Machine Learning Engineer Nanodegree

Capstone Project

Amol Pol
August 4th, 2019

I. Definition

Project Overview

Human Activity Recognition(HAR) Using Deep Learning

This project is to build a model that predicts the human activities such as Walking, Walking_Upstairs, Walking_Downstairs, Sitting, Standing or Laying.

Human activity recognition is the problem of classifying sequences of accelerometer data recorded into known well-defined movements. By automatically monitoring human activities, one can find many applications like security surveillance, healthcare, logistics support to location based services, wildlife observation, energy conservation etc.

It is a challenging problem given the large number of observations produced each second and lack of a clear way to relate accelerometer data to known movements.
One could devise a technique for HAR using classical machine learning. The performance(accuracy) of such methods largely depends on hand crafting features or good feature extraction methods. The difficulty is that this feature engineering requires deep expertise in the domain field.
Deep learning methods such as recurrent neural networks can also be tried on this challenging activity recognition tasks which would not require any feature engineering.

There have been many academic research on this problem of human activity recognition. I am highlighting a few of them over here –

- https://www.sciencedirect.com/science/article/pii/S1877050914008643
- https://www.hindawi.com/journals/js/2018/8580959/
- https://www.hindawi.com/journals/wcmc/2018/2618045/

Problem Statement

The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors.

So, in short the problem statement is a multi-class classification problem. We are given sensor readings of 30 study participants. These are accelerometer and gyroscope sensor readings. And our task is to predict the activity that the person is performing. These activities (class labels) are –

- Walking
- Walking_Upstairs
- Walking_Downstairs
- Sitting

- Standing
- Laying

The dataset contains 2 .csv files and raw time series of accelerometer and gyroscope sensor data with all the information necessary to make predictions.

1. Train.csv and test.csv contents:
    a. Columns 1 to column 561 are expert engineered features.
    b. The last column 564 is the activity name(output).
2. Train.csv contains 7352 data points.
3. Test.csv contains 2947 data points.

Data usage:

1. The above csv files will be used to make predictions using classical machine learning approach – Random Forest an ensembled method.
2. Later half of the project, I will make use of accelerometer and gyroscope sensor data which will be directly fed to the RNN model(LSTM) to make predictions.

Metrics

The problem is a multi-class classification problem. Also, it has been established earlier in the capstone proposal and will be established again in the Data Exploration section under Analysis in the report that the dataset is almost well balanced. Hence, I choose accuracy as my evaluation metrics. Of course, I will also look at multi-class confusion matrix and we will also look at multi-class log-loss. The confusion matrix tells us what types of confusions are happening. For ex. Confusing between sitting and standing up. So, confusion matrix is a very important way of understanding for which classes the model is doing well and for which classes is it getting confused. So, final conclusions will be made using the above metrics – accuracy, multi-class confusion matrix and multi-class log-loss with accuracy playing the major role.

II. Analysis

Data Exploration

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

Exploration of Data for Classical Machine Learning Approach

Training data

```
In [16]: # get the data from txt files to pandas dataffame
X_train = pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, header=None, names=features)

# add subject column to the dataframe
X_train['subject'] = pd.read_csv('UCI_HAR_dataset/train/subject_train.txt', header=None, squeeze=True)

y_train = pd.read_csv('UCI_HAR_dataset/train/y_train.txt', names=['Activity'], squeeze=True)
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                              4:'SITTING', 5:'STANDING',6:'LAYING'})

# put all columns in a single dataframe
train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.head(3)

/Users/ampol/Virtualenvs/ai/lib/python3.6/site-packages/pandas/io/parsers.py:702: UserWarning: Duplicate names specified. This will raise an error in the future.
  return _read(filepath_or_buffer, kwds)
```

Out[16]:

	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	...	angle(tBodyAccMean,gravity)	angle
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.112754	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.053477	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.118559	

3 rows × 564 columns

```
In [17]: train.shape
Out[17]: (7352, 564)
```

We have 7352 data points in our training data.

Test data

```
In [9]: # get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_dataset/test/X_test.txt', delim_whitespace=True, header=None, names=features)

# add subject column to the dataframe
X_test['subject'] = pd.read_csv('UCI_HAR_dataset/test/subject_test.txt', header=None, squeeze=True)

# get y labels from the txt file
y_test = pd.read_csv('UCI_HAR_dataset/test/y_test.txt', names=['Activity'], squeeze=True)
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                            4:'SITTING', 5:'STANDING',6:'LAYING'})


# put all columns in a single dataframe
test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels
test.head()
```

Out[9]:

	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	...	angle(tBodyAccMean,gravity)	angle
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249	-0.674302	-0.894088	...	0.006462	
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401	-0.945823	-0.894088	...	-0.083495	
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735	-0.963483	-0.939260	...	-0.034956	
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471	-0.968897	-0.938610	...	-0.017067	
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953	-0.977346	-0.938610	...	-0.002223	

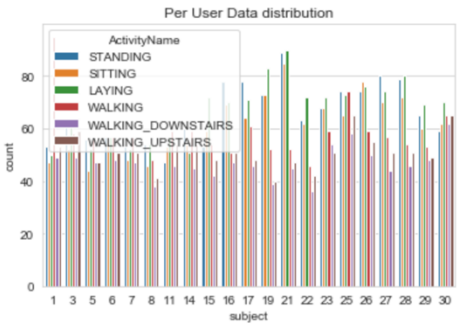
5 rows × 564 columns

```
In [18]: test.shape
Out[18]: (2947, 564)
```

We have 2947 data points in testing data.
Also, shown above few data points from test as well as training data using train.head and test.head as sample.

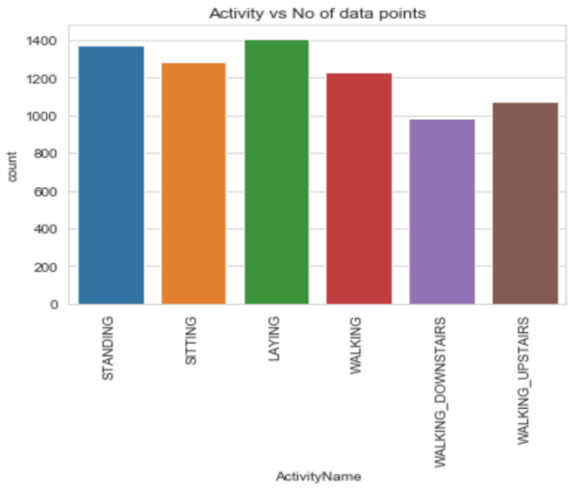
Check for data imbalance

```
In [14]: sns.set_style('whitegrid')
plt.title('Per User Data distribution')
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()
```



We have got almost same number of data points from all the subjects

```
In [15]: plt.title('Activity vs No of data points')
         sns.countplot(train.ActivityName)
         plt.xticks(rotation=90)
         plt.show()
```



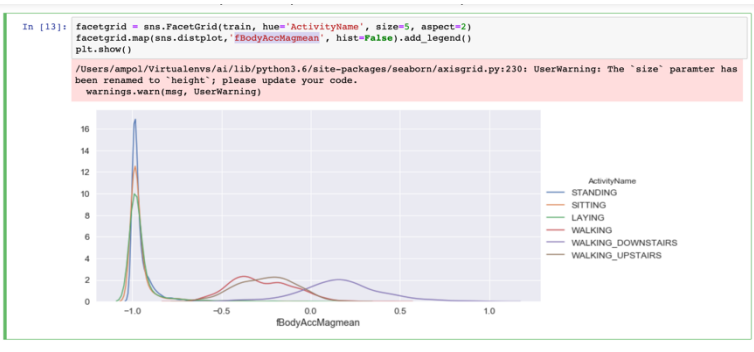
Our data is almost well balanced across all class labels.
This dataset is provided by Kaggle and UCI Machine Learning Repository and can be obtained here.

Exploration of Raw Time Series Data for Deep Learning Model – RNN (LSTM)

There are three main signal types in the raw data: total acceleration, body acceleration and body gyroscope. Each has 3 axises of data. This means that there are a total of nine variables for each time step.
Further, each series of data has been partitioned into overlapping windows of 2.56 seconds of data or 128 time steps. These windows of data correspond to the windows of engineered features as described earlier.
So one row of data has (128*9) = 1152 elements.
The signals are stored in the /Inertial Signals/ directory under train and test subdirectories. Each axis of each signal is stored in a separate file, meaning that each of the train and test datasets have nine input files to load and one output file to load.

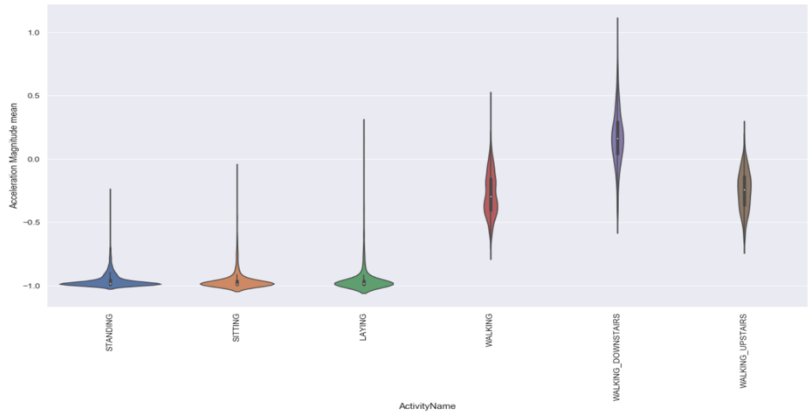
Exploratory Visualization

I started the EDA with univariate analysis. I divided the activities into 2 types of activities - Stationary and Moving activities.
Stationary activities – ‘Standing’, ‘Sitting’, ‘Laying’
Moving activities – ‘Walking’, ‘Walking_upstairs’, ‘Walking_downstairs’
The first feature that I took was ‘fBodyAccMagmean’. I find that there is a clear distinction between the Stationary and Moving activities.



Observations: The above graph shows that activities like Standing, Sitting, and Laying can be easily separated by ‘fBodyAccMagmean’ feature. Its a feature formed by applying fast fourier transformation to accelerometer reading.

In [14]:
```
plt.figure(figsize=(20, 10))
sns.violinplot(x='ActivityName', y='fBodyAccMagmean',data=train)
plt.ylabel('Acceleration Magnitude mean')
plt.xticks(rotation=90)
plt.show()
```



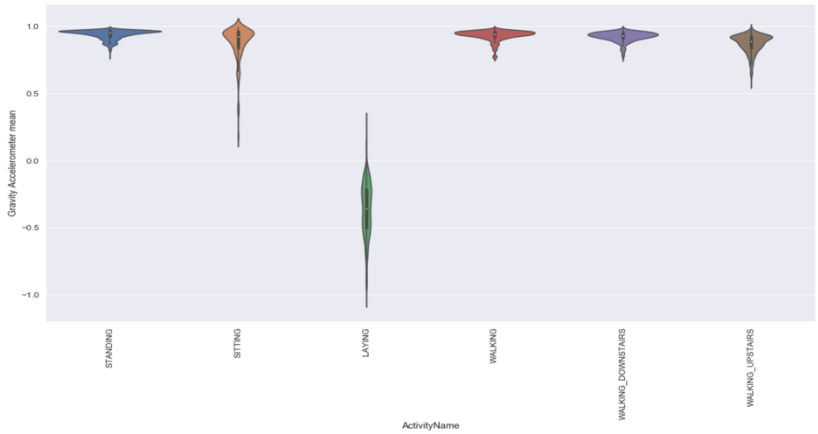
Observations:

1. If fBodyAccMagmean is < -0.9 then the activities are either Standing or Sitting or Laying.
2. If fBodyAccMagmean is > -0.5 then the activities are either Walking or WalkingDownstairs or WalkingUpstairs.
3. If fBodyAccMagmean > 0.1 then the activity is WalkingDownstairs.

With the help of above if else conditions we can classify data points with some errors.

In the above figure you can see that, I tried a violin plot on feature ‘fBodyAccMagmean’ and I found quite a clear separation. I can put some threshold say -0.9 and then I can say that the given data belongs to ‘Standing’ or ‘Sitting’ or ‘Laying’ and if greater than say -0.5 then it belongs to ‘Walking’ or ‘WalkingDownStairs’ or ‘WalkingUpstairs’ and if greater than 0.1 then the activity us ‘WalkingDownstairs’. I could build a simple rule based classification system. So, I can classify fairly well even with simple rule based classification with just a single feature. This shows that these features are well designed by the experts and well thought through.

In [15]:
```
plt.figure(figsize=(20, 10))
sns.violinplot(x='ActivityName', y='tGravityAccmeanX',data=train)
plt.ylabel('Gravity Accelerometer mean')
plt.xticks(rotation=90)
plt.show()
```



Observations:

1. If tGravityAccmeanX < 0 then the activity is laying.

We can see that, using only "tGravityAccmeanX" feature we can say whether this belongs to Laying activity or not.

I tried many other features for univariate analysis, some of them weren’t useful, whereas some were. For ex, for other feature ‘tGravityAccMeanX’, I could see that there is a particular threshold say 0 which can distinguish data points belonging to laying down activity to other features.
So, to conclude, I can say that we can design a fairly well system using only on feature.

Visualisation of all 561 features by applying t-sne on the data

I tried visualisation using all the 561 features via t-sne. Simple univariate analysis showed that using just 'fBodyAccMagmean' or 'tGravityAccMeanX' feature we were able to do some kind of classification, so imagine what could be done using all the 561 features. Hence, I thought I could generate very good visualization using all the features.
Now, when the number of features are huge, then we can make use of t-sne for visualization purpose. Since we have 561 dimensional vector representation of expert engineered features and we saw earlier that even single features are very useful in determining some class labels. I tried to understand the power of using all the features by applying t-sne on the data. Using t-sne, I mapped the 561 dimensional space to 2 dimensional space. In the below plot, each of the colour coding is one of the class label. First thing, we notice is that most of the points are well clustered together. There is not much overlap except for standing and sitting. Red points and blue points are overlapping a lot. Of course there is a region where blue and red points overlap and there is some region where they don't overlap. Here, blue and red points means standing and sitting respectively. But, we see that rest other features are well grouped together and no overlap. This implies that we should be able to separate all the classes very well. Probably, we may face some challenges when we separate standing and sitting. Because there is some overlap in the t-sne plot.
The t-sne plot was plotted using perplexity=30 and iterations=1000 which are the default values in the scikit library. So to conclude, from the t-sne diagram we can conclude that we should be able to build classifiers using these 561 expert engineered features. But, it will still have some problem in distinguish standing from sitting. Except that everything else it should work fairly well. Even simple linear classifier might work well on these features. So the key takeaway from this t-sne plot is that we could face challenges when we have to separate standing and sitting classes.

Observations:

1. Most of points are well clustered together.
2. There is not much overlap except for standing and sitting class labels.
3. Red points and blue points are overlapping a lot. Ofcourse there is a region where blue and red overlap and there is some region where they dont overlap. Here, blue and red means standing and sitting respectively. But, we see that rest other features are well grouped together and no overlap.
4. So, the tsne plot implies that we should be able to separate all the classes very well. Probably, we may face some challenges when we separate standing and sitting.

Algorithms and Techniques

Classical Machine Learning Approach
Here, I will be applying Random Forest using 561 expert engineered features. I will be working on preprocessed csv files generated in Data Cleaning step. I will use RF with grid search. I will be trying out 2 hyperparameters – 'max_depth' and 'n_estimators'.
Random forest is just an improvement over the top of the decision tree algorithm. The core idea behind Random Forest is to generate multiple small decision trees from random subsets of the data. Each of the decision tree gives a biased classifier. They each capture different trends in the data. This ensemble of trees is like a team of experts each with a little knowledge over the overall subject but thorough in their area of expertise. To classify majority vote is considered.
Deep Learning Model – RNN (LSTM)
Recurrent neural network (RNN) are a type of neural network where the output from previous step are fed as input to the current. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.
Here, I will develop a Long Short-Term Memory network model(LSTM) a type of RNN that are able to learn and remember over long sequences of input data. They are intended for use with data that is comprised of long sequences of data, upto 200 to 400 time steps. That is why I have decided to go with this for the problem.
The model will learn to extract features from sequences of observations and how to map the internal features to different activity types.
The benefit of using LSTMs for sequence classification is that they can learn from the raw time series data directly and in turn do not require domain expertise to manually engineer input features. The model can learn an internal representation of the time series data. Also, I will try to achieve comparable performance to RF model fit on the dataset with engineered features.

Benchmark

For this problem, the benchmark model will be a naïve predictor which will always predict the output class label as Laying. The reason I chose this was because number of data points for this class label were highest across training and testing dataset.

So looking at following numbers –

```
In [21]: print(train.ActivityName.value_counts())
         print(test.ActivityName.value_counts())

         LAYING                1407
         STANDING              1374
         SITTING               1286
         WALKING               1226
         WALKING_UPSTAIRS      1073
         WALKING_DOWNSTAIRS     986
         Name: ActivityName, dtype: int64
         LAYING                537
         STANDING              532
         WALKING               496
         SITTING               491
         WALKING_UPSTAIRS      471
         WALKING_DOWNSTAIRS    420
         Name: ActivityName, dtype: int64
```

The naïve predictor's accuracy will be 537/2947 = 18.22%. Hence, the benchmarking model will be this naïve predictor and my model has to perform better than this model.

III. Methodology

Data Preprocessing

Summarising again, from the previous section of Exploratory Data analysis:
I checked for any duplicates in train and test dataset. Similarly, I also checked for null values. There were no duplicates and no null values.
Next thing to check was if I had imbalance dataset. Because we have 6-class classification task, so one of the very important things that I needed to test for is, is there any imbalance in the data, So, I checked for any imbalance in the data. We have got almost same number of data points from all the subjects. So, I plotted for each subject/person, I plotted how many datapoints I have for each of the class - standing, sitting, etc. There is no imbalance in the data.
Now, next question that needs to be answered was how many data points do I have per activity. Here, also I notice that I have lesser data points across total data for walking downstairs but it's not significantly lower than laying down. There is some class imbalance. It's not severe enough to be worried about.
Special symbols were removed from feature names of the dataset.
All of this work can be found in the ipython notebook named **EDA.ipynb**

Implementation

There are two parts to the implementation:
Classical machine learning approach: Can be found in **PREDICTION_RF.ipynb**
Deep Learning Model – RNN (LSTM): Can be found in **PREDICTION_LSTM.ipynb**

Classical machine learning approach
Here, I used only 561 expert engineered features and applied Random Forest a classical machine learning model. I used preprocessed csv files generated in Data Cleaning step. After loading the train csv file, I got 564 columns. The last 3 columns were basically extra columns - subject id, acitivity index and activity name. I removed these columns when I was constructing our train and test data. These extra columns are dependent variables, y.
After loading, I saw that I had 7352 points in train dataset and 2947 points in test dataset.
Afterwards I wrote some helper functions to help me speed up things like to plot confusion matrix.
I tried RF with grid search. I tried out 2 hyperparameters – 'max_depth' and 'n_estimators'.
It took around 0.5 minutes to train the model. And the accuracy that I obtained was 91%.
Now, let's have a look at the confusion matrix. From confusion matrix, we can see that laying has diagonal value as 1. It means that I was able to identify this activity with 100% precision and 100% recall. This was obvious because we saw that in univariate analysis, there were some simple features with which we were easily able to detect laying activity. Now for other class labels like walking_downstairs whose value is above 0.90, it means the model is pretty good. There is one class label standing, whose value is 0.88. For this class, around 12% of the points were labelled/classified as sitting. This matches with our observation in the t-sne plot. We saw that there is some overlap between sitting and standing. So in the final report we see that sitting has low recall and standing has low precision. Other values are pretty good.
So to conclude, the confusion matrix tells that overall the model is pretty good at classifying the activities. Its classifying 100% for laying type of activity. But the model is confusing between sitting and standing little bit. The overall accuracy is pretty decent. We decided to look at overall accuracy and f1-score and confusion matrix too.
The best hypermeters are – max_depth: 7 and n_estimators: 90
And cross validation score is 91%

Now, with using these 561 features, the model is still confusing between sitting and standing. The model is decent no doubt, but in this case one could ask the expert and ask them to design some more special features which can help resolving the confusion between standing and sitting which can specifically distinguish between standing and sitting.

Deep Learning Model – RNN (LSTM)
Here I tried a deep learning model - a LSTM model. The input is a raw time series data.
I started off with creating few helper functions for loading data, drawing confusion matrix. Initialized random seed as 7. I used tensorflow and keras both. I initialised my hyperparameters, which are, epochs = 30, batch_size=64.
The design of LSTM is very important. I tried out a very simple and elegant architecture.

The network architecture: I tried out 3 variants of LSTM sequentials models.

1. First model: The first layer is a LSTM layer with 32 units. So, my inputs went to 32 parallel LSTMs. Then I added a dropout layer of 0.4. Then I added a dense layer with 100 neurons with relu activation function. Then I added a dense layer with sigmoid activation function. This layer will outcome 6 classes.
2. Second model: The second model has two layers of LSTM instead of only one. The first layer is a LSTM layer with 32 units. Then I added a dropout layer of 0.4. Then one more layer of LSTM with 32 units. Then again a dropout layer of 0.4. Then I added a dense layer with 100 neurons with relu activation function. Then I added a dense layer with sigmoid activation function. This layer will outcome 6 classes.
3. Third model: The third model is similar to second model but has an batch normalization layer in sandwitched in between. The first layer is a LSTM layer with 32 units. Then I added a batch normalization layer. Then I added a dropout layer of 0.4. Then one more layer of LSTM with 32 units. Then I added a batch normalization layer. Then again a dropout layer of 0.4. Then I added a dense layer with 100 neurons with relu activation function. Then I added a dense layer with sigmoid activation function. This layer will outcome 6 classes.

All of the above models were trained with rmsprop optimizer and the metric I cared about was accuracy and the loss that I am trying to minimize is categorical_crossentropy because its a multi class.

Refinement

Improvements in Random Forest Model:
To improve the model and develop a better model as much as possible I played with following numbers:

Max_depth	3	5	7	9	
n_estimators	10	30	50	70	90

I found that max_depth: 7 and n_estimators: 90 gave me the best result without overfitting. This was achieved using gridsearch.
Reference: **PREDICTION_RF.ipynb**

Improvements in Deep Learning Model – RNN (LSTM):
To improve the model and develop a better model as much as possible I tried various LSTM architectures. To increase the performance I added one more layer of LSTM with a dropout layer. The performance dipped little bit. The performance on training data set had improved but on validation dataset, the performance had decreased. Hence, I decided to use batch normalization layer as it can reduce overfitting because of its regularisation effects.
The model's performance increased little bit and this model was the best among the three models that I tried.
Reference: **PREDICTION_LSTM.ipynb**

IV. Results

Model Evaluation and Validation

Classical machine learning approach:
The final Random forest model performs very well in my problem. I used validation dataset to make sure that there is no overfitting. The model is pretty robust as can be seen by its performance on the unseen data – my validation data set. The overall accuracy was pretty good around 91%. This shows that the model hasn't overfit my training data set and can generalize well to unseen data. The model is robust enough and the model's result can be trusted very much enough.

Deep Learning Model – RNN (LSTM):
The third LSTM architecture is the best model among the three LSTM models that I tried. Its performance on the validation data set was touching 92%. It's pretty robust given its performance on the unseen data. Hence, this proves that the model generalizes well to unseen data and its result can be trusted very much.

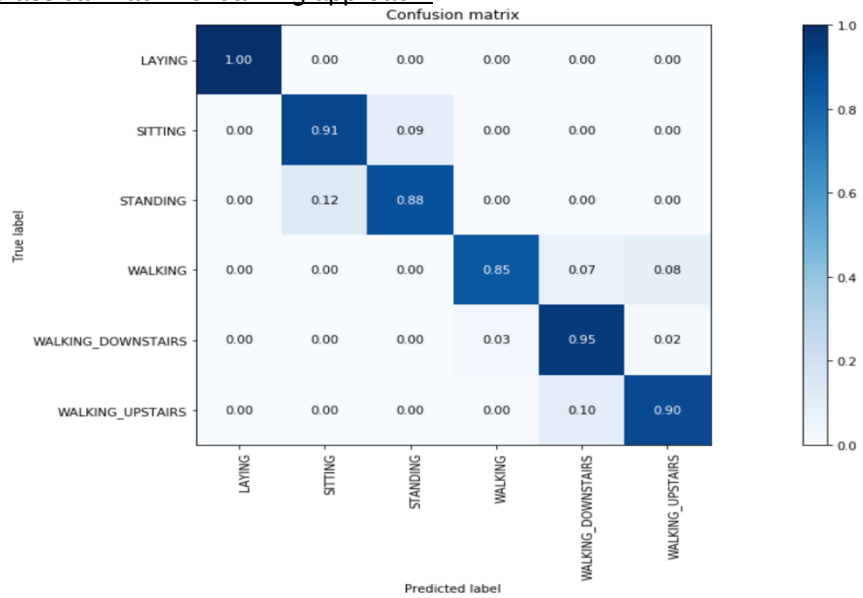
Justification

Yes, the model's final solution is significantly better than the benchmark model that I had decided. The benchmark model was a naïve predictor which will always predict the output class label as Walking_Downstairs. The naïve predictor's accuracy will be 420/2947 = 14.25% as shown earlier.

V. Conclusion

Free-Form Visualization

Classical machine learning approach:



This confusion matrix of the best random forest model shows the beauty and the importance of exploratory data analysis. Via univariate analysis and the t-sne plot using all the 561 features we were had a hint that it would be tough to distinguish between 'sitting' and 'standing' type of activities and the 'laying' activity can be easily distinguished.

Deep Learning Model – RNN (LSTM):

LSTM Architecture - 3

```
In [29]: # Initiliazing the sequential model
         model_3 = Sequential()
         # Configuring the parameters
         model_3.add(LSTM(32, return_sequences = True, input_shape=(timesteps, input_dim)))
         model_3.add(BatchNormalization())
         model_3.add(Dropout(0.4))
         model_3.add(LSTM(32))
         model_3.add(BatchNormalization())
         model_3.add(Dropout(0.4))
         model_3.add(Dense(100, activation='relu'))
         # Adding a dense output layer with sigmoid activation
         model_3.add(Dense(n_classes, activation='sigmoid'))
         model_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 32)           128
_________________________________________________________________
dropout_6 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
lstm_7 (LSTM)                (None, 32)                8320
_________________________________________________________________
batch_normalization_2 (Batch (None, 32)                128
_________________________________________________________________
dropout_7 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_6 (Dense)              (None, 100)               3300
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 606
=================================================================
Total params: 17,858
Trainable params: 17,730
Non-trainable params: 128
_________________________________________________________________
```

The above image helps us to realise the importance of dropout layers and batch normalization layer. Architecture-3 is very much similar to architecture-1. Also, it has an extra layer of LSTM as compared to architecture-1. Given the training data size and the number of params of the LSTM network=17858, the model still performs quite well. This is merely possible because of the dropout layer and batch normalization layer.

Reflection

At first the problem seemed little bit easy when I tried to apply random forest. The dataset had very good expert engineered features. EDA showed me that a simple if else conditions would be able to classify the given data little bit better than my benchmark model. To increase the performance of the model, I increased the number of estimators to around 90. Initially I thought, it might overfit but that wasn't the case(Can be checked via the models performance on CV data against various hyperparamters).

In my second approach, I wanted to try out a deep learning model, and see how it performs against expert engineered features. This deep learning model was a type of RNN called as LSTM. This model was given just a series of raw signals and it was able to identify the relationships with the labelled output. The first model architecture itself gave accuracy little bit better than the RF model. I also wanted to try out the extra learning that I did during the course – dropouts and batch normalization layer etc. This definitely improved the model's performance and the accuracy went on to achieve the 92% mark. The difficulty part of this second approach was how to design the architecture of the model. Which layer to put first, how many hidden units at LSTM layers, how much dropout to use. These were some of the questions that I had to crack to get a good Deep neural network model that performs as better as the classical machine learning model which uses expert engineered features.

Improvement

The handcrafted expert engineered features gave me around 91% accuracy. Without any domain knowledge, just training a very simple LSTM model which had only two layers of LSTM, I was still able to get around 92% accuracy. If I had much more datapoints say 100 times more data, I could have designed a much more complex LSTM network and could have achieved much better accuracy. This shows the pros as well as cons of using deep learning models. Deep learning model requires lots of data to train but doesn't necessarily require expert engineered features, whereas if the same amount

of data set is given to classical machine learning models, then the model can perform very well if the features input good values into the system which is possible only when they are designed by experts. There were few algorithms like Convolutional Neural network LSTM that I researched but since I didn't know much about it, hence I didn't try. I will definitely give it a try in near future.
And definitely I would be able to achieve a much better solution even if I set my final solution as the new benchmark.