Human Activity Recognition with Machine Learning

Name : NAVEEN M

Reg.No : URK20CS1131

Recognition of human activity is one of the active research areas in machine learning for various contexts such as safety surveillance, healthcare and human-machine interaction. In this article, I will walk you through the task of Human Activity Recognition with machine learning using Python.

Human Activity Recognition

Recognition of human activity is an ability to interpret the gestures or movements of the human body via sensors and to determine human activity or action. Most everyday human tasks can be simplified or automated if they can be recognized through the activity recognizing systems. Generally, the human activity recognition system may or may not be supervised.

The recognition of human activity is mainly used in health systems installed in the residential environment, hospitals and rehabilitation centres. It is widely used to monitor the activities of the elderly staying in rehabilitation centres for chronic disease management and disease prevention.

In the section below, I will take you through a Machine Learning project on Human Activity Recognition using the smartphone data and the Python programming language.

Human Activity Recognition with Machine Learning

Now let’s start the task of Human Activity Recognition with machine learning by importing the necessary Python libraries:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score
```

I import NumPy and pandas to manage tables and datasets. Then matplotlib is included to be used to create visualizations. To use various machine learning algorithms, I import SVM, Logistic Regression, K Nearest Neighbors Classifier, and Random Forest Classifier provided by Scikit-Learn. Also, accuracy_score is included to calculate the accuracy of the Activity Recognition Model.

Now let’s import the smartphone data and have a look at the first five rows from the dataset:

```
from google.colab import files
data_to_load = files.upload()
```

```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to en
Saving train.csv to train.csv
```

```
import io
df = pd.read_csv(io.BytesIO(data_to_load['train.csv']))
```

```
from google.colab import files
data_to_load = files.upload()
```

```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to en
Saving test.csv to test.csv
```

```
import io
df = pd.read_csv(io.BytesIO(data_to_load['test.csv']))
```

```
training_data = pd.read_csv('train.csv')
testing_data = pd.read_csv('test.csv')
```

```
training_data.head()
```

	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	...	fBodyBodyGy
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	

5 rows × 563 columns

There are a total of 7352 records in the training dataset. Also, there are no null values in the dataset. The test dataset contains 2947 records to test our models. This dataset does not have null values.

I can see that the dataset consists of accelerometer and gyro sensor values for each record. In addition, the last two columns are the subject which refers to the subject number and the activity which defines the type of activity. The Activity column will be represented by the label y and all other columns will be represented by X:

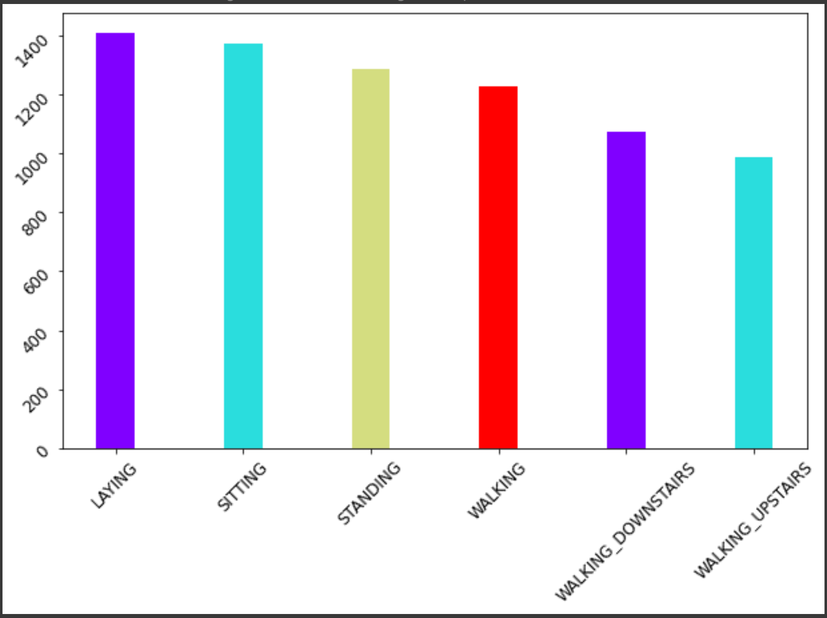
```
# Get X and y for training data
X_train = training_data.drop(columns = ['Activity', 'subject'])
y_train = training_data["Activity"]

# Get X and y for testing data
y_test = testing_data['Activity']
X_test = testing_data.drop(columns = ['Activity', 'subject'])
```

Data Visualization

Now let’s visualize the data to understand some hidden features of the dataset:

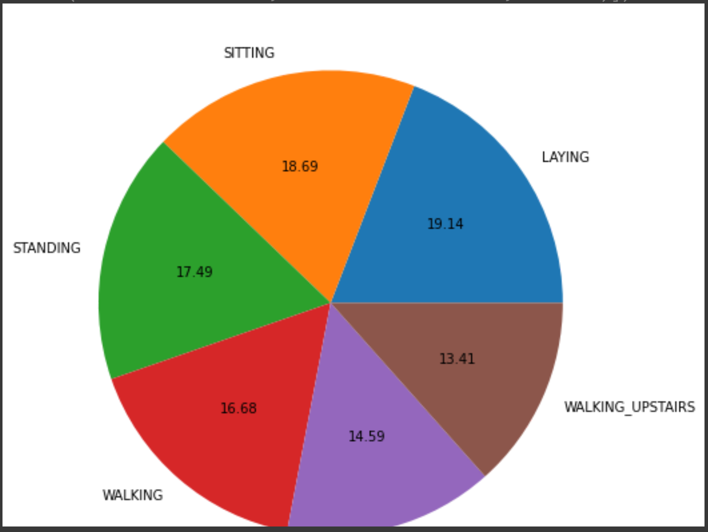
```
count_of_each_activity = np.array(y_train.value_counts())
activities = sorted(y_train.unique())
colors = cm.rainbow(np.linspace(0, 1, 4))
plt.figure(figsize=(10,6))
plt.bar(activities,count_of_each_activity,width=0.3,color=colors)
plt.xticks(rotation=45,fontsize=12)
plt.yticks(rotation=45,fontsize=12)
```

```
(array([   0.,  200.,  400.,  600.,  800., 1000., 1200., 1400., 1600.]),
 <a list of 9 Text major ticklabel objects>)
```



```
plt.figure(figsize=(16,8))
plt.pie(count_of_each_activity, labels = activities, autopct = '%0.2f')
```

```
([<matplotlib.patches.Wedge at 0x7ff8a20dcc90>,
  <matplotlib.patches.Wedge at 0x7ff8a20e8390>,
  <matplotlib.patches.Wedge at 0x7ff8a20e8c10>,
  <matplotlib.patches.Wedge at 0x7ff8a2070750>,
  <matplotlib.patches.Wedge at 0x7ff8a207c3d0>,
  <matplotlib.patches.Wedge at 0x7ff8a207cc90>],
 [Text(0.9071064061014833, 0.6222201925441275, 'LAYING'),
  Text(-0.23874635466468208, 1.073778458591122, 'SITTING'),
  Text(-1.0745883152841482, 0.2350743555872831, 'STANDING'),
  Text(-0.7193129027755119, -0.832219290752544, 'WALKING'),
  Text(0.29301586483507763, -1.0602554894717366, 'WALKING_DOWNSTAIRS'),
  Text(1.0038008332903794, -0.4498709671511826, 'WALKING_UPSTAIRS')],
 [Text(0.4947853124189908, 0.3393928322967968, '19.14'),
  Text(-0.13022528436255384, 0.5856973410497028, '18.69'),
  Text(-0.5861390810640807, 0.12822237577488166, '17.49'),
  Text(-0.3923524924230064, -0.453937794955933, '16.68'),
  Text(0.15982683536458778, -0.5783211760754926, '14.59'),
  Text(0.5475277272492978, -0.2453841639006450 2, '13.41')])
```



The percentage of values shows that the size of the data for each activity is comparable. The dataset is also distributed. By inspecting the dataset, I can see that there are a lot of features.

It is easy to identify that there is an accelerometer, gyroscope, and other values in the data set. I can check everyone's share by plotting a bar graph of each type. Accelerometer values have Acc in them, Gyroscope values have Gyro, and rest can be considered like others:

```
Acc = 0
Gyro = 0
other = 0

for value in X_train.columns:
    if "Acc" in str(value):
        Acc += 1
    elif "Gyro" in str(value):
        Gyro += 1
    else:
        other += 1

plt.figure(figsize=(12,8))
plt.bar(['Accelerometer', 'Gyroscope', 'Others'],[Acc,Gyro,other],color=('r','g','b'))
```

```
<BarContainer object of 3 artists>
```

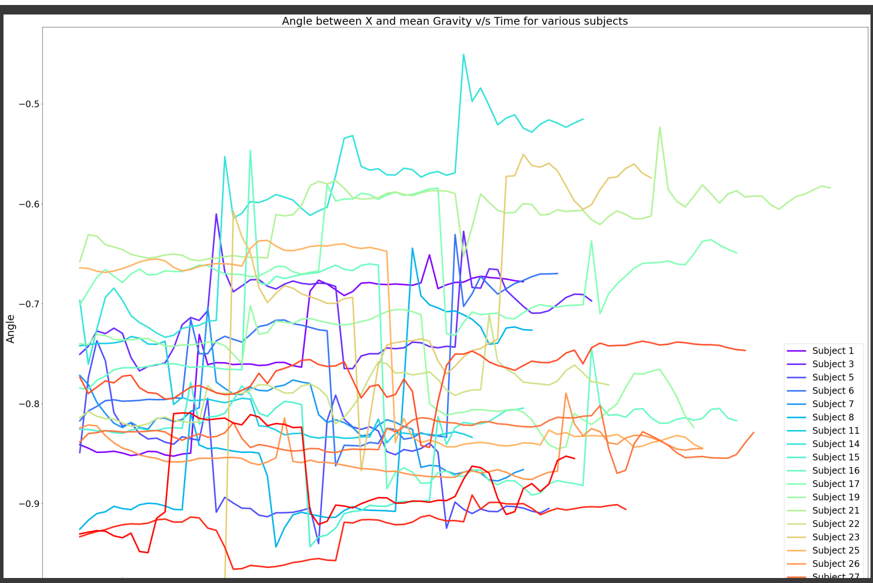
The accelerometer provides the maximum functionality, followed by the gyroscope. The other features are much less so.

```
training_data['subject'].unique()
standing_activity = training_data[training_data['Activity'] == 'STANDING']
# Reset the index for this dataframe
standing_activity = standing_activity.reset_index(drop=True)
time = 1
index = 0
time_series = np.zeros(standing_activity.shape[0])
print(time_series)
```

```
[0. 0. 0. ... 0. 0. 0.]
```

The data collected is in continuous time series for each individual and was recorded at the same rate. So I can just assign time values to each activity starting from 0 whenever the topic changes:

```
for row_number in range(standing_activity.shape[0]):
    if (row_number == 0
        or standing_activity.iloc[row_number]['subject'] == standing_activity.iloc[row_number - 1]['subject']):
        time_series[index] = time
        time += 1
    else:
        time_series[index] = 1
        time = 2
    index += 1

# Combine the time_series with the standing_activity dataframe
time_series_df = pd.DataFrame({ 'Time': time_series })
standing_activity_df = pd.concat([standing_activity, time_series_df], axis = 1)

colors = cm.rainbow(np.linspace(0, 1, len(standing_activity_df['subject'].unique())))

# Create plot for each subject, which will all be displayed overlapping on one plot
id = 0
for subject in standing_activity_df['subject'].unique():
    plt.rcParams.update({'figure.figsize': [40, 30], 'font.size': 24})
    plt.plot(standing_activity_df[standing_activity_df['subject'] == subject]['Time'],
             standing_activity_df[standing_activity_df['subject'] == subject]['angle(X,gravityMean)'],
             c = colors[id],
             label = 'Subject ' + str(subject),
             linewidth = 4)
    plt.xlabel('Time',fontsize=28)
    plt.ylabel('Angle',fontsize=28)
    plt.title('Angle between X and mean Gravity v/s Time for various subjects')
    plt.legend(prop = {'size': 24})
    id += 1
```



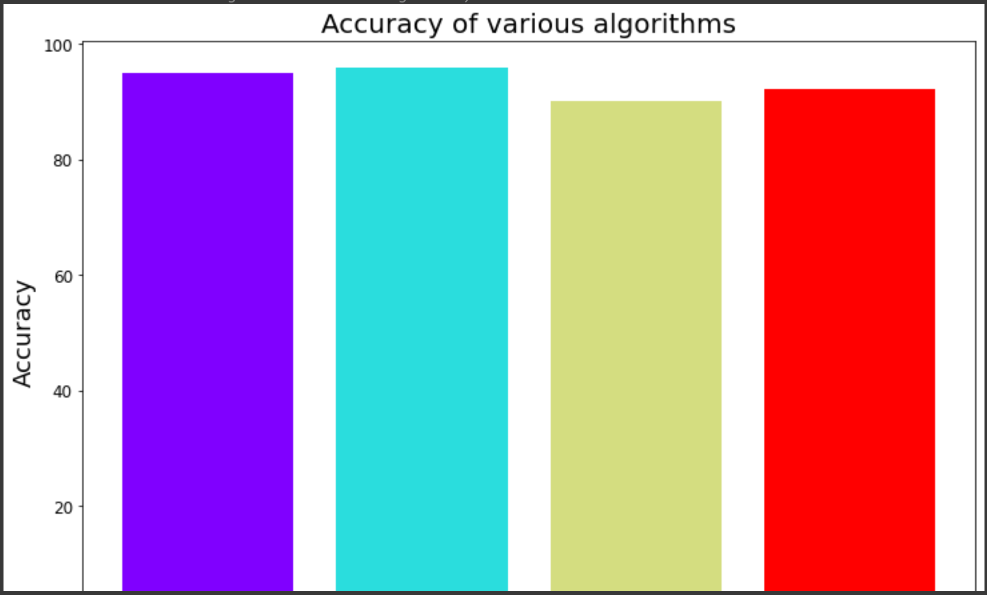
If I take a closer look at the graph, we can see that each row on average transit between a maximum range of 0.2 to 0.3 values. This is indeed the expected behaviour as slight variations can be attributed to minor human errors.

Human Activity Recognition Model with Python:

Now I will train machine learning models for the task of recognizing human activity. Here I will be using various machine learning algorithms available in the Scikit-Learn library in Python that I have already imported. For each algorithm, I'll calculate the accuracy of the prediction and identify the most accurate algorithm:

```
accuracy_scores = np.zeros(4)

# Support Vector Classifier
clf = SVC().fit(X_train, y_train)
prediction = clf.predict(X_test)
accuracy_scores[0] = accuracy_score(y_test, prediction)*100
print('Support Vector Classifier accuracy: {}%'.format(accuracy_scores[0]))

# Logistic Regression
clf = LogisticRegression().fit(X_train, y_train)
prediction = clf.predict(X_test)
accuracy_scores[1] = accuracy_score(y_test, prediction)*100
print('Logistic Regression accuracy: {}%'.format(accuracy_scores[1]))

# K Nearest Neighbors
clf = KNeighborsClassifier().fit(X_train, y_train)
prediction = clf.predict(X_test)
accuracy_scores[2] = accuracy_score(y_test, prediction)*100
print('K Nearest Neighbors Classifier accuracy: {}%'.format(accuracy_scores[2]))

# Random Forest
clf = RandomForestClassifier().fit(X_train, y_train)
prediction = clf.predict(X_test)
accuracy_scores[3] = accuracy_score(y_test, prediction)*100
print('Random Forest Classifier accuracy: {}%'.format(accuracy_scores[3]))

plt.figure(figsize=(12,8))
colors = cm.rainbow(np.linspace(0, 1, 4))
labels = ['Support Vector Classifier', 'Logsitic Regression', 'K Nearest Neighbors', 'Random Forest']
plt.bar(labels,
        accuracy_scores,
        color = colors)
plt.xlabel('Classifiers',fontsize=18)
plt.ylabel('Accuracy',fontsize=18)
plt.title('Accuracy of various algorithms',fontsize=20)
plt.xticks(rotation=45,fontsize=12)
plt.yticks(fontsize=12)
```

```
Support Vector Classifier accuracy: 95.04580929759076%
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (sta
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Logistic Regression accuracy: 95.82626399728538%
K Nearest Neighbors Classifier accuracy: 90.02375296912113%
Random Forest Classifier accuracy: 92.19545300305396%
(array([  0.,  20.,  40.,  60.,  80., 100., 120.]),
 <a list of 7 Text major ticklabel objects>)
```



So we can clearly see that the Logistic Regression model performs the best for the task of Human Activity Recognition with Machine Learning. I hope you liked this article on Machine Learning Project on Recognizing Human Activities with Python. Feel free to ask your valuable questions in the comments section below.