Capstone Project Report

Name: Sumit Patel

Course: AI and ML (Batch – AUG 2020)

Duration: 10 months

HMM for Human Activity Recognition

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

What things you need to install the software and how to install them:

Python 3.6 This setup requires that your machine has latest version of python. The following url https://www.python.org/downloads/ can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-externalcommand/. Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic. Second and easier option is to download anaconda and use its anaconda prompt to run the commands.

To install anaconda check this url https://www.anaconda.com/download/ You will also need to download and install below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.6 then run below commands in command prompt/terminal to install these packages:

pip install numpy

pip install pandas

pip install matplotlib

pip install sklearn

pip install hmmlearn

If you have chosen to install anaconda then run below commands in anaconda prompt to install these packages:

conda install -c anaconda numpy

conda install -c anaconda pandas

conda install -c anaconda matplotlib

conda install -c anaconda sklearn

conda install -c anaconda hmmlearn

Dataset used:

Dataset Link: Human Activity Recognition with Smartphones https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones

Importing the libraries and loading dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import f1_score, accuracy_score
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from hmmlearn import hmm
```

PCA to reduce the number of features

```
pca = PCA(n_components = 0.98)
x_train_d = pca.fit_transform(x_train)
x_test_d = pca.fit_transform(x_test)
x_train_d.shape, x_test_d.shape
```

```
((7352, 69), (2947, 78))
```

```
# Number of components used from above
pca = PCA(n_components = 80)
fit = pca.fit(x_train)
```

```
x_train_d = fit.transform(x_train)
x_test_d = fit.transform(x_test)
x_train_d.shape, x_test_d.shape
```

```
((7352, 80), (2947, 80))
```

HMM Model Training

```
# Hidden Markov Model
hmm_train = pd.DataFrame(x_train_d)
hmm_train['Activity'] = train['Activity']
hmm_test = pd.DataFrame(x_test_d)
hmm_test['Activity'] = test['Activity']
```

```
hmm_train_STAND = hmm_train[hmm_train['Activity']=='STANDING']
hmm_train_SIT = hmm_train[hmm_train['Activity']=='SITTING']
hmm_train_LAY = hmm_train[hmm_train['Activity']=='LAYING']
hmm_train_WALK = hmm_train[hmm_train['Activity']=='WALKING']
hmm_train_WALKD = hmm_train[hmm_train['Activity']=='WALKING_DOWNSTAIRS']
hmm_train_WALKU = hmm_train[hmm_train['Activity']=='WALKING_UPSTAIRS']
```

```python
# Calculate true labels
labels_test = []
for j in range(len(hmm_test)):
    if (hmm_test['Activity'].iloc[j]=='STANDING'):
        labels_test.append(0)
    elif (hmm_test['Activity'].iloc[j]=='SITTING'):
        labels_test.append(1)
    elif (hmm_test['Activity'].iloc[j]=='LAYING'):
        labels_test.append(2)
    elif (hmm_test['Activity'].iloc[j]=='WALKING'):
        labels_test.append(3)
    elif (hmm_test['Activity'].iloc[j]=='WALKING_DOWNSTAIRS'):
        labels_test.append(4)
    else:
        labels_test.append(5)
labels_test = np.array(labels_test)
labels_test.shape
```

```
(2947,)
```

```python
# Implementing HMM
# Fitting for each activity
def HMM_F1score(N,M,labels_true):
    hmm_stand = hmm.GMMHMM(n_components = N, n_mix = M, covariance_type = 'diag')
    hmm_sit = hmm.GMMHMM(n_components = N, n_mix = M, covariance_type = 'diag')
    hmm_lay = hmm.GMMHMM(n_components = N, n_mix = M, covariance_type = 'diag')
    hmm_walk = hmm.GMMHMM(n_components = N, n_mix = M, covariance_type = 'diag')
    hmm_walk_d = hmm.GMMHMM(n_components = N, n_mix = M, covariance_type = 'diag')
    hmm_walk_u = hmm.GMMHMM(n_components = N, n_mix = M, covariance_type = 'diag')

    hmm_stand.fit(hmm_train_STAND.iloc[:,0:80].values)
    hmm_sit.fit(hmm_train_SIT.iloc[:,0:80].values)
    hmm_lay.fit(hmm_train_LAY.iloc[:,0:80].values)
    hmm_walk.fit(hmm_train_WALK.iloc[:,0:80].values)
    hmm_walk_d.fit(hmm_train_WALKD.iloc[:,0:80].values)
    hmm_walk_u.fit(hmm_train_WALKU.iloc[:,0:80].values)

    # Calculating F1_Score
    labels_predict = []
    for i in range(len(hmm_test)):
        log_likelihood_value = np.array([hmm_stand.score(hmm_test.iloc[i,0:80].values.reshape(1,80)),
                                         hmm_sit.score(hmm_test.iloc[i,0:80].values.reshape(1,80)),
                                         hmm_lay.score(hmm_test.iloc[i,0:80].values.reshape(1,80)),
                                         hmm_walk.score(hmm_test.iloc[i,0:80].values.reshape(1,80)),
                                         hmm_walk_d.score(hmm_test.iloc[i,0:80].values.reshape(1,80)),
                                         hmm_walk_u.score(hmm_test.iloc[i,0:80].values.reshape(1,80))])
        labels_predict.append(np.argmax(log_likelihood_value))
    labels_predict = np.array(labels_predict)

    F1 = f1_score(labels_true, labels_predict, average = 'micro')
    acc = accuracy_score(labels_true, labels_predict)
    return F1,acc
```

```
score = pd.DataFrame([np.array(F1_value_states), np.array(acc_value_states)])
score
```

	0	1	2	3	4	5	6	7	8	9	...	25	26	
0	0.873431	0.744825	0.728877	0.710214	0.592806	0.518154	0.525619	0.543604	0.468273	0.475399	...	0.366474	0.374618	0.364
1	0.873431	0.744825	0.728877	0.710214	0.592806	0.518154	0.525619	0.543604	0.468273	0.475399	...	0.366474	0.374618	0.364

2 rows × 35 columns

Prediction using Neural Network

```
# encode class values as integers

encoder = LabelEncoder()
encoder.fit(y_train)
encoded_y_train = encoder.transform(y_train)
encoded_y_test = encoder.transform(y_test)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y_train = np_utils.to_categorical(encoded_y_train)
dummy_y_test = np_utils.to_categorical(encoded_y_test)
```

```
print(encoded_y_train.shape, encoded_y_test.shape)
encoded_y_train, encoded_y_test
```

```
(7352,) (2947,)

(array([2, 2, 2, ..., 5, 5, 5]), array([2, 2, 2, ..., 5, 5, 5]))
```

```
dummy_y_train[:5], dummy_y_test[1000:1005]
```

```
(array([[0., 0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0.]], dtype=float32),
 array([[0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0., 0.]], dtype=float32))
```

```
# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(128, input_shape = (80,), activation='relu'))
    model.add(Dense(6, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
model = baseline_model()
```

```python
y_pred = np.round(model.predict(x_test_d))
y_pred
```

```
array([[0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```python
from sklearn.metrics import classification_report
target_names = ['LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS', 'WALKING_UPSTAIRS']
print(classification_report(dummy_y_test, y_pred, target_names = target_names))
```

```
                    precision    recall  f1-score   support

            LAYING       1.00      0.97      0.98       537
           SITTING       0.95      0.84      0.89       491
          STANDING       0.85      0.95      0.90       532
           WALKING       0.90      0.98      0.94       496
WALKING_DOWNSTAIRS       0.97      0.97      0.97       420
  WALKING_UPSTAIRS       0.97      0.86      0.91       471

         micro avg       0.93      0.93      0.93      2947
         macro avg       0.94      0.93      0.93      2947
      weighted avg       0.94      0.93      0.93      2947
       samples avg       0.93      0.93      0.93      2947
```