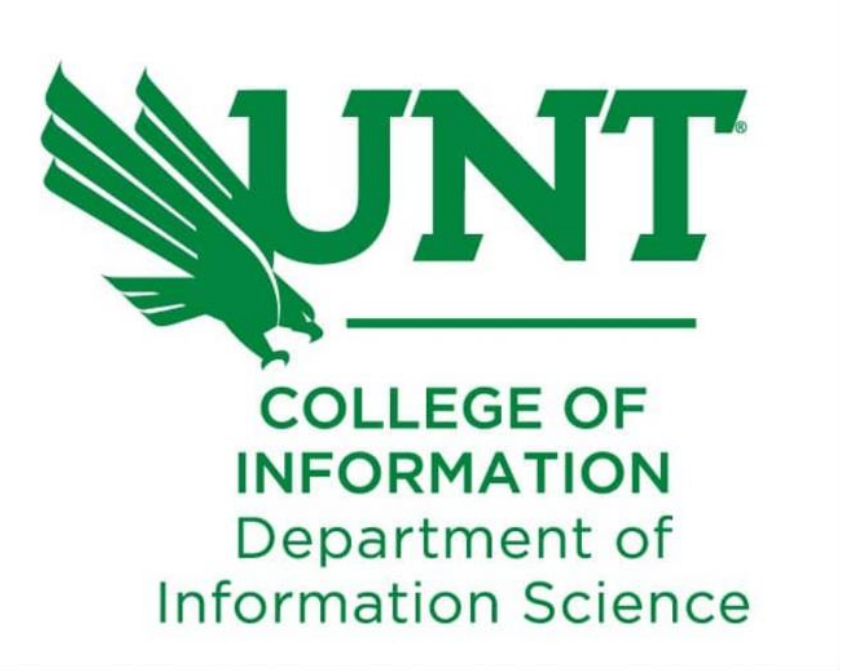


Principles and Techniques of Data Science

INFO- 5502

Fall 2022

Project: Human Activity Recognition Classification using Machine Learning

Dataset: Human Activity Recognition Dataset

Rahul Nomula	**Devi Nageswari Kondeti**	**Sujit Murahari Giridharan**	**Vineeth Thogarla**
niversity of North Texas 1155 Union Circle Denton, TX 76203	University of North Texas 1155 Union Circle Denton, TX 76203	University of North Texas 1155 Union Circle Denton, TX 76203	Jniversity of North Texas 1155 Union Circle Denton, TX 76203
11560577	11565319	11589051	11598227
rahulnomula@my.unt.edu	devinageswarikondeti@my.unt.edu	sujitmuraharigiridharan@my.unt.edu	vineeththogarla@my.unt.edu

Abstract

Human activity recognition (HAR) attempts to determine a user's status by continually monitoring a variety of physiological signals derived from sensor data. HAR has grown in popularity in recent years due to its applicability in a variety of domains like as health, security, and surveillance. An effective method for remotely monitoring the activity of old and physically disabled patients can be developed using the sensor data of human activity captured using smart phones linked to the subject body.

The project intends to accurately categorize input data into its underlying activity category using readings from accelerometers and gyroscopes sensors. Walking, walking upstairs, walking downstairs, sitting, standing, and laying are all tracked behaviors. Various predictive models are used to accurately classify these actions.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 About

Human Activity Recognition (HAR) is from a sensor data which can be very useful in developing assistive technology for applications like daily activity monitoring elderly and physically impaired people. The basic six activities (walking, walking upstairs, walking downstairs, sitting, standing, laying) can create constructive feedback for numerous disease monitoring systems.

1.2 About Machine Learning

A data analysis technique called machine learning automates the creation of analytical models. It is a subfield of artificial intelligence founded on the notion that machines are capable of learning from data, seeing patterns, and making judgments with little assistance from humans.

1.3 Problem Statement

The objective is to classify the activities performed in one of the six activities based on reading obtained from the accelerometer and gyroscope.

1.4 Description

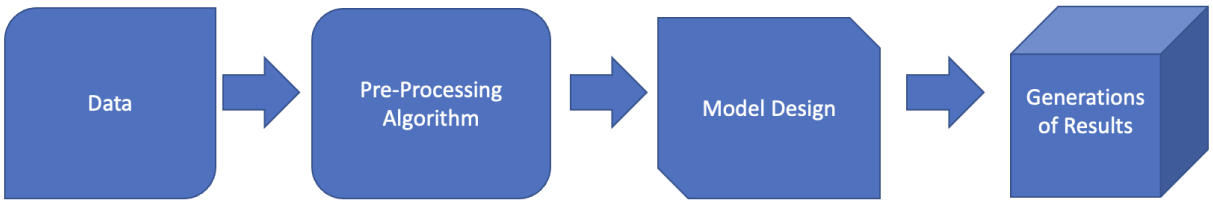
The Human Activity Recognition Dataset was created through trials with 30 volunteers, aged between 19 and 48, who carried out daily activities while wearing a smartphone attached on their waist, including walking, going upstairs, downstairs, sitting, standing, and lying down (Samsung Galaxy S II).

3-axial linear acceleration and 3-axial angular velocity at a rate of 50Hz were recorded using the integrated accelerometer and gyroscope. The experiments were videotaped so that the data could be manually labeled. The obtained dataset was then split into two sections at random; information from 70% of the population was chosen to create the training data, and information from the remaining 30% was utilized to create the test data.

2 DESIGN & ARCHITECTURE

The project's architecture is presented below in the form of a flowchart and is broadly divided into the following four stages:

1. Collecting the data
2. Pre-processing the data
3. Designing the data model
4. Generating and visualizing the results



The following are the hardware and software requirements utilized in this project:

- The minimum hardware requirements are 8 GB RAM, a dual-core CPU, and a 64-bit operating system (Windows/Linux).

- Jupyter Notebook with Anaconda is the Integrated Development Environment (IDE) utilized in this project. You can use it to create and execute machine learning code. The programming language chosen for this project is Python.

3 DATA CARPENTRY

The data source link of the chosen project is **https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones** .

The above dataset contains two data sets in which we have taken the test dataset for the better purpose of understanding.

```
In [2]: df = pd.read_csv("C:/Users/dkond/Downloads/archive/test.csv")
        df.head()
```

Out[2]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle(
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249	-0.674302	-0.894088	...	-0.705974	
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401	-0.945823	-0.894088	...	-0.594944	
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735	-0.963483	-0.939260	...	-0.640736	
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471	-0.968897	-0.938610	...	-0.736124	
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953	-0.977346	-0.938610	...	-0.846595	

5 rows × 563 columns

Fig 3.1

These are the properties of the dataset, which contains three-dimensional data represented as XYZ. The following information is included for each dataset record:

Attribute Data:

The predicted body acceleration and the total triaxial acceleration from the accelerometer. Gyroscope-derived triaxial angular velocity a 561-feature vector with variables in the time and frequency domains.

3-axial signals in the X, Y, and Z axes are denoted by the symbol "-XYZ."

The column names are as follows:

1.tBodyAcc-XYZ
2.tGravityAcc-XYZ
3.tBodyAccJerk-XYZ
4.tBodyGyro-XYZ
5.tBodyGyroJerk-XYZ
6.tBodyAccMag
7.tGravityAccMag
8.tBodyAccJerkMag
9.tBodyGyroMag
10.tBodyGyroJerkMag
11.fBodyAcc-XYZ
12.fBodyAccJerk-XYZ
13.fBodyGyro-XYZ
14.fBodyAccMag
15.fBodyAccJerkMag
16.fBodyGyroMag
17.fBodyGyroJerkMag

These signals were used to estimate the following set of variables:

- mean() Mean value
- std() Standard deviation
- mad() Median of the dataset
- max() Largest value in array
- min() Smallest value in array
- sma() Signal magnitude area
- energy() Energy measure
- iqr() Interquartile range
- entropy() Signal entropy
- arCoeff() Autorregres is on coefficients with Burg order equal to 4
- correlation() correlation coefficient between two signals
- maxInds() index of the frequency component with largest magnitude
- meanFreq() Weighted average of the frequency components to obtain a mean frequency
- skewness() skewness of the frequency domain signal
- kurtosis() kurtosis of the frequency domain signal
- bandsEnergy() Energy of a frequency interval within the 64 bins of the FFT of each window
- angle() Angle between to vectors

The following image depicts the sample descriptive statistics of the data as the entire statics cannot be put into a single image:

```
In [12]: df_des=df.describe()
         df_des
```

Out[12]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ	tBodyAccmadX	tBodyAccmadY	tBodyAccmadZ	t
count	2947.000000	2947.000000	2947.000000	2947.000000	2947.000000	2947.000000	2947.000000	2947.000000	2947.000000	
mean	0.273996	-0.017863	-0.108386	-0.613635	-0.508330	-0.633797	-0.641278	-0.522676	-0.637038	
std	0.060570	0.025745	0.042747	0.412597	0.494269	0.362699	0.385199	0.479899	0.357753	
min	-0.592004	-0.362884	-0.576184	-0.999606	-1.000000	-0.998955	-0.999417	-0.999914	-0.998899	
25%	0.262075	-0.024961	-0.121162	-0.990914	-0.973664	-0.976122	-0.992333	-0.974131	-0.975352	
50%	0.277113	-0.016967	-0.108458	-0.931214	-0.790972	-0.827534	-0.937664	-0.799907	-0.817005	
75%	0.288097	-0.010143	-0.097123	-0.267395	-0.105919	-0.311432	-0.321719	-0.133488	-0.322771	
max	0.671887	0.246106	0.494114	0.465299	1.000000	0.489703	0.439657	1.000000	0.427958	

8 rows × 562 columns

Fig: 3.2

The correlation of the data is as follows:

```
In [13]: df_corr= df.corr()
         df_corr
```

Out[13]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ	tBodyAccmadX	tBo
tBodyAccmeanX	1.000000	0.041274	-0.129645	0.016984	-0.001799	-0.008065	0.022942	
tBodyAccmeanY	0.041274	1.000000	0.225980	-0.054264	-0.059066	-0.077051	-0.052501	
tBodyAccmeanZ	-0.129645	0.225980	1.000000	-0.038578	-0.048340	-0.042342	-0.037851	
tBodyAccstdX	0.016984	-0.054264	-0.038578	1.000000	0.910636	0.896031	0.998828	
tBodyAccstdY	-0.001799	-0.059066	-0.048340	0.910636	1.000000	0.874501	0.909197	
...	...	...	...	...	...	...	...	
angletBodyGyroJerkMeangravityMean	0.049701	0.092905	-0.021375	-0.033609	-0.018611	-0.029401	-0.033386	
angleXgravityMean	-0.058421	-0.017138	-0.013933	-0.382696	-0.383742	-0.383490	-0.381896	
angleYgravityMean	0.034220	-0.030253	-0.007318	0.401433	0.467572	0.427469	0.397630	
angleZgravityMean	0.038936	-0.027410	-0.051057	0.388747	0.405681	0.488541	0.386152	
subject	0.005077	0.003163	0.021476	-0.068487	-0.036466	-0.028508	-0.064529	

562 rows × 562 columns

Fig 3.3

4 EXPLORATORY DATA ANALYSIS

4.1- Finding the right sensor essential for categorizing participants based on their walking style.

```
[ ]  Acc=0
     Gyro=0
     other=0
     for i in df.columns:
         if'Acc'in i:
             Acc += 1
         elif'Gyro'in i:
             Gyro += 1
         else:
             other += 1

     plt.figure(figsize=(8,8))
     plt.bar(x = ['Accelerometer ','Gyroscope '], height = [Acc,Gyro], color=['tab:gray', 'tab:orange'])
     plt.title("Sensor importance", fontsize = 25)
     plt.xlabel("sensor")
     plt.ylabel("used times")
     plt.xticks(fontsize=15)
     plt.show()
```

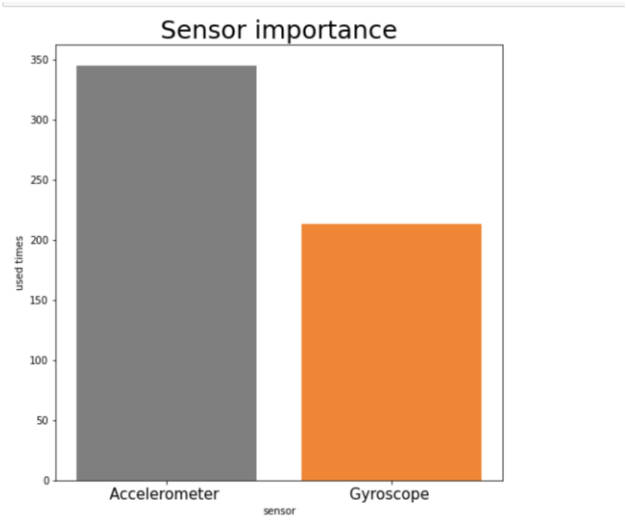


Fig: 4.1

From the above analysis we can define that the most widely used sensor is accelerometer.

4.2 - Displaying the distribution of moving and stationary activities.

```
sns.set_palette("Set1", desat=1)
facetgrid = sns.FacetGrid(df, hue='Activity', size=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", xy=(-0.956,10), xytext=(-0.8, 14), size=10,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

plt.annotate("Moving Activities", xy=(0,3), xytext=(0.2, 9), size=10,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))
plt.title('Stationary vs Moving activities', fontsize=20)
plt.xlabel("Acc Magnitude mean", size=15)
plt.ylabel('Density', size=15)
plt.show()
```

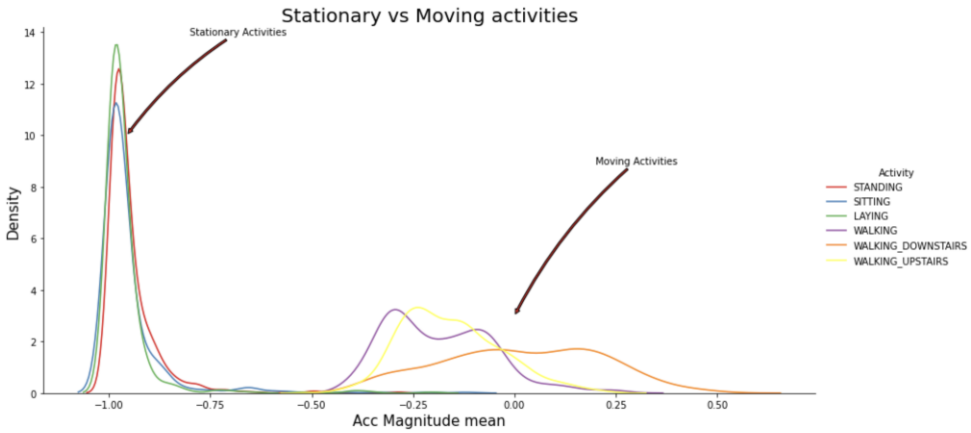


Fig: 4.2

From the observation, the distribution of stationary activities is greater than that of the moving activities.

4.3 - Examining the relationship between x and gravity

```
plt.figure(figsize=(9,7))
sns.boxplot(x='Activity', y= 'angleXgravityMean', data=df, showfliers=False)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,dashes=(5,5))
plt.title('Effect of activity on the Angle (X axis & Gravity)', fontsize=25)
plt.xlabel("Activity", size=23)
plt.ylabel('Angle (X & Gravity)', size=22)
plt.xticks(rotation = 90, fontsize = 20)
plt.show()
```

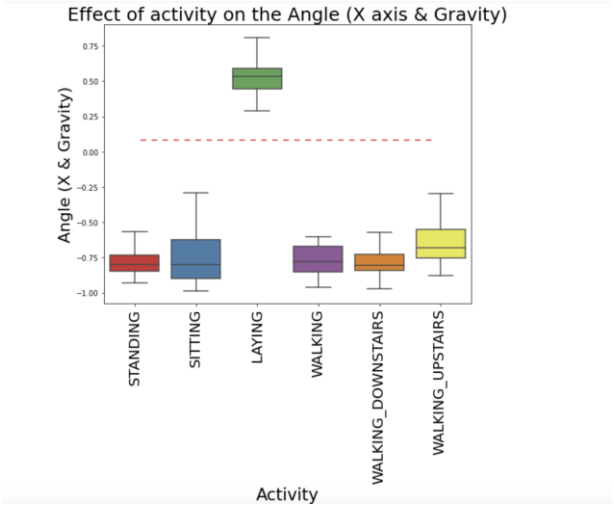


Fig: 4.3

From the observation, the Activity is laying if angle (X, gravity Mean) > 0

4.4 - Plotting the data provided by each user.

```
[ ]  sns.set_style('whitegrid')
     plt.figure(figsize=(16,8))
     sns.color_palette("tab10")
     plt.title('Data provided by each user', fontsize=24)
     sns.countplot(x='subject',hue='Activity', data = df)
     plt.xlabel("volunteer", size=23)
     plt.ylabel("Count", size=23)
     plt.xticks(size=15)
     plt.show()
```

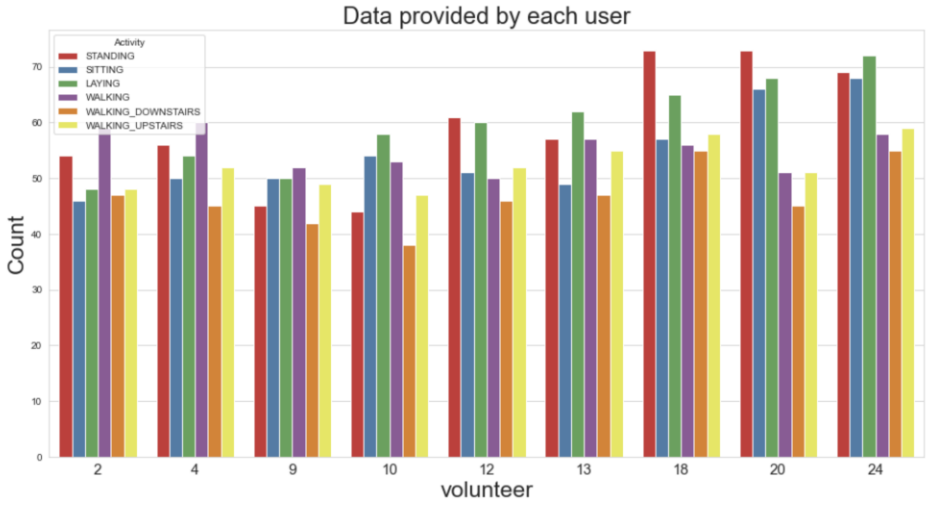


Fig: 4.4

4.5 - Checking for the imbalance in the data.

```
fig = px.pie(df, names='Activity',width=900)
fig.update_layout(
    title={
        'text': "Activities distribution in the data",
        'y':0.95,
        'x':0.40,
        'xanchor': 'center',
        'yanchor': 'top'},
         legend_title ="Activities Involved",
         font=dict(
         family="Arial",
         size=12))
fig.show()
```

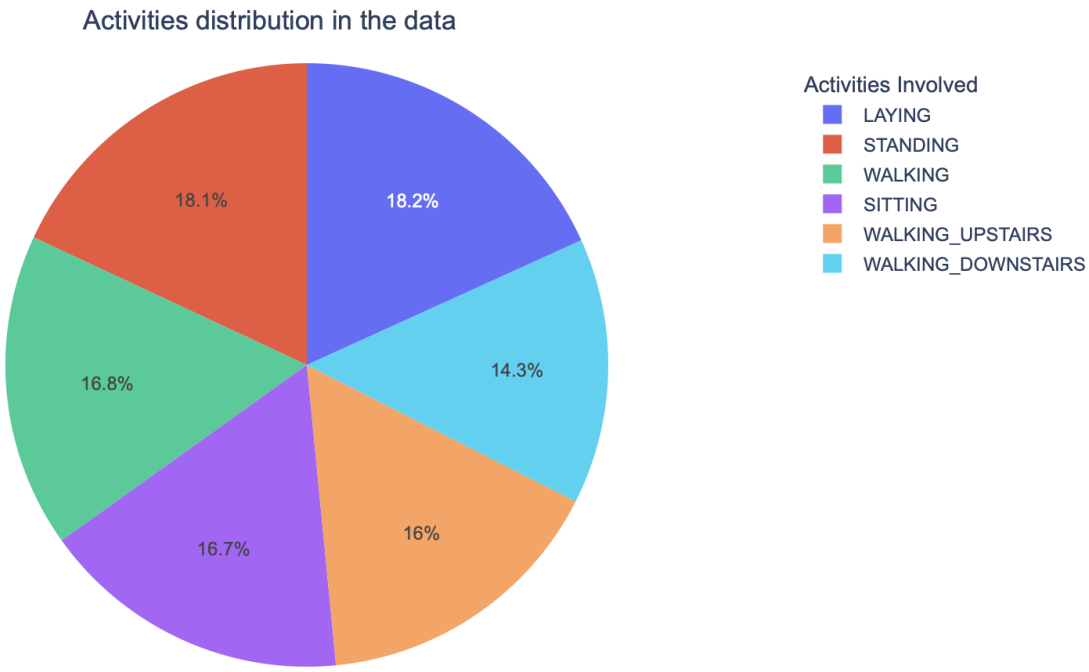


Fig: 4.5

We will not conduct resampling because the data is balanced.

5. IMPLEMENTATION USING MACHINE LEARNING

5.1Unsupervised Learning using T-SNE:

Data visualization following T-SNE dimension reduction:

```
from sklearn.manifold import TSNE
def perform_tsne(X_data, y_data, perplexities, n_iter=1000, img_name_prefix='t-sne'):

    for index,perplexity in enumerate(perplexities):
        # perform t-sne
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)

        # prepare the data for seaborn
        print('Creating plot for this t-sne visualization..')
        df = pd.DataFrame({'x':X_reduced[:,0], 'y':X_reduced[:,1] ,'label':y_data})

        # draw the plot in appropriate place in the grid
        sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,\
                   palette="Set1",markers=['^','v','s','o', '1','2'
        plt.title("perplexity : {} and max_iter : {}".format(perple
        img_name = img_name_prefix + '_perp_{}_iter_{}.png'.format(perplexity, n_iter)
        print('saving this plot as image in present working directory...')
        plt.savefig(img_name)
        plt.show()
        print('Done')
```

(variable) perplexity: Any

```
[ ] X_pre_tsne = df.drop(['subject', 'Activity'], axis=1)
    y_pre_tsne = df['Activity']
    perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[20])
```

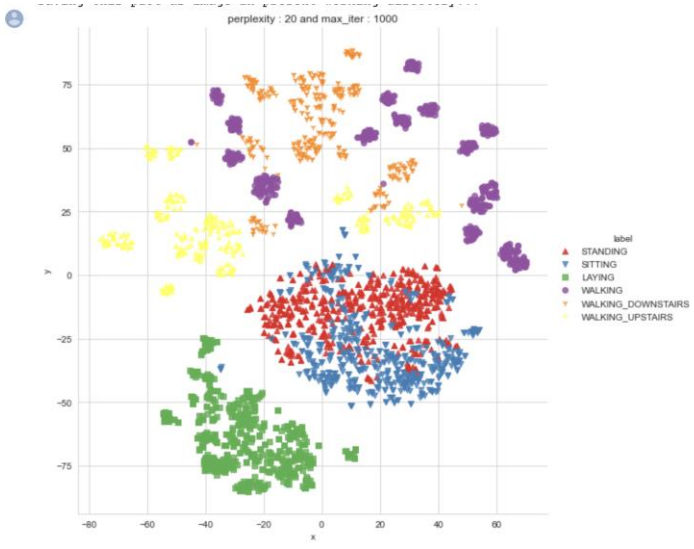


Fig: 5.1

The unsupervised Machine Learning algorithm tSNE was created in 2008 by Laurens van der Maaten and Geoffery Hinton. It is commonly used in bioinformatics and data science in general to visualize the structure of high-dimensional data in 2 or 3 dimensions.

Conversion of Categorical labels into numerical:

Label Encoding:

To transform categorical data into numerical data, we will use the LabelEncoder() function from the Sklearn package. Fit Transform() will be utilized along the procedure.

```
encoder=preprocessing.LabelEncoder()
encoder.fit(y)
y=encoder.transform(y)
y.shape
```

```
(2947,)
```

```
[67] encoder.classes_
```

```
array(['LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS',
       'WALKING_UPSTAIRS'], dtype=object)
```

Fig: 5.0

Aspect_Scaling

```
[ ] scaler=StandardScaler()
    x=scaler.fit_transform(x)
```

Fig: 5.1

Dividing the Training and the Validation sets

```
[ ] x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=100)
    x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Fig: 5.2

5.2 Supervised Learning Models Using:

Using labeled training data, supervised learning trains a function that may later be applied to new samples. The training includes a critic who may assess a function's accuracy before modifying it to produce the desired result. There are many more algorithms, but one of the most well-known ones is the back-propagation algorithm.

Using supervised learning, one may create a function (or model) using labeled training data that comprises input data and a desired output. The desired outcome acts as the control, enabling the function to be changed in response to the actual output it produces. Once trained, one may use this function to brand-new data to produce an output (prediction or classification), which in theory should produce accurate results.

The supervised learning method uses a set of categorized data to produce a model. This simulation can then be used in production with actual data or new data to verify its correctness.

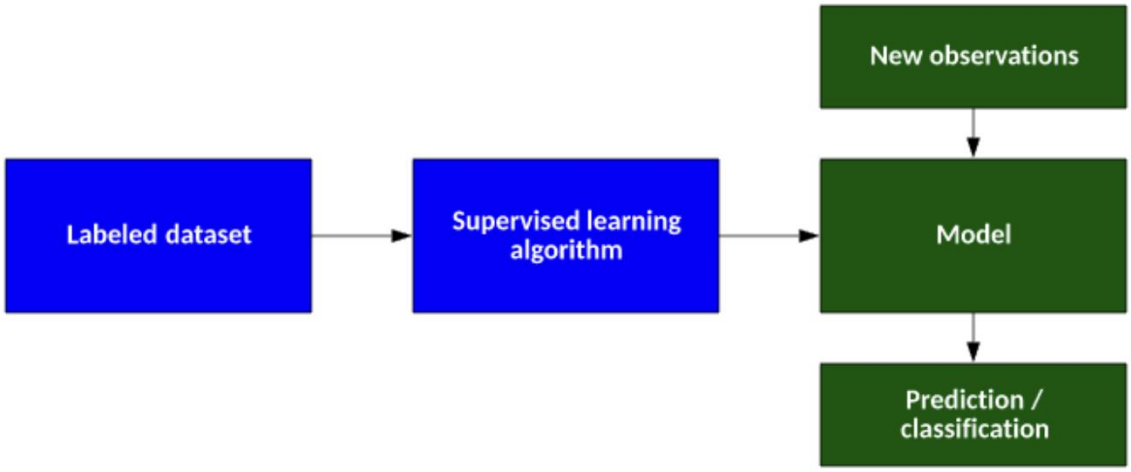


Fig: 6.0

6. MACHINE LEARNING MODELS

In this project we are using:

- SVM
- Random Forest
- Confusion Matrix
- Decision Tree

6.1 Support Vector Machine (SVM):

Support vector machines are strongly advised since they provide notably accurate results while consuming less computing power. Both regression and classification applications can benefit from the usage of SVMs, or Support Vector Machines. It is, nonetheless, widely used to achieve categorization goals. The goal of the SVM algorithm is to locate a hyperplane in an N-dimensional space that distinctly classifies the data points.

Dimensionality

The SVM model separates data differently than other models. Richard Bellman invented the term "the curse of dimensionality," which refers to certain phenomena that emerge with larger dimensional data, to describe the difficulties associated with analyzing such data. The fundamental property that the accuracy fundamentally increases with the number of features is known as the "curse of dimensionality," and it refers to the fact that high-dimensional algorithms are more difficult to build and typically have running times that are exponentially linked to the dimensions. Although storing more data theoretically is possible with a bigger number of dimensions, this seldom works out well since real-world data is more likely to have noise and redundancy.

```
In [29]: svc=SVC()
         svc.fit(x_train,y_train)

Out[29]: SVC()

In [30]: y_pred=svc.predict(x_test)
         print('Score for model correctness using default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))

         Score for model correctness using default hyperparameters: 0.9814
```

Fig: 6.1

several hyperparameters where kernel use rbf and C=100.0

```
In [31]: svc_2=SVC(kernel='rbf',C=100.0)


         # fitting the classifier to the training data
         svc_2.fit(x_train,y_train)

         # making predictions on test set
         y_pred2 = svc_2.predict(x_test)

         # computing and printing the accuracy score
         print('Score of model correctness using rbf kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred2)))

         Score of model correctness using rbf kernel and C=100.0 : 0.9881
```

Fig: 6.2

6.2 Random Forest Classifier:

Random forest is a popular supervised machine learning technique for classification and regression issues. The bulk of the samples are used for classification, and the average of the samples is used for regression, in order to create decision trees. The bootstrap sample, a data sample generated from a training set with replacement, is what makes up each decision tree in the ensemble that makes up the random forest approach.

One of the key characteristics of the Random Forest Algorithm is that it can handle data sets with both continuous variables, as in regression, and categorical variables, as in classification. In terms of categorizing issues, it yields superior outcomes. But among the classifiers, the random forest classifier comes in very high.

```
In [32]: random_clf=RandomForestClassifier(random_state=5)
         random_clf.fit(x_train,y_train)

Out[32]: RandomForestClassifier(random_state=5)

In [33]: # computing and printing the accuracy score
         random_clf.score(x_test,y_test)

Out[33]: 0.9796610169491525
```

Fig: 6.3

6.3 Confusion Matrix:

To explain how effectively a categorization system performs, a confusion matrix is utilized. A confusion matrix displays and rates a classification algorithm's performance. To display important predictive statistics, such as recall, specificity, accuracy, and precision, confusion matrices are utilized. Confusion matrices are useful because they allow for direct comparisons of variables like True Positives, False Positives, True Negatives, and False Negatives.

```
confusion_matrix(y_test, y_pred)
```

```
array([[113,   0,   0,   0,   1,   0],
       [  0,  88,   3,   0,   0,   0],
       [  0,   6, 105,   0,   0,   0],
       [  0,   0,   0,  86,   0,   0],
       [  0,   0,   0,   1,  94,   0],
       [  0,   0,   0,   0,   0,  93]])
```

```
accuracy_score(y_test, y_pred)
```

```
0.9813559322033898
```

```
precision_score(y_test, y_pred, average=None)
```

```
array([1.        , 0.93617021, 0.97222222, 0.98850575, 0.98947368,
       1.        ])
```

```
#calculating the F1 Score
f1_score(y_test, y_pred, average=None)
```

```
array([0.99559471, 0.95135135, 0.95890411, 0.99421965, 0.98947368,
       1.        ])
```

Fig: 6.4

6.4 Decision Tree Classifier:

Decision Trees are supervised learning techniques that may be used to address regression and classification problems as well as function well with classifiers. It is a tree-structured classifier, with each leaf node denoting the classification result and inner nodes denoting the properties of a dataset. The two nodes that make up a decision tree are the Decision Node and the Leaf Node. Decision nodes are used to make any decision and have many branches, whereas Leaf nodes indicate the results of certain decisions and do not have any additional branches.

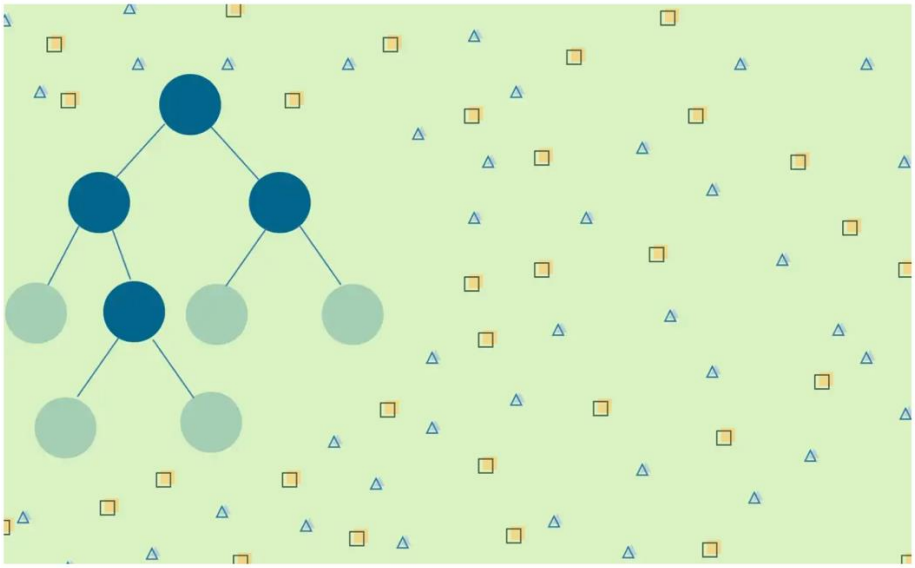


Fig: 6.5

```
dtree=DecisionTreeClassifier()
dtree.fit(x_train,y_train)
```

```
DecisionTreeClassifier()
```

```
# Predicting the values of test data
y_pred = dtree.predict(x_test)
print("Classification report - \n", classification_report(y_test,y_pred))
```

```
Classification report -
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       114
           1       0.84      0.89      0.86        91
           2       0.90      0.86      0.88       111
           3       0.92      0.97      0.94        86
           4       0.94      0.95      0.94        95
           5       0.97      0.91      0.94        93

    accuracy                           0.93       590
   macro avg       0.93      0.93      0.93       590
weighted avg       0.93      0.93      0.93       590
```

7. CONCLUSION AND FUTURE

The performance was improved by the expert-generated features. It demonstrates the significance of domain expertise in obtaining more effective models. Feature generations can help in better detection in future.

8. REFERENCES

1. https://www.ijcseonline.org/spl_pub_paper/ICIPSCMCACGS-19-13.pdf?cv=1
2. https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones