Assignment Project Report

Human Activity Recognition from Smart Phone Data

Name: Arun Govind
Course: AI and ML
(Batch 4)

- **Problem Statement**

  Perform activity recognition on the dataset using a hidden markov model. Then perform the same task using a different classification algorithm (logistic regression/decision tree) of your choice and compare the performance of the two algorithms.

- **Prerequisites**

  - Software:
    - Python 3 (Use anaconda as your python distributor as well)

  - Tools:
    - Numpy
    - Pandas
    - Seaborn
    - Matplotlib
    - Sklearn
    - Hmmlearn

  - Dataset: Human Activity Recognition with smartphone dataset from Kaggle

- **Method Used**

  The HMM is based on augmenting the Markov chain. A Markov chain is a model Markov chain that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. The states before the current state have no impact on the future except via the current state. It’s as if to predict tomorrow’s weather you could examine today’s weather but you weren’t allowed to look at yesterday’s weather.

Recognizing human activities from temporal streams of sensory data observations is a very important task on a wide variety of applications in context recognition. Human activities are hierarchical in nature, i.e. the complex activities can be decomposed to several simpler ones. Human activity recognition is the problem of classifying sequences of accelerometer data recorded by pre-installed sensors in smart phones into known well-defined movements to make it ready for predictive modelling.

- **Implementation:**

1. Load all required libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from hmmlearn import hmm
import warnings
warnings.filterwarnings("ignore")
```

2. Calling test and train dataset

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

```
final_test = test.copy()
```

```
train.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	...	fBodyBodyGyroJerkMag-kurtosis()	angle(tB
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.710304	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	-0.861499	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.760104	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.482845	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	-0.699205	

5 rows × 563 columns

3. Implementing One Hot encoding

```
y_train.replace(to_replace='WALKING',value=1,inplace=True)
y_train.replace(to_replace='WALKING_UPSTAIRS',value=2,inplace=True)
y_train.replace(to_replace='WALKING_DOWNSTAIRS',value=3,inplace=True)
y_train.replace(to_replace='SITTING',value=4,inplace=True)
y_train.replace(to_replace='STANDING',value=5,inplace=True)
y_train.replace(to_replace='LAYING',value=6,inplace=True)
```

```
y_test.replace(to_replace='WALKING',value=1,inplace=True)
y_test.replace(to_replace='WALKING_UPSTAIRS',value=2,inplace=True)
y_test.replace(to_replace='WALKING_DOWNSTAIRS',value=3,inplace=True)
y_test.replace(to_replace='SITTING',value=4,inplace=True)
y_test.replace(to_replace='STANDING',value=5,inplace=True)
y_test.replace(to_replace='LAYING',value=6,inplace=True)
```

4. Code for confusion matrix

```
def plot_confusion_matrix(cm,lables):
    fig, ax = plt.subplots(figsize=(12,8)) # for plotting confusion matrix as image
    im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.YlOrBr)
    ax.figure.colorbar(im, ax=ax)
    ax.set(xticks=np.arange(cm.shape[1]),
    yticks=np.arange(cm.shape[0]),
    xticklabels=lables, yticklabels=lables,
    ylabel='True label',
    xlabel='Predicted label')
    plt.xticks(rotation = 90)
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, int(cm[i, j]),ha="center", va="center",color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
```

5. Implementing HMM algorithm

```
model = hmm.GaussianHMM(n_components=6)
model.fit(X_train_new)
```

```
GaussianHMM(n_components=6)
```

```
predictions = model.predict(X_test_new)
```

```
accuracy_logistic = accuracy_score(y_true = y_test, y_pred = predictions)
print("Accuracy is : ", accuracy_logistic)
```

```
Accuracy is :  0.27078384798099764
```

6. Implementing Logistic regression

```
model = LogisticRegression()
model.fit(X_train_new, y_train)
```

```
LogisticRegression()
```

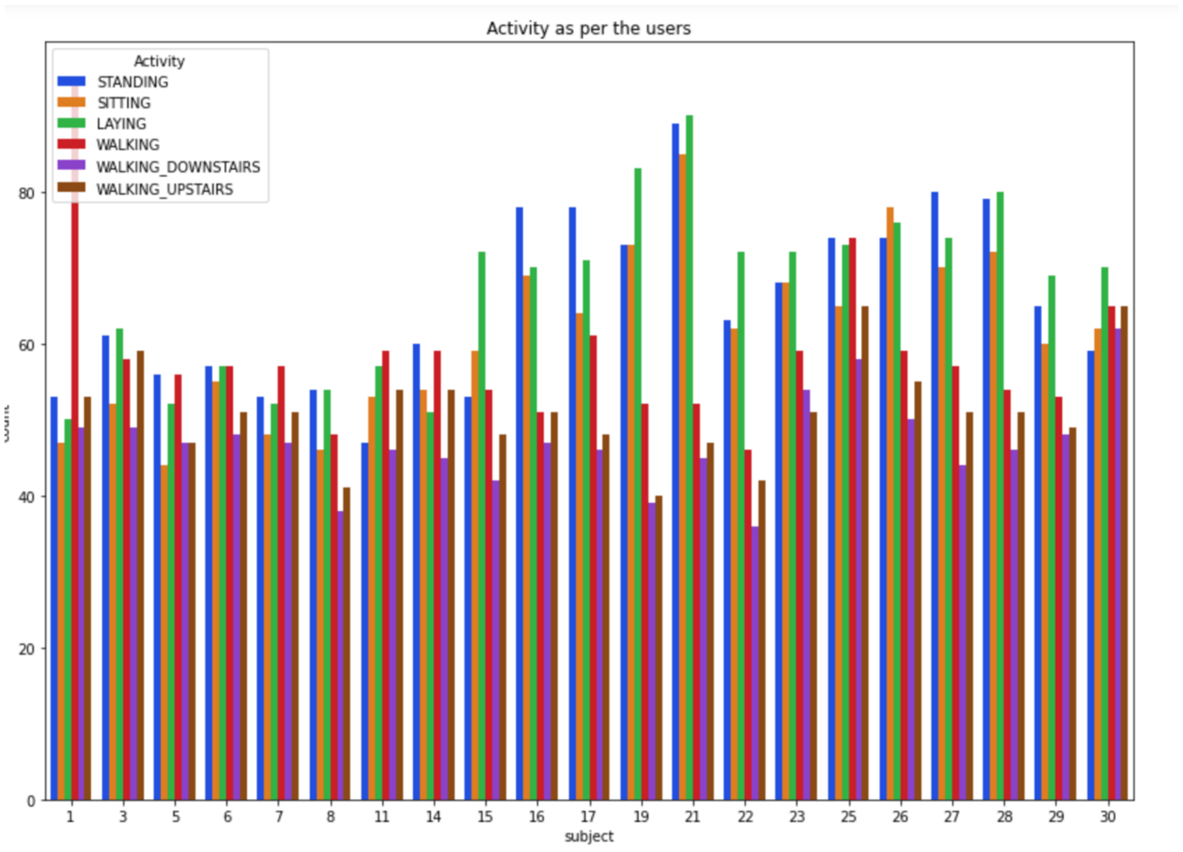
```
predictions = model.predict(X_test_new)
```

```
accuracy_logistic = accuracy_score(y_true = y_test, y_pred = predictions)
print("Accuracy is : ", accuracy_logistic)
```

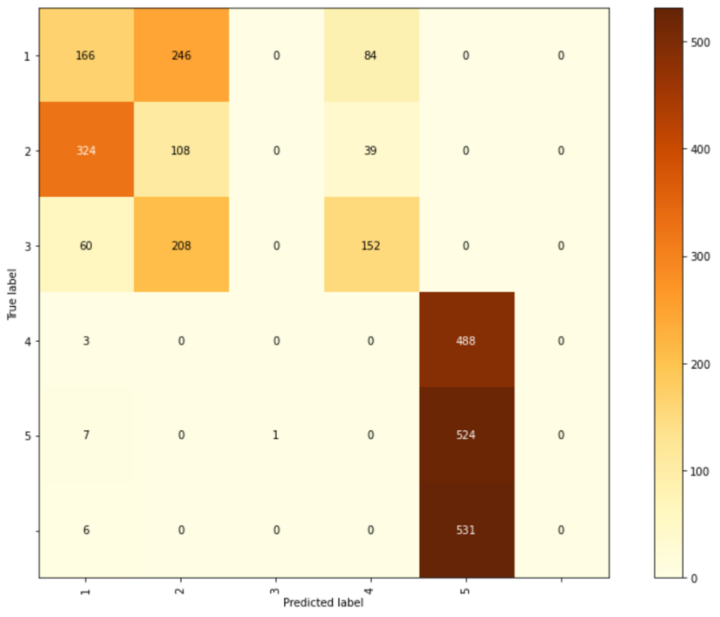
```
Accuracy is :  0.9613165931455717
```

- **Results:**

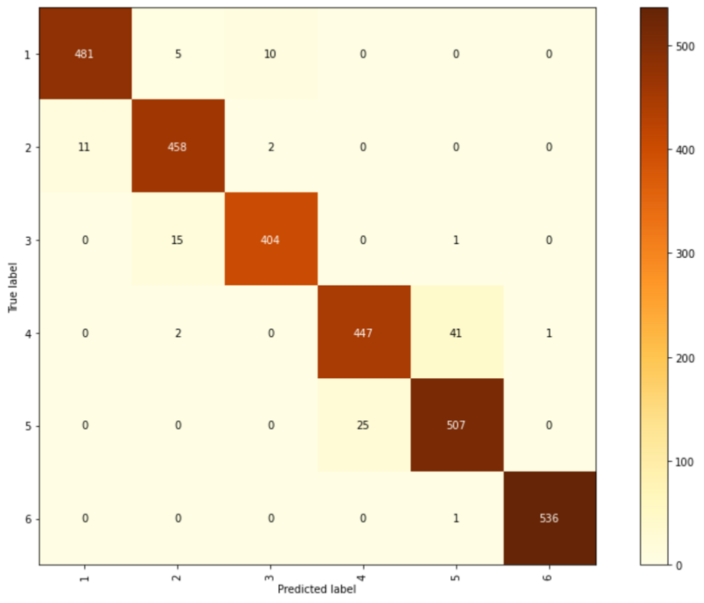
1. Resulting graph of different activities performed:



2. Confusion matrix for HMM algorithm



3. Confusion matrix for Logistic Regression


True label
Predicted label