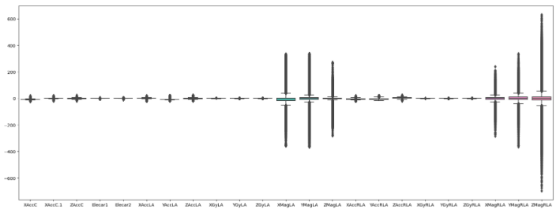


***Fig. 6**: Outliners box plot.*

An observation that differs abnormally from other values in a population-based random sample is referred to as an *outlier*. In a way, this concept defers the analyst's judgement (or a consensus process) in determining what constitutes aberrant behavior [3]. To overcome the outliners' effect, we need to perform Scaling techniques and for the dataset with outliners, the robust scaler technique is the best option to scale the data accordingly.

With this *robust scaler*, the median is eliminated and the data is scaled according to the quantile range (IQR, or interquartile range, by default). The interval between the first quartile's (25th quantile) and third quartile's (75th quantile) values is known as the IQR.[4].

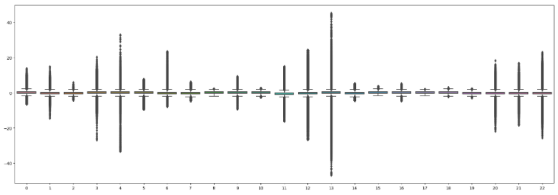


***Fig. 7**: Box plot of scaled data*

After applying the robust scaler technique, the complete data got scaled to IQR, and above is the box plot for the same.[Fig. 7].

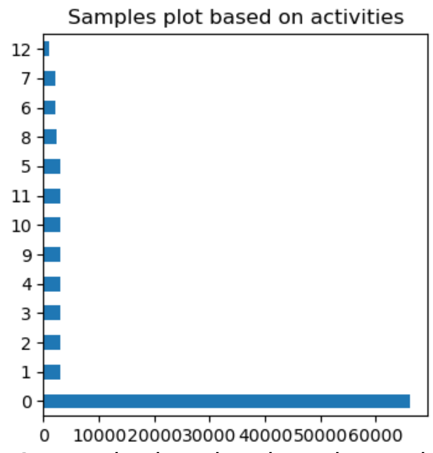


***Fig. 8:** samples bar chart based on activities*

We have observed that the data is imbalanced in terms of data for each activity[Fig. 8] and training the imbalance dataset to the Machine learning algorithms will lead to biased prediction and accuracy and the error rate will get affected due to the imbalance dataset. To overcome these types of scenarios, we need to resample the data to balance it.

Oversampling the minority numbers is one method for dealing with imbalanced datasets. The simplest technique is to duplicate instances from the minority class, but these examples contribute no additional insight into the model. Instead, new instances may be produced by combining existing ones. The **Synthetic Minority Oversampling Technique (SMOTE)** is a data augmentation technique for the minority population [5].

Below is the outcome of the SMOTE oversampling technique [Fig. 9].

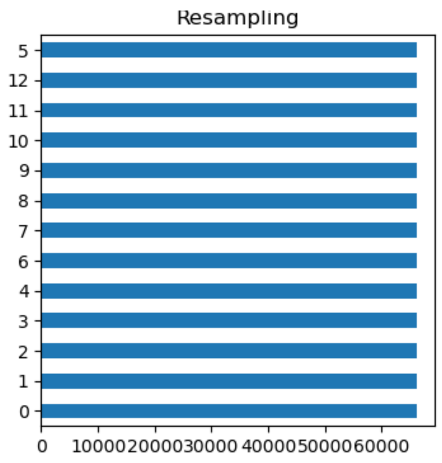
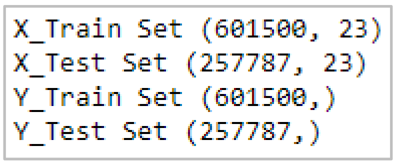


***Fig. 9:** Samples bar graph after oversampling (SMOTE)*

Through these steps, the dataset is cleaned and processed to train the machine learning algorithms to predict the values.

To go with the ML algorithms, I have split the dataset into a train set and test set with 70% and 30% respectively [Fig. 10].


```
X_Train Set (601500, 23)
X_Test Set (257787, 23)
Y_Train Set (601500,)
Y_Test Set (257787,)
```
***Fig. 10:** Dataset Split into Train and Test.*

3. About ML algorithms used:

As the dataset is related to multiclass classification, I have used the classifier algorithms to train and test the data in my artifact. Below are the algorithms used,

3.1. Decision Tree classifier:

A decision tree classifier is a supervised machine learning algorithm that is used in both classification and regression that uses a set of principles to make judgements, much like how the human brain makes a decision.

Below is a small example of how the decision tree classifier classifies the data by answering questions at every step [Fig. 11][6]

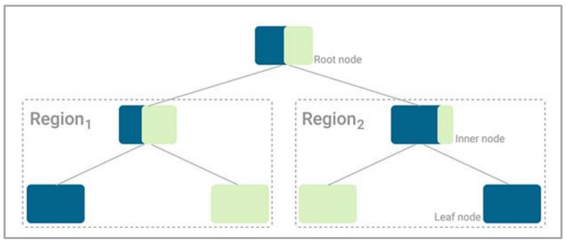


Fig. 11[6]: *Example of the different regions and types of nodes in a tree*

3.1.1. Parameters used in a decision tree classifier.

The Parameters are used to tune the algorithm to provide the maximum accuracy without overfitting the model.

The parameter used in my code is max_depth. Max depth is used to determine the depth of the decision tree and in the absence of the depth, the default value will be 'None' which leads to overfitting of the model.

To get the accurate value of max depth, I have checked with some depth values, and below is the observation of the accuracy and error with the respective max depth value [Fig .12].

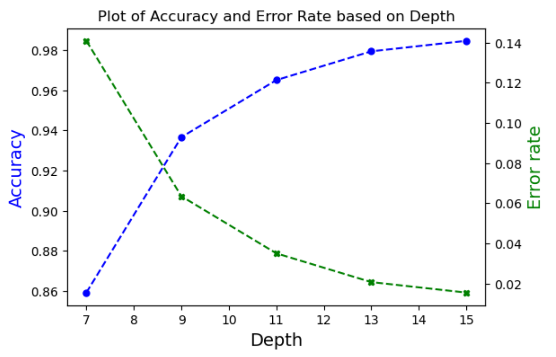


***Fig. 12**: Plot of Accuracy and Error rate based on max depth.*

From the above graph [Fig. 12], we can see to the max depth of 11, the accuracy and error rate line hoped very high, Hence I'm assuming the Max depth is 11 for my dataset, and below are the predicted values versus actual values graph [Fig 13].

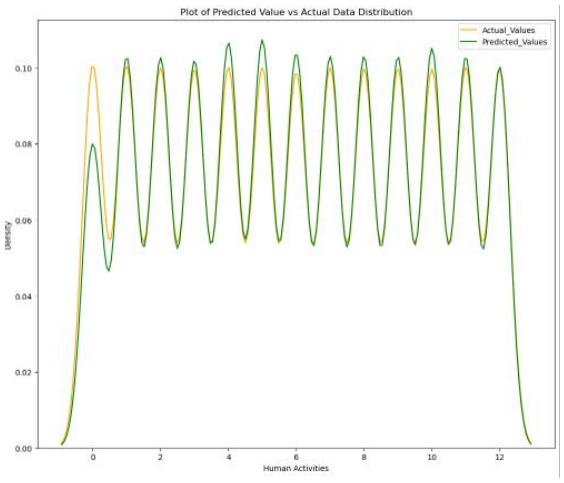


***Fig 13:** Predicted vs. Actual data plot*

3.2. Random Forest Classifier:

Random forest classifier is a supervised machine learning algorithm that is also used in both classification and regression. A random forest consists of multiple decision trees and each tree predicts an output. As a classifier, it will take the majority-voted output from all the decision trees for a sample. As it combines n number of decision trees and takes the max-voted output as the prediction for the sample [Fig 14][7], it will be more accurate than the decision tree classifier algorithm.

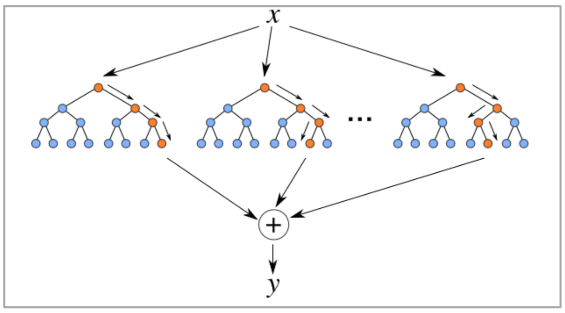


***Fig. 14[7]:** Flowchart of random forest*

3.2.1. Parameters used in Random forest classifier.

The parameter used in my code is max_depth. As the random forest classifier is a combination of the decision tree algorithm, the max_depth concept remains the same. Normally it is defaulted to 'None' which will consider the tree till the last leaf; hence there is a high chance of overfitting in the model. To overcome this issue, I have checked the possible depth values to determine the suitable depth variable which gives max accuracy with less error rate [Fig 15].

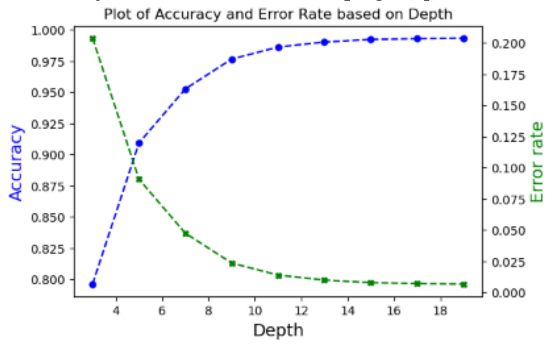


Fig 15: *Plot of Accuracy and Error rate based on max depth.*

From the above graph, we can see the model became stable after max depth 11, Hence I'm assuming the max_depth as 11 from my random forest classifier with my dataset. Below is the plot for actual values vs. predicted values with max_depth 11 [Fig. 16].

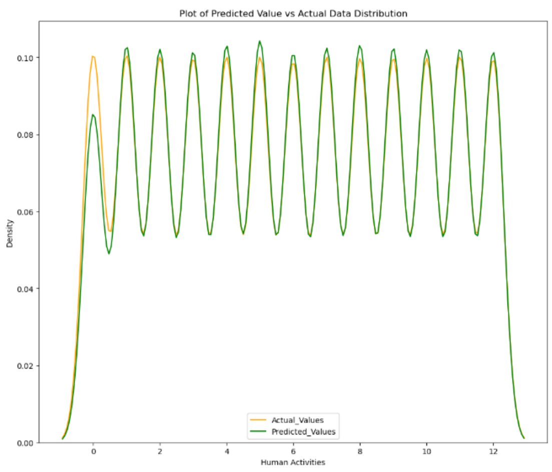


Fig 16: *Predicted vs. Actual data plot*

3.3. k-nearest neighbors algorithm:

The k-nearest neighbour algorithm also called a KNN or k-NN is a non-parametric, supervised learning classifier. It uses the k value as a parameter to check the k number of neighbors to determine/predict the sample's class.

Below is the pictorial representation of how the KNN algorithm works.

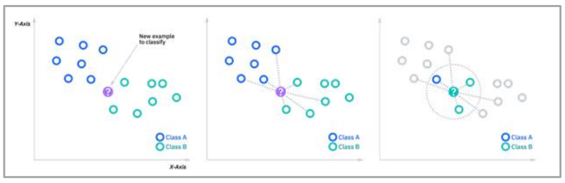


Fig. 17[8]*: Knn algorithm working principle.*

In the above figure [Fig. 17][8], the new data point's class needs to determine and for that k is taken as 3. Hence it compared the data with 3 nearest neighbors which are having minimum Euclidean distance with the train data and new data point.

3.3.1. Parameters used for KNN algorithm.

In the KNN algorithm, the main boundary to tune the model is n_neighbors. Several values of k must be taken into consideration when defining it to avoid either overfitting or underfitting. By default, it is set to 5. Bigger upsides of k might bring areas of strong bias and low variance, while more modest upsides of k might have high variance but low bias. The determination of k will be intensely affected by the information dataset, as information with additional anomalies or clamor will likely perform better with higher upsides of k. As a general rule, it is recommended to involve an odd number for k to prevent classification ties.

Below is the plot for accuracy and error rates versus K values.[Fig. 18]

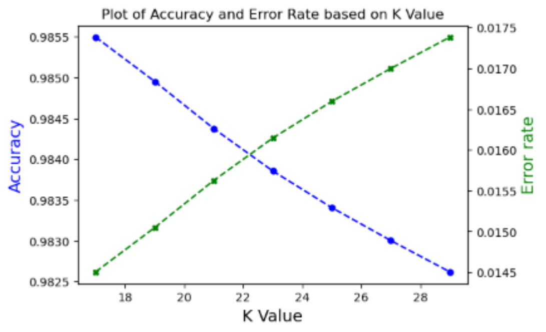


Fig. 18*: Plot of Accuracy and Error rate based on k value.*

Based on the above plot, we can see that after the k value =19 the accuracy and error rate are showing some stable accuracy and error rate till k=27. Hence I'm assuming n_neighbors=19 and with this value below is the plot of actual and predicted values [Fig 19].

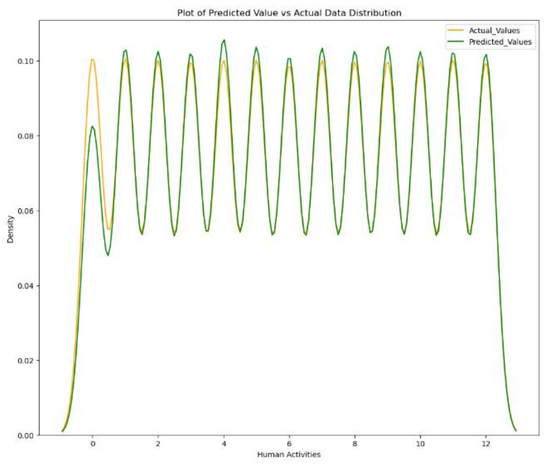


Fig 19: *Predicted vs. Actual data plot*

4. Validation technique:

The most crucial benefit of using appropriate validation approaches is the estimation of an unbiased generalisation performance, which aids in understanding the model.

4.1. k-fold Cross Validation:

To validate my models, I have used the k-fold Cross Validation technique as this is a statistical method to gauge the expertise of machine learning models. In this validation technique, the data is divided into k folds, after which it is trained on k-1 folds and tested on the remaining fold [Fig 20][9]. This is done for all possible combinations, and the outcome for each instance is averaged.

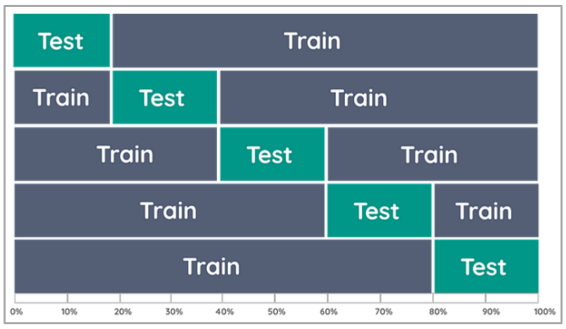
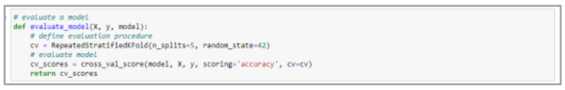


Fig. 20[9]*: 5-fold cross-validation on the dataset.*

On my dataset, I have applied 5-fold cross-validation on the dataset on all the models by calling a function **evaluate_model()**. Below is the code for the same [Fig. 21]



```
# evaluate a model
def evaluate_model(X, y, model):
    # define evaluation procedure
    cv = RepeatedStratifiedKFold(n_splits=5, random_state=42)
    # evaluate model
    cv_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv)
    return cv_scores
```

Fig. 21*: Cross-validation code*

And below are the evaluation details of all the 3 models and we can see the standard deviati on is not more than 0.001. Hence we can say t he models are neither over-fitted nor under-fitt ed.

5. Discussion:

The models used in the code and obtained acc uracies respectively as shown in the below gra ph [Fig. 22], the Random Forest Algorithm obt ained % accuracy with the 0.000 [Fig. 23]. Hen ce I can conclude the Random Forest Algorith m is the best-performing model on my dataset.

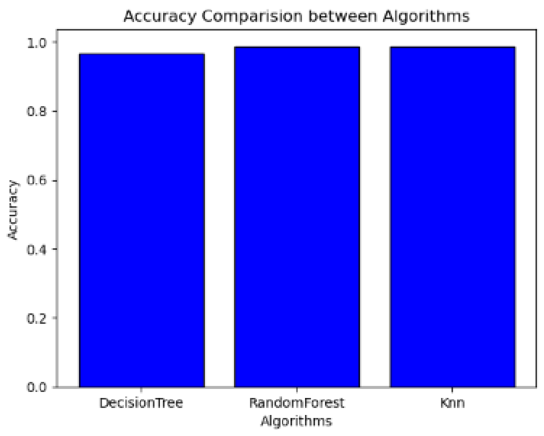
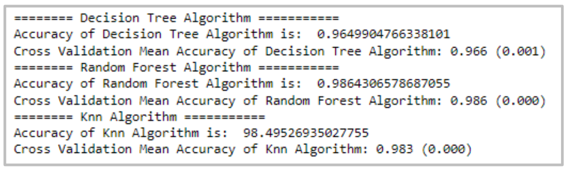


Fig. 22*: Comparison of accuracies of a model.*



```
======== Decision Tree Algorithm ===========
Accuracy of Decision Tree Algorithm is:  0.9649904766338101
Cross Validation Mean Accuracy of Decision Tree Algorithm: 0.966 (0.001)
======== Random Forest Algorithm ===========
Accuracy of Random Forest Algorithm is:  0.9864306578687055
Cross Validation Mean Accuracy of Random Forest Algorithm: 0.986 (0.000)
======== Knn Algorithm ===========
Accuracy of Knn Algorithm is:  98.49526935027755
Cross Validation Mean Accuracy of Knn Algorithm: 0.983 (0.000)
```

Fig. 23: *Results*

6. References:

[1]. Banos, O., Garcia, R., Holgado, J. A., Damas, M., Pomares, H., Rojas, I., Saez, A., Villalonga, C. mHealthDroid: a novel framework for agile development of mobile health applications. Proceedings of the 6th International Work-conference on Ambient Assisted Living and Active Ageing (IWAAL 2014), Belfast, Northern Ireland, December 2-5, (2014).

[2]. Nguyen, L. T., Zeng, M., Tague, P., Zhang, J. (2015). Recognizing New Activities with Limited Training Data. In IEEE International Symposium on Wearable Computers (ISWC).

[3].National Institute of Standard and Technology, US. Department of Commerce (2023) URL: https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm#:~:text=An%20outlier%20is%20an%20observation,what%20will%20be%20considered%20abnormal.

[4]. Scikit learn (2023) URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

[5].Machine learning mastery (2023) URL: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

[6].Towards data science (2023) URL: https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575

[7]. Data-driven investor (2023) URL: https://medium.datadriveninvestor.com/random-forest-pros-and-cons-c1c42fb64f04

[8] IBM.com (2023) URL: https://www.ibm.com/topics/knn

[9] Towards data science (2023) URL: https://towardsdatascience.com/validating-your-machine-learning-model-25b4c8643fb7

Code:

```
# Multiclass classification: Human Physical Activities Recognition using wearable Sensors data.
#!pip install imblearn
# Libraries for Data Preprocessing
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt

# Libraries for Classifiers
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Libraries for measuring accuracy
from numpy import mean
from numpy import std
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score

# Libraries to ignore the future warnings
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
data = pd.read_csv('Dataset2.csv')
data.head()
data.shape
data=data.drop_duplicates()
data.shape
data.isna().sum()
data.info()
data.describe().T
data.corr()
# This statement reduces the size of image
plt.figure(figsize=(20,15))
# plotting the graph
sns.heatmap(data.corr(),annot=True,cmap='coolwarm')
# As the correlation of the columns is not more than 0.7, Hence we can say the columns are completely independent
Xdata = data.drop(['Label'], axis=1)
Ydata= data['Label']
print(Xdata.shape)
print(Ydata.shape)
# plot to show the raw data to check the outliners
plt.figure(figsize=(22,8))
sns.boxplot(data=Xdata)
plt.show()
#As we can see there are outliners in the data, Hence i'm using a Robust scaling techinque to remove the outliners.
# Robust Scaling technique to removes the median and scales the data according to the Quartile range
RS = RobustScaler()
Xdata = RS.fit_transform(Xdata)
# plot to show the Scaled data to the quartile range.
plt.figure(figsize=(24,8))
```

```
sns.boxplot(data=Xdata)
plt.show()
# To check the inbalance data
#let's show this with bar chart
plt.figure(figsize=(4,4))
Ydata.value_counts().plot(kind='barh')
plt.title('Samples plot based on activities')
# As the data is not balanced, we need to sample the data according to overcome the biased results.
# performing Synthetic Minority Oversampling Technique, (SMOTE) to create by synthesising the
current ones.
overSampling = SMOTE()
X_res, y_res = overSampling.fit_resample(Xdata, Ydata)
plt.figure(figsize=(4,4))
ax = y_res.value_counts().plot(kind='barh')
_ = ax.set_title("Resampling")
#Split data into random train and test subsets (30% Test dataset, 70% training dataset).
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_res, y_res, test_size=0.3, random_state=101)
y_train1 = np.ravel(y_train1)
print ('X_Train Set' ,X_train1.shape)
print ('X_Test Set' ,X_test1.shape)
print ('Y_Train Set' ,y_train1.shape)
print ('Y_Test Set' ,y_test1.shape)
# Declaring functions for Machine Learning algorithm and metrics
# Function for Algorithm
def MLAlgorithm (model, X_train, y_train,X_test,y_test):
    # We fit our model with our train data
    model.fit(X_train, y_train)
    # Then predict results from X_test data
    predy = model.predict(X_test)
    # Then calculate the Accuracy from the model
    Accu = accuracy_score(y_test, predy)
    return predy, Accu
# evaluate a model with cross validation
def evaluate_model(X, y, model):
    # define evaluation procedure
    cv = RepeatedStratifiedKFold(n_splits=5, random_state=42)
    # evaluate model
    cv_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv)
    return cv_scores
#To plot Error and Accuracy graphs based on the parameters.
def AccuErrPlot(acc, err, refVal, title,Xlabel):
    # create figure and axis objects with subplots()
    fig,ax = plt.subplots(figsize=(6,4))
    # make a plot
    ax.plot(refVal,acc,color='blue', linestyle='dashed', marker='o',markerfacecolor='blue', markersize=5,
label = 'Accuracy')
    # twin object for two different y-axis on the sample plot
    ax2=ax.twinx()
    # make a plot with different y-axis using second axis object
    ax2.plot(refVal,err,color='green',     linestyle='dashed',     marker='X',markerfacecolor='green',
markersize=5, label = 'Error')
    # set x-axis label
    ax.set_xlabel(Xlabel, fontsize = 14)
    # set y-axis label
    ax.set_ylabel("Accuracy",color="blue",fontsize=14)
    ax2.set_ylabel("Error rate",color="green",fontsize=14)
    plt.title(title)
    plt.show()
#To plot Predicted Values vs Actual Values
def PredvsActuPlot(Actual_Values, Predicted_Values):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))
    # Defining the axis
    ax1 = sns.kdeplot(Actual_Values,  color="orange", label='Actual_Values')
    ax2 = sns.kdeplot(Predicted_Values, color="green", label='Predicted_Values', ax=ax1)
    # set Title
    plt.title('Plot of Predicted Value vs Actual Data Distribution')
    # set x-axis label
    plt.xlabel('Human Activities')
    plt.legend(loc="best")
    plt.show()
    plt.close()
Algorithms =[]
Accuracy = []
#Decision Tree Classifier
DTacc=[]
DTerr=[]
DTdepth = []
for i in range(7,16,2):
    DTdepth.append(i)
    DecisionTreeTest = DecisionTreeClassifier(max_depth = i, random_state = 42)
    predy_dt_test, DT_acc = MLAlgorithm(DecisionTreeTest,X_train1,y_train1,X_test1,y_test1)
    DTacc.append(DT_acc)
    DTerr.append(mean(predy_dt_test != y_test1))
AEP_Title = 'Plot of Accuracy and Error Rate based on Depth'
X_label = 'Depth'
AccuErrPlot(DTacc, DTerr, DTdepth, AEP_Title, X_label)
# As we can see from the graph, after the depth 11 the accuracy and error rate are showning some
sudden deviation.
#Hence assuming the depth as 11 for Decision Tree algorithm
AlgorithmName = 'DecisionTree'
depth = 11
DecisionT = DecisionTreeClassifier(max_depth = depth, random_state = 42)
predy_dt, DT_Accuracy = MLAlgorithm(DecisionT,X_train1,y_train1,X_test1,y_test1)
Algorithms.append(AlgorithmName)
Accuracy.append(DT_Accuracy)
PredvsActuPlot(y_test1, predy_dt)
dtScores = evaluate_model(X_train1,y_train1, DecisionT)
#Random Forest Classifier
RFacc=[]
RFerr=[]
RFdepth = []
for i in range(3,20,2):
    RFdepth.append(i)
    RandomForestTest = RandomForestClassifier(max_depth = i)
    predy_rf_test, RF_acc = MLAlgorithm(RandomForestTest,X_train1,y_train1,X_test1,y_test1)
    RFacc.append(RF_acc)
    RFerr.append(mean(predy_rf_test != y_test1))
AEP_Title = 'Plot of Accuracy and Error Rate based on Depth'
X_label = 'Depth'
AccuErrPlot(RFacc, RFerr, RFdepth, AEP_Title, X_label)
# As we can see from the graph, after the depth 11 the accuracy and error rate are showning some
sudden deviation.
#Hence assuming the depth as 11 for Random Forest algorithm
AlgorithmName='RandomForest'
RandomForest = RandomForestClassifier(max_depth = 11)
predy_rf, RF_Accuracy = MLAlgorithm(RandomForest,X_train1,y_train1,X_test1, y_test1)
Algorithms.append(AlgorithmName)
Accuracy.append(RF_Accuracy)
```

```
PredvsActuPlot(y_test1, predy_rf)
rfScores = evaluate_model(X_train1,y_train1, RandomForest)
#KNN Classifier
kacc=[]
kerr=[]
kvalue=[]
for k in range(17,30, 2):
    kvalue.append(k)
    KnnTest = KNeighborsClassifier(n_neighbors=k, n_jobs =-1)
    predy_k_test, Knn_Acc = MLAlgorithm(KnnTest,X_train1,y_train1,X_test1, y_test1)
    kacc.append(Knn_Acc)
    kerr.append(mean(predy_k_test != y_test1))
    print("Accuracy=",Knn_Acc, "K= ",k)
AEP_Title = 'Plot of Accuracy and Error Rate based on K Value'
X_label = 'K Value'
AccuErrPlot(kacc, kerr, kvalue, AEP_Title, X_label)
# As we can see from the above graph, after the k value =19 the accuracy and error rate are
showning some stable accuracy and error rate till k=27.
# Hence assuming the k value as 19 for Knn algorithm
AlgorithmName='Knn'
kValue= 19
Knn = KNeighborsClassifier(n_neighbors=kValue, n_jobs =-1)
predy_knn, KNN_Accuracy = MLAlgorithm(Knn,X_train1,y_train1,X_test1, y_test1)
Algorithms.append(AlgorithmName)
Accuracy.append(KNN_Accuracy)
PredvsActuPlot(y_test1, predy_knn)
knnScores = evaluate_model(X_train1,y_train1, Knn)
#Plot for Accuracy vs Algorithms
plt.bar(Algorithms, Accuracy, color='blue', ec='black')
plt.xlabel('Algorithms')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparision between Algorithms')
plt.show()
#Printing the Scores and Standard deviation.
print("======== Decision Tree Algorithm ==========")
print("Accuracy of Decision Tree Algorithm is: ",DT_Accuracy*100)
print('Cross Validation Mean Accuracy of Decision Tree Algorithm: %.3f (%.3f)' % (mean(dtScores),
std(dtScores)))
print("======== Random Forest Algorithm ==========")
print("Accuracy of Random Forest Algorithm is: ",RF_Accuracy*100)
print('Cross Validation Mean Accuracy of Random Forest Algorithm: %.3f (%.3f)' % (mean(rfScores),
std(rfScores)))
print("======== Knn Algorithm ==========")
print("Accuracy of Knn Algorithm is: ",KNN_Accuracy*100)
print('Cross Validation Mean Accuracy of Knn Algorithm: %.3f (%.3f)' % (mean(knnScores),
std(knnScores)))
##
```