Capstone Project Report

Name: Pooja Ramdas Kadam
Course: AI & 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

Along with Python below packages needed to be installed

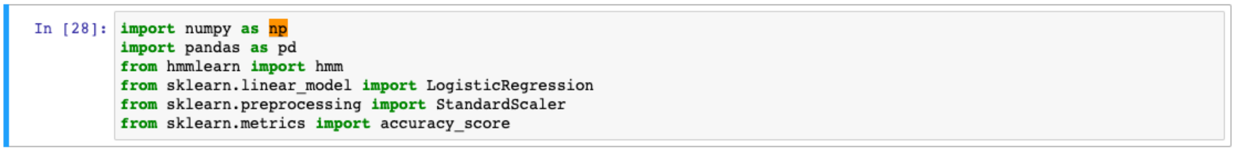
hmmlearn
Sklearn
Pandas

Dataset Used

https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones

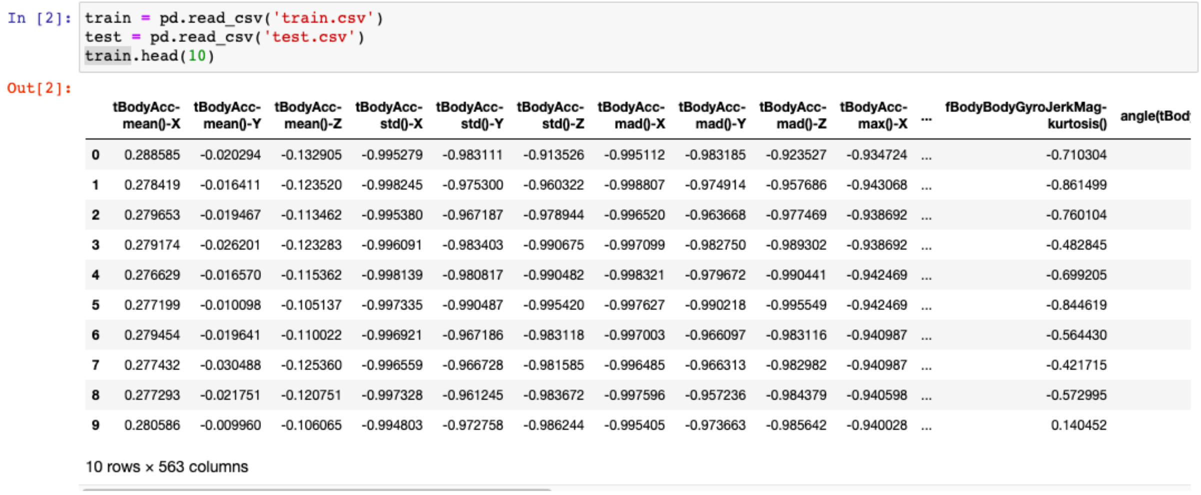
Implementation

Import required libraries and load data



```
In [28]: import numpy as np
         import pandas as pd
         from hmmlearn import hmm
         from sklearn.linear_model import LogisticRegression
         from sklearn.preprocessing import StandardScaler
         from sklearn.metrics import accuracy_score
```

Load data



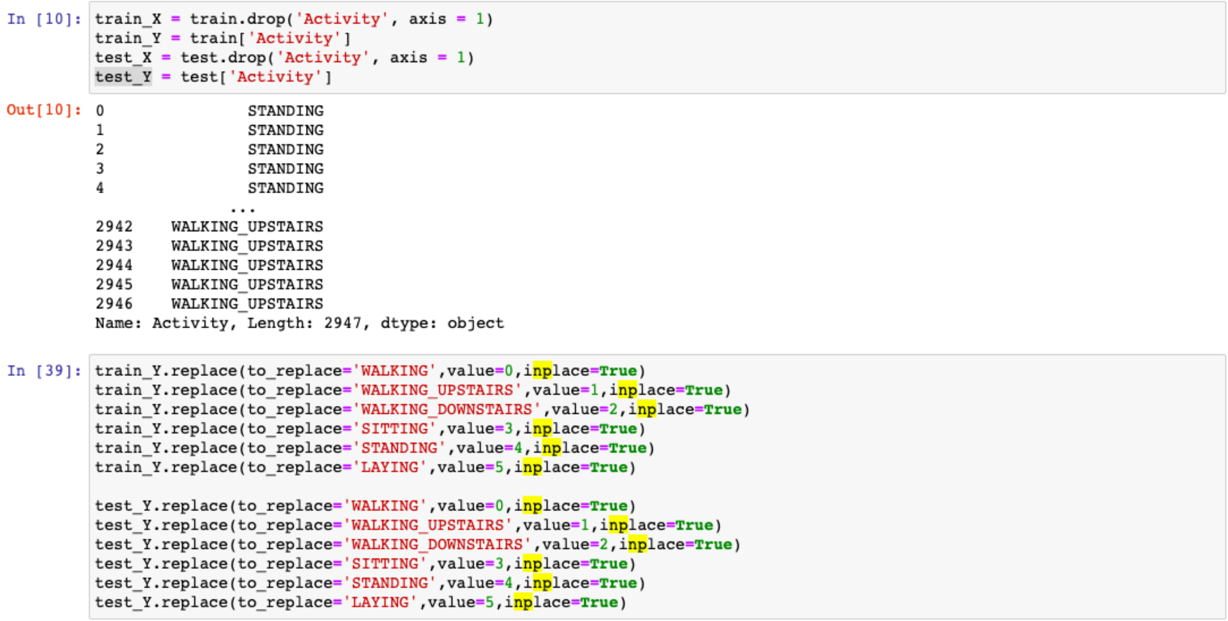
```
In [2]: train = pd.read_csv('train.csv')
        test = pd.read_csv('test.csv')
        train.head(10)
```

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(tBod
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	0.277199	-0.010098	-0.105137	-0.997335	-0.990487	-0.995420	-0.997627	-0.990218	-0.995549	-0.942469	...	-0.844619	
6	0.279454	-0.019641	-0.110022	-0.996921	-0.967186	-0.983118	-0.997003	-0.966097	-0.983116	-0.940987	...	-0.564430	
7	0.277432	-0.030488	-0.125360	-0.996559	-0.966728	-0.981585	-0.996485	-0.966313	-0.982982	-0.940987	...	-0.421715	
8	0.277293	-0.021751	-0.120751	-0.997328	-0.961245	-0.983672	-0.997596	-0.957236	-0.984379	-0.940598	...	-0.572995	
9	0.280586	-0.009960	-0.106065	-0.994803	-0.972758	-0.986244	-0.995405	-0.973663	-0.985642	-0.940028	...	0.140452	

10 rows × 563 columns

Prepare Data to get X and Y



```
In [10]: train_X = train.drop('Activity', axis = 1)
         train_Y = train['Activity']
         test_X = test.drop('Activity', axis = 1)
         test_Y = test['Activity']

Out[10]: 0               STANDING
         1               STANDING
         2               STANDING
         3               STANDING
         4               STANDING
                       ...
         2942    WALKING_UPSTAIRS
         2943    WALKING_UPSTAIRS
         2944    WALKING_UPSTAIRS
         2945    WALKING_UPSTAIRS
         2946    WALKING_UPSTAIRS
         Name: Activity, Length: 2947, dtype: object
```

```
In [39]: train_Y.replace(to_replace='WALKING',value=0,inplace=True)
         train_Y.replace(to_replace='WALKING_UPSTAIRS',value=1,inplace=True)
         train_Y.replace(to_replace='WALKING_DOWNSTAIRS',value=2,inplace=True)
         train_Y.replace(to_replace='SITTING',value=3,inplace=True)
         train_Y.replace(to_replace='STANDING',value=4,inplace=True)
         train_Y.replace(to_replace='LAYING',value=5,inplace=True)

         test_Y.replace(to_replace='WALKING',value=0,inplace=True)
         test_Y.replace(to_replace='WALKING_UPSTAIRS',value=1,inplace=True)
         test_Y.replace(to_replace='WALKING_DOWNSTAIRS',value=2,inplace=True)
         test_Y.replace(to_replace='SITTING',value=3,inplace=True)
         test_Y.replace(to_replace='STANDING',value=4,inplace=True)
         test_Y.replace(to_replace='LAYING',value=5,inplace=True)
```

Apply HMM

```
In [12]: model = hmm.GaussianHMM(n_components=6)
         model.fit(train_X)
         h = model.predict(test_X)
```

```
In [18]: total = 0
         correct = 0
         for i in range(test_Y.shape[0]):
             print(test_Y[i], h[i])
             if test_Y[i] == h[i]:
                 correct = correct + 1
             total = total + 1
```

Apply Logistic Regression

```
In [40]: scaler = StandardScaler()
         train_X = scaler.fit_transform(train_X)
         test_X = scaler.fit_transform(test_X)
```

```
In [41]: model = LogisticRegression()
         model.fit(train_X, train_Y)
```

```
/usr/local/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs fai
led to converge (status=1):
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
  n_iter_i = _check_optimize_result(
```

```
Out[41]: LogisticRegression()
```

```
In [42]: y_pred = model.predict(test_X)
```

```
In [43]: print("Accuracy score : ", accuracy_score(y_true = test_Y, y_pred=y_pred))

         Accuracy score :  0.9602986087546658
```