Human Activity Recognition using Smartphone Data

Ayush Anand

Third year Undergraduate, Computer Science and Engineering, Indian Institute of Technology, Gandhinagar.
ayush_anand@iitgn.ac.in

In this paper, we look at how we can use smartphone sensor data for human activity recognition using different methods. These methods have been separately implemented in certain related papers; in this paper we try to compare their performance and conclude which one performs better in general. The different methods that we study are Feature engineering, k-nearest neighbors with Dynamic Time Warping as the distance metric and an LSTM model.

Additional Keywords and Phrases: kNN, Ubiquitous Computing, Dynamic Time Warping, LSTM, Smartphones

1 INTRODUCTION

Human activity recognition (HAR) is the problem of classifying sequences of accelerometer data recorded by specialized sensors or smart phones into known well-defined movements. Human Activity Recognition has appealing use in healthcare applications, such as monitoring activities of the elderly or dementia patients through the exploitation of Ambient Intelligence (AmI).

There have been various methods employed in the past to solve the problem of human activity recognition. Classical approaches to the problem include feature engineering using sliding window technique to extract useful features from the timeseries data and then apply machine learning algorithms like decision trees or SVM for classification as described by Davide et al in [1]. Another approach to the problem is using k-nearest-neighbor with Dynamic Time Warping as the distance metric, as described by Mark et al in [2]. Recently, deep learning methods such as recurrent neural networks or Long-Short-Term-Memory models have gained attention, due to their better performance and no need for feature engineering. In this paper, we shall employ the above 3 methods for Activity recognition and compare their performance.

In this paper, we use the UCI-HAR-Dataset, which is further described in the coming sections. Since it consists of smartphone accelerometer data, this paper also serves as a study of ubiquitous solutions to the activity recognition problem.

In the next section, we cite certain noteworthy related works in this field. Then we move on to describing the dataset, and then a brief overview of each method employed. We then look at the results from these different methods and compare them. Finally, we conclude with the future scope and limitation of our work.

2. RELATED WORK

Using smartphone data for activity recognition has gained popularity due to several advantages such as easy device portability without the need for additional fixed equipment, and comfort to the user due to the unobtrusive sensing. However, the earlier works in the field used specific purpose hardware devices like in [3] or sensor body networks as in [4]. Although, the use of numerous sensors could improve the performance of the classification algorithms, they are not convenient and most scenarios would lack the presence of a specific sensor. As such, smartphones have been the recent area of interest in the HAR field. The use of smartphones for HAR also has certain disadvantages like resource sharing with other apps, battery overheads, privacy concerns and susceptibility to external environments, but they are easily overlooked due to the ubiquity they provide

There are various ML methods that have been previously employed for human activity recognition These include Naive Bayes, Support Vector Machines, Threshold-based and Markov chains [5]. SVMs for classification was studied by Davide et al in [1], Ravi et al in [6] and Weiss et al in [7]. SVMs have found successful application in several important areas including heterogeneous types of recognition such as handwritten characters as described by LeCun et al in [8] and speech [9].

3. EXPERIMENT

In this section, we briefly describe the experiment that was performed by the authors of the UCI-HAR-Dataset at the UCI. 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, 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz were captured. The experiments were video-recorded and then manually labeled. The dataset was randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data. Figure 1 shows a participant performing the experiment.

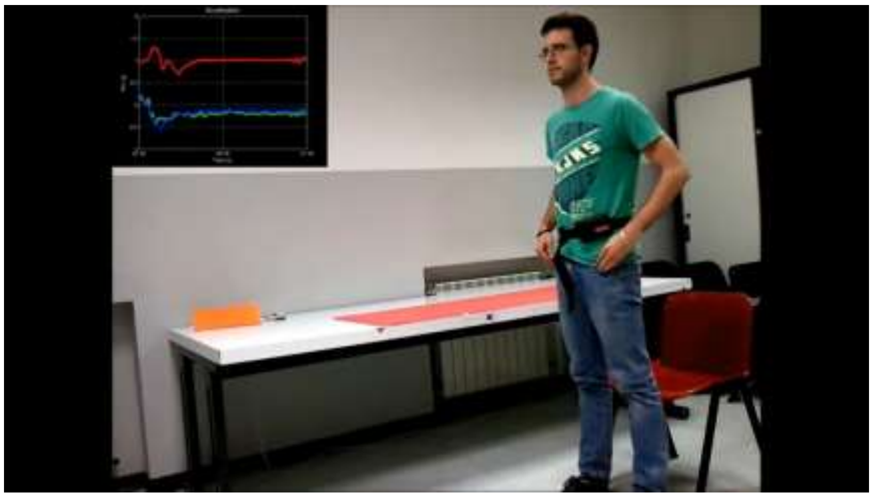


Figure 1: Participant with the phone tied to his waist performing 1 of the 6 activities (standing)

4. DATA ANALYSIS

In this section we try to explain the dataset provided and provide certain basic visualizations which will clarify things further.

The dataset captures 3-axial linear acceleration and 3-axial angular velocity at a frequency of 50 Hz. 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 was separated into body acceleration and gravity using a Butterworth low pass filter. 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 561 features was obtained by calculating variables from the time and frequency domain.

The dataset contains the following information for each data record

- Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
- Triaxial Angular velocity from the gyroscope.
- A 561-feature vector with time and frequency domain variables.
- Its activity label.
- An identifier of the subject who carried out the experiment.

Figure 2 visualizes sample observations from the HAR dataset for different classes (taken from Mark et al[2])

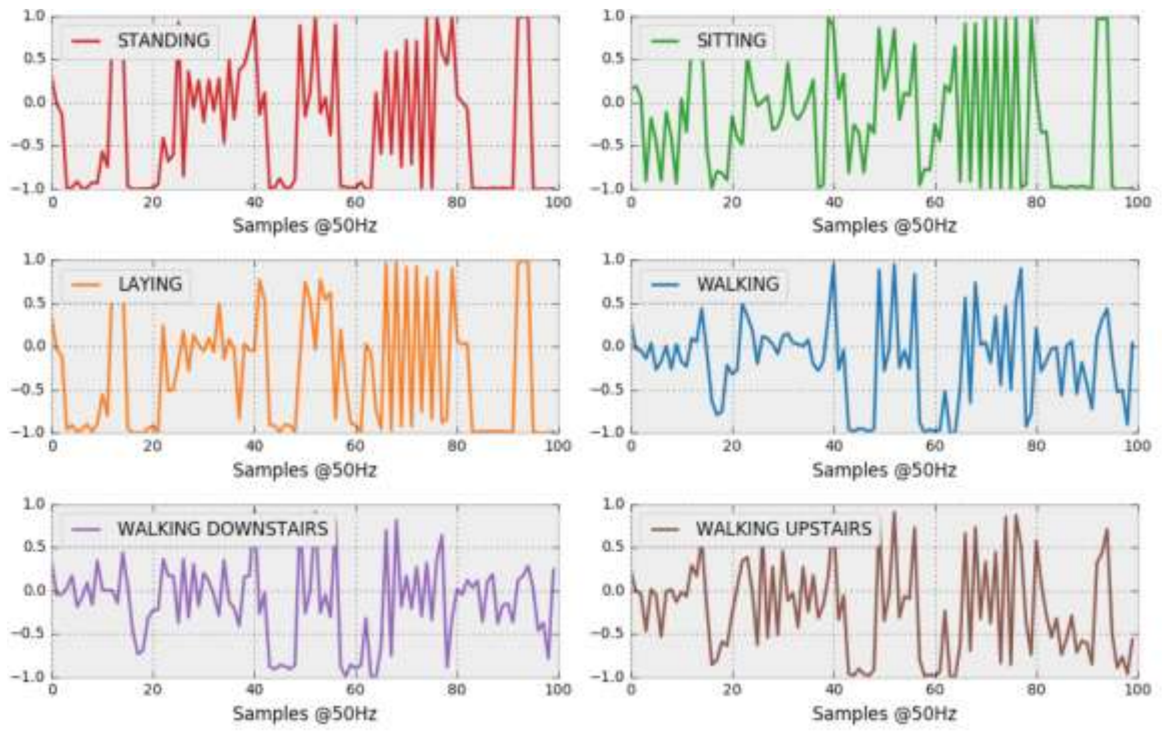


Figure 2: Dataset Visualization

We can see the above signal values are irregular, and thus will not give very accurate readings. To simplify the signal further, we perform a FFT on the signal to note which frequencies contribute the most to the above plot, and obtain the following results:

5. Algorithms

5.1 K Nearest Neighbors

K-Nearest Neighbor is a supervised machine learning algorithm used for many classification problems. It takes an unlabeled data point and compares it to a population of labeled observations. By finding the class labels for k nearest neighbors to the datapoint, we classify the datapoint into a class depending on majority inference. Figure 3 shows the idea behind the k nearest neighbor algorithm:

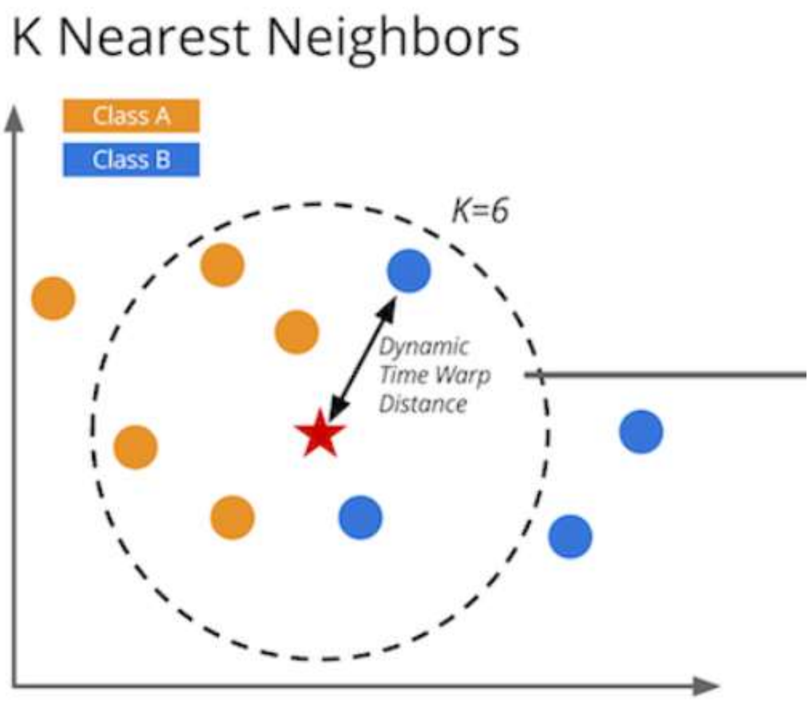


Figure 3: Broad level overview of K Nearest Neighbors

Now, there is an inherent question that pops up when we look at the definition of kNN: How do we define the notion of neighbors? The first guess would be the Euclidean distance between the points, but it is not so straightforward in all cases. Consider the problem we have at hand, for example. How do we compute the “distance” between two time series? We use the method of Dynamic Time Warping to overcome this problem.

Dynamic Time Warping (DTW) finds the optimum alignment between two sequences of observations by warping the time dimension with certain constraints. Because of this temporal dimension warping, DTW is good for classifying sequences that have different frequencies or that are out of phase

Figure 4 and 5 below provide a visual representation of how DTW works. Let us look at the algorithm behind DTW in a little more detail.

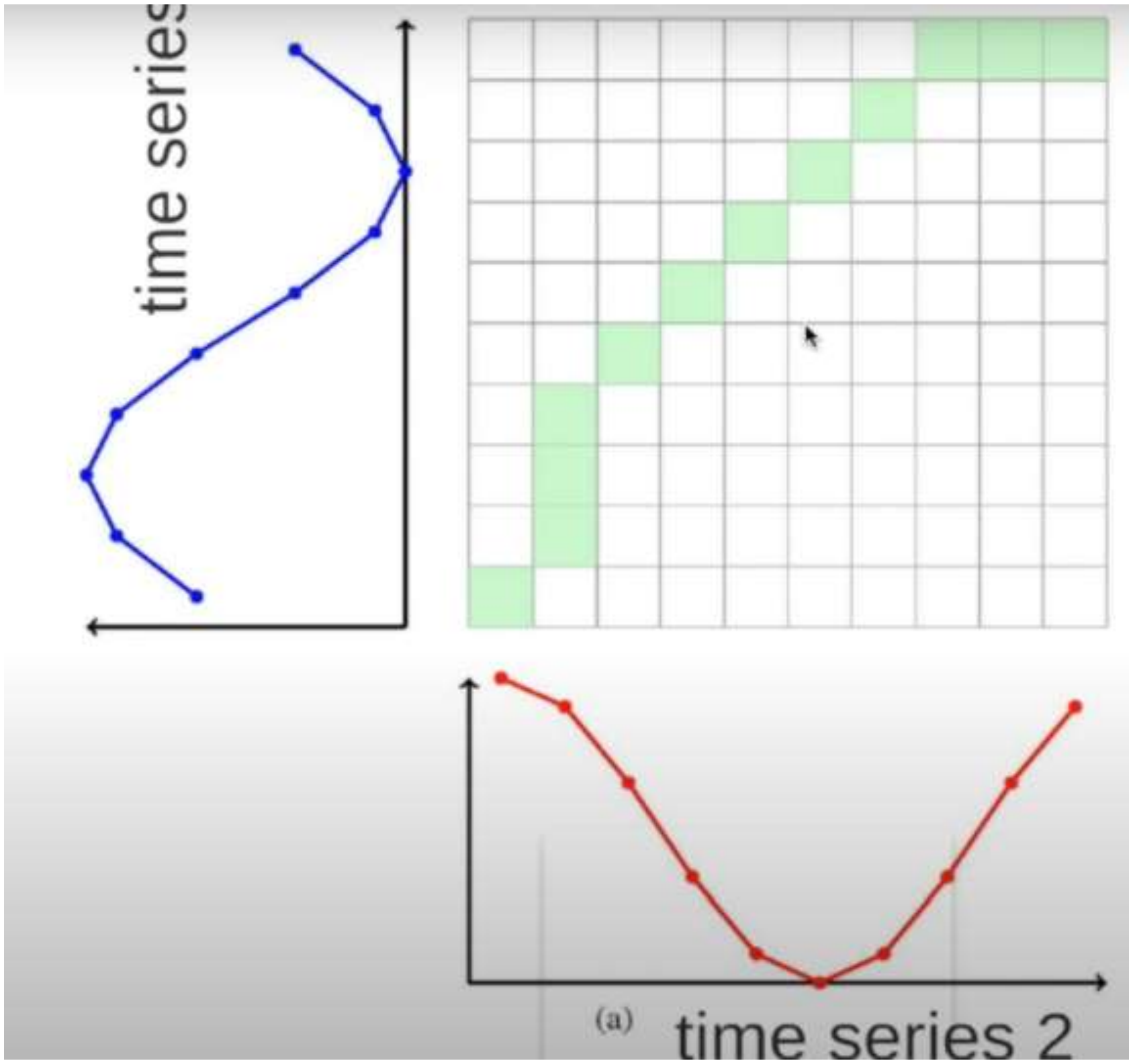
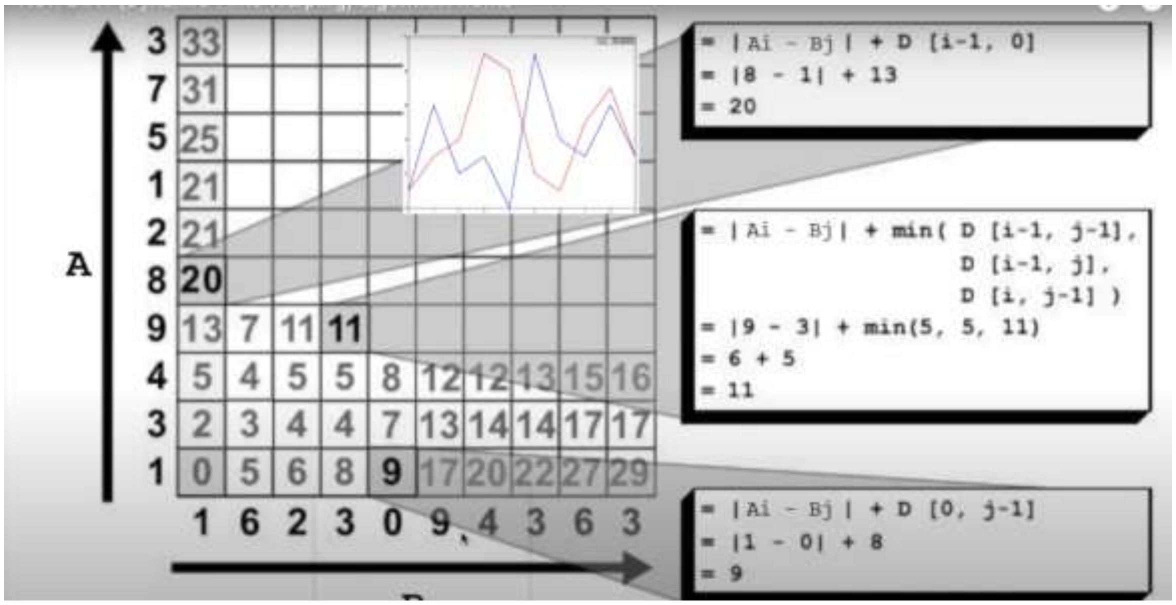


Figure 4: The idea behind DTW



A										
3	33									
7	31									
5	25									
1	21									
2	21									
8	20									
9	13	7	11	11						
4	5	4	5	5	8	12	12	13	15	16
3	2	3	4	4	7	13	14	14	17	17
1	0	5	6	8	9	17	20	22	27	29
B	1	6	2	3	0	9	4	3	6	3

```
= |Ai - Bj| + D [i-1, 0]
= |8 - 1| + 13
= 20
```

```
= |Ai - Bj| + min( D [i-1, j-1],
                   D [i-1, j],
                   D [i, j-1] )
= |9 - 3| + min(5, 5, 11)
= 6 + 5
= 11
```

```
= |Ai - Bj| + D [0, j-1]
= |1 - 0| + 8
= 9
```

Figure 5: Distance matrix calculation in DTW

Let us say we have two timeseries sequences A and B with n data points each. DTW tries to find the best alignment between the sequences by computing an n*n distance matrix as demonstrated in figure 2 and 3 above. Each entry in the matrix is filled by a dynamic programming algorithm which is described below:

```
Cost= abs(A[i] - B[j])
DTW [i, j]: = cost + minimum (DTW [i-1, j], DTW [i, j-1], DTW [i-1, j-1])
```

Computing the full distance matrix between A and B scales with O(n^2), which is clearly computationally expensive. This performance can be improved by constraining the amount of warping allowed. This limits the number of cells that need to be computed in the DTW distance matrix. Research by Keogh et al has shown that a warping window not only improves performance but also improves classification accuracy [10]. For our experiment, we have taken the warping window to be 10 and the number of neighbors to be 1.

5.2 Feature Engineering

Feature engineering is the process of using domain knowledge to extract features (characteristics, properties, attributes) from raw data. For time series data specially, it becomes important to collect important information stored in the data and then apply the classification algorithm we want like SVM, decision trees etc.

Choosing which features are important is a difficult task, and requires experience and expertise. For the UCI-HAR-Dataset, the authors have already provided the data after applying feature engineering, and have extracted a total of 561 features. The complete list of features can be found here.

Applying classification algorithms on this feature engineered data gives excellent results, as the features extracted are from both the time domain and frequency domain. This also almost beats the other methods we are looking at, as we will see in the next section.

We have tried feature engineering on the data on our own as well. We combined the Inertial accelerometer signals from the three axes and computed the total acceleration data. We then tried different features for extraction like autocorrelation and mean absolute deviation, and the results are poor as compared to the previous one, as expected.

5.3 Long Short-Term Memory (LSTM)

LSTM network models are a special type of recurrent neural network 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, with up to 200-to-400-time steps. They prove to be a good fit for time series classification, and perform excellently well on the raw data itself.

We will not go much into the details of how an LSTM works, as it is beyond the scope of this paper. Our implementation of the LSTM model follows the description provided in [11]. We provide a brief description of how LSTM works before moving on to the next section, following along the lines of [12]

The architecture which allows LSTM to remember is called the LSTM cell and is shown in figure 6. This LSTM cell forms the basic unit of the LSTM model. Next, we briefly look at the steps involved in the basic functioning of the LSTM model.

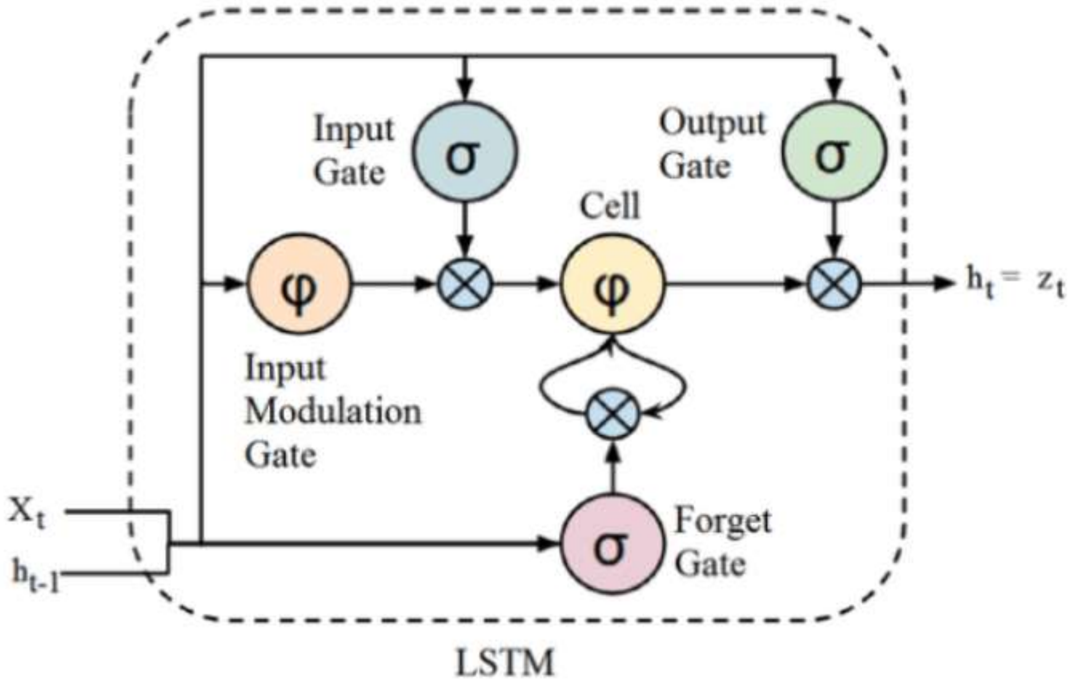


Figure 6: LSTM Cell

The first step in the LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values will be updated. Next, a *tanh* layer creates a vector of new candidate values, that could be added to the state. In the next step, these two are combined create an update to the state.

Next the old cell state is updated into, Ct, from the previous cell state Ct-1. This involves certain calculations which we will not go into

Finally, we need to decide what we're going to output. This output will be based on the cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh and multiply it by the output of the sigmoid gate, so that we only output the parts we want.

5 EVALUATIONS

In this section, we present the evaluations and results of our study.

5.1 kNN on Raw Dataset

To apply k-Nearest Neighbor for classification, we take both the feature engineered dataset as done by Mark et al in [2] and the raw dataset. For the raw dataset, we generate a total acceleration vector, which is simply the square root of the sum of squares of accelerations along the x, y and z axes.

```
X_train=np.sqrt(ax_train**2 + ay_train**2 + az_train**2)
```

The X_train matrix is a 7352 * 128 matrix, and running kNN on this whole dataset is not feasible, as it involves millions of DTW distance calculation which is both computationally and memory-wise expensive. Thus, we adopt the method shown by Mark et al in [2] and take every 10th datapoint from the dataset. This still ends up in the algorithm having to compute 217120 DTW distances, which typically takes around half an hour.

The results for the K Nearest Neighbors algorithm ran on the raw dataset are shown in table 1 and the confusion matrix is shown in figure 7

```
                      Precision     Recall    F1-score   support

           WALKING       0.94       0.87       0.90         54
  WALKING UPSTAIRS       0.83       0.63       0.72         63
WALKING DOWNSTAIRS       0.52       0.92       0.67         25
           SITTING       0.65       0.49       0.56         67
          STANDING       0.70       0.73       0.71         48
            LAYING       0.62       0.84       0.71         38

          accuracy                             0.71        295
         macro avg       0.71       0.75       0.71        295
      weighted avg       0.73       0.71       0.71        295
```

Table 1: Classification Report for kNN on Raw Dataset

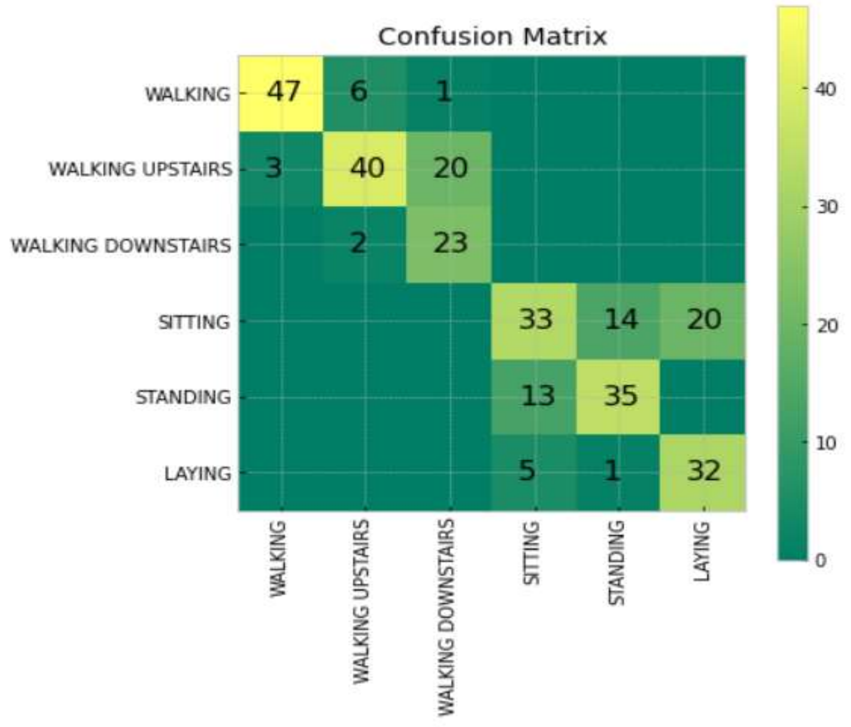


Figure 7: Confusion Matrix for kNN on Raw dataset

This achieves an accuracy of 71 percent, which is a reasonable estimate on dataset with no feature engineering. We take this to be as baseline and compare other approaches to this in the next part.

5.2 kNN on Feature Engineered Data

The results for the K Nearest Neighbors algorithm ran on the raw dataset are shown in table 2 and the confusion matrix is shown in figure 8.

```
                    Precision    Recall    F1-score   Support

           WALKING       0.96      0.80      0.87        60
  WALKING UPSTAIRS       0.85      0.80      0.83        51
WALKING DOWNSTAIRS       0.68      0.97      0.80        31
           SITTING       0.78      0.78      0.78        51
          STANDING       0.84      0.76      0.80        55
            LAYING       0.90      1.00      0.95        47

          accuracy                           0.84       295
         macro avg       0.84      0.85      0.84       295
      weighted avg       0.85      0.84      0.84       295
```

Table 2: Classification Report for kNN on Feature engineered Dataset

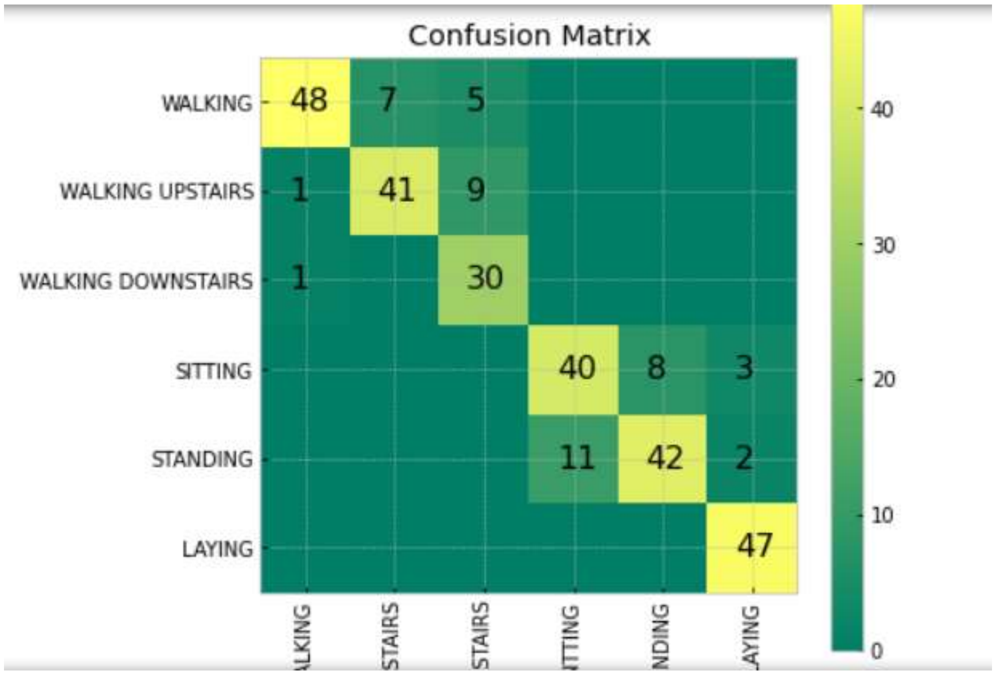


Figure 8: Confusion Matrix for kNN on Feature engineered data

This achieves an accuracy of 84 percent, which is a better as compared to the baseline, as expected.

5.3 Feature Engineering

For the feature engineering experiment, we run a logistic regression algorithm on the dataset with features already extracted as provided in the UCI-HAR-Dataset. Table 3 shows the classification report of this part and figure 9 shows the corresponding confusion matrix.

```
              Precision  Recall   F1-score   support

           1       0.94      0.99      0.97       496
           2       0.96      0.95      0.95       471
           3       0.99      0.94      0.97       420
           4       0.97      0.87      0.92       491
           5       0.89      0.97      0.93       532
           6       1.00      0.99      1.00       537

    accuracy                           0.95      2947
   macro avg       0.96      0.95      0.95      2947
weighted avg       0.96      0.95      0.95      2947
```

Table 3: Classification Report for Feature Engineering + Logistic Regression

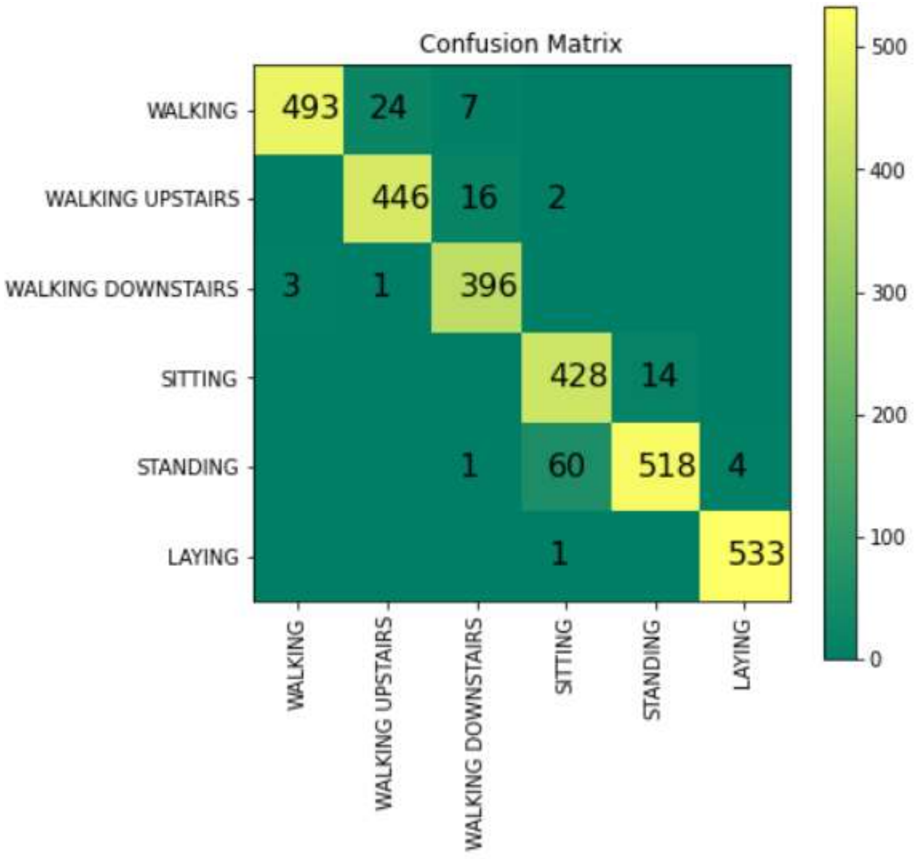


Figure 9: Confusion Matrix for Feature Engineering + Logistic Regression

This achieves an accuracy of almost 95 percent, which is the best of the three methods that we see. This is also expected as the data we have used has been heavily feature engineered, combining features from both the time and frequency domain. We combine this with logistic regression and hence we get pretty high accuracy.

5.4 Custom Feature Engineering

In this section, we manually extract a feature from the raw dataset, that is autocorrelation of the signal in particular. We tried extracting other features as well, however here we provide analysis for implementation with autocorrelation only. Table 4 shows the classification report and the figure 10 shows the corresponding confusion matrix.

```
                    Precision   Recall  F1-score   support

     WALKING           0.44       0.84      0.58       496
WALKING UPSTAIRS       0.34       0.05      0.09       471
WALKING DOWNSTAIRS     0.64       0.49      0.55       420
     SITTING           0.61       0.56      0.58       491
     STANDING          0.71       0.85      0.78       532
     LAYING            0.84       0.82      0.83       537

    accuracy                                0.61      2947
   macro avg           0.60       0.60      0.57      2947
weighted avg           0.60       0.61      0.58      2947
```

Table 4: Classification report for Feature Engineering with Autocorrelation as the feature

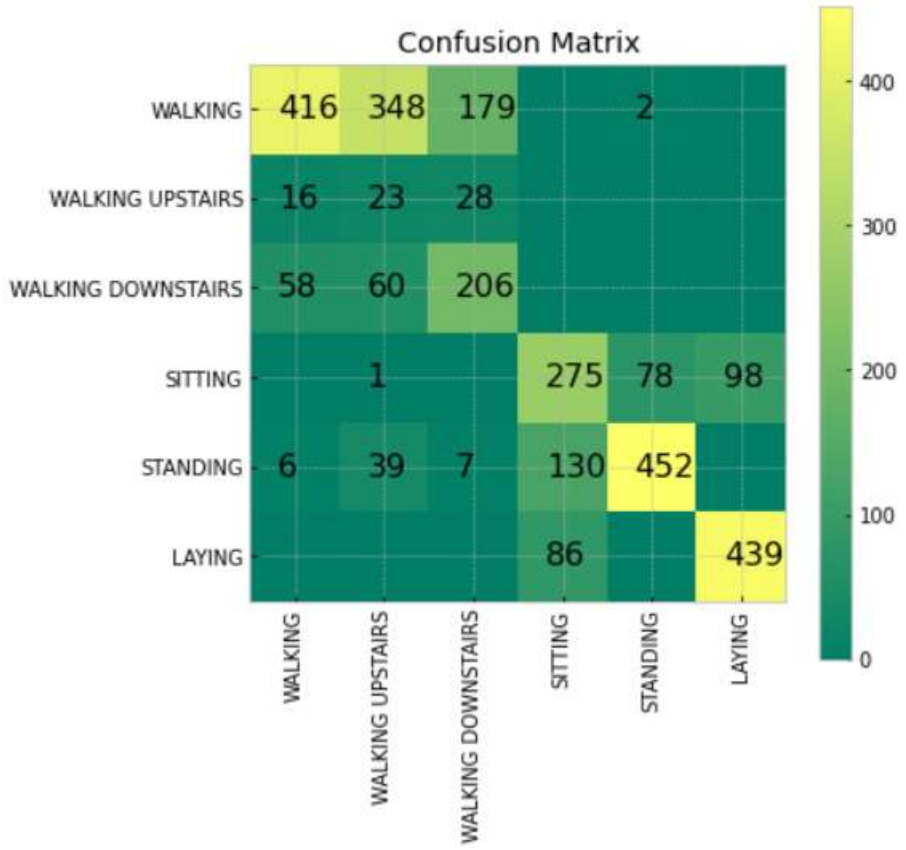


Figure 10: Confusion Matrix for Feature Engineering with Autocorrelation as the feature

This method gives us an accuracy of around 61 percent, and this varies with the choice of the feature and the algorithm used.

5.5 LSTM

In this section we train an LSTM model on the raw data. There are three main signal types in the raw data: total acceleration, body acceleration, and body gyroscope. Each has 3 axes of data. This means that there are a total of nine variables for each time step. We combine these to form a 3d array in the format [samples, time steps, features]. We then train the LSTM model on this data. Table 5 shows the classification report for this approach and figure 11 shows the corresponding confusion matrix.

```
                    precision    recall    f1-score   support

           WALKING       0.95      0.90       0.93       521
  WALKING UPSTAIRS       0.93      0.86       0.89       509
WALKING DOWNSTAIRS       0.83      0.95       0.89       366
           SITTING       0.75      0.85       0.80       433
          STANDING       0.88      0.80       0.84       584
            LAYING       0.99      1.00       0.99       534

          accuracy                            0.89      2947
         macro avg       0.89      0.89       0.89      2947
      weighted avg       0.90      0.89       0.89      2947
```

Table 5: Classification Report for HAR with LSTM

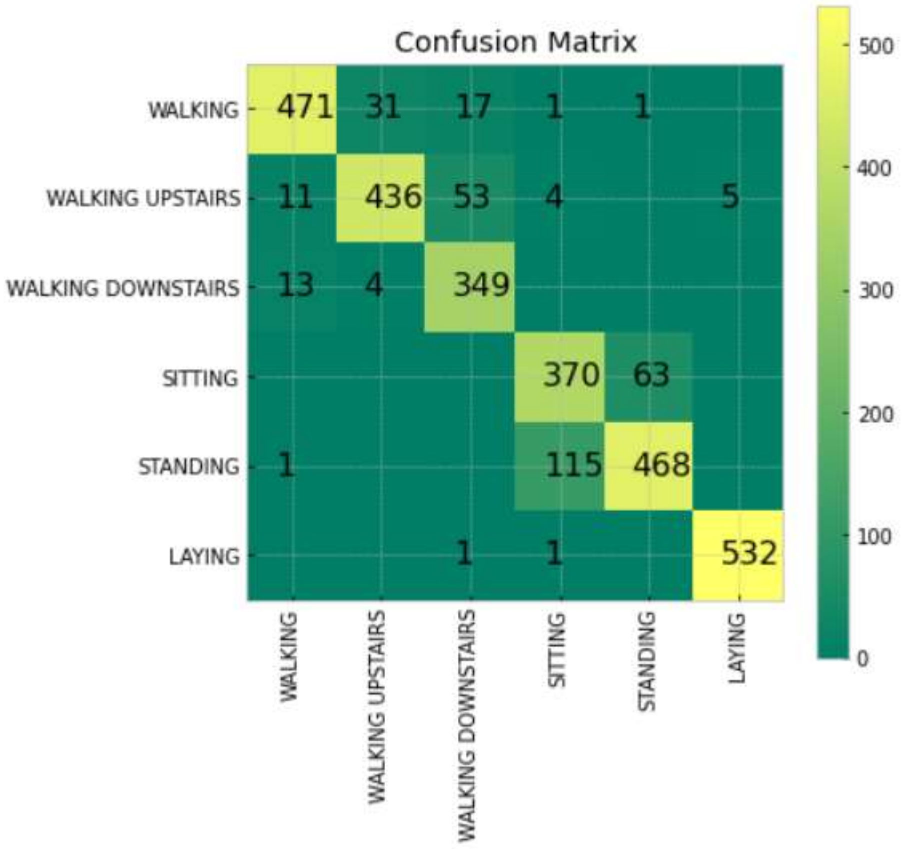


Figure 11: Confusion Matrix for HAR with LSTM

5.6 Comparative Analysis

As a final comparison, we compare all the methods described above with kNN on raw dataset as the baseline. Table 6 summarizes the results of the comparison.

Algorithm	Accuracy wrt ground truth	RMSE wrt ground truth
kNN on raw dataset(baseline)	0.7118	0.7433
Custom Feature Engineering	0.6145	0.9842
Feature Engineering as done in the dataset provided	0.9548	0.2436
LSTM	0.8978	0.4915

Table 6: Comparative Analysis

In general, we conclude that LSTM is the most powerful of the three, as it provides an accuracy of almost 90 percent with no feature engineering whatsoever required. K Nearest neighbor performs well in general, but is computationally very heavy and scales poorly as the size of the data increases. Feature engineering can prove to be handy if the important features are extracted carefully, and in general it requires a domain knowledge of feature extraction and some expertise. If done extensively, as in the UCI-HAR-Dataset originally, combined with powerful machine learning algorithms like SVR it can give very good results.

6. CONCLUSION:

In this paper we compared three different methods for human activity recognition, namely feature engineering, k nearest neighbors and LSTMs. We looked at the variations of the above methods and how their performance varies depending upon how they are applied (raw dataset or some processed part of it). We used the UCI-HAR-Dataset made public by the UCI in 2012 for all experiments. We evaluated each model independently by comparing it with the ground truth, and found LSTM to be the most efficient. We also looked at downsides of methods like kNN, and benefits of extensive feature engineering. This study could further be extended in the future by analyzing more algorithms for HAR as well as making modifications to the ones we saw in this paper.

REFERENCES

[1] Anguita, Davide, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine." In International workshop on ambient assisted living, pp. 216-223. Springer, Berlin, Heidelberg, 2012.Shu-Di Bao, Yuan-Ting Zhang, and Lian-Feng Shen. Physiological signal-based entity authentication for body area sensor networks and mobile healthcare systems. In 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, pages 2455–2458. IEEE, 2006.

[2] https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping/blob/master/data/UCI-HAR-Dataset/features.txt

[3] Cook, D.J., Das, S.K.: Pervasive computing at scale: Transforming the state of the art. Pervasive and Mobile Computing 8(1) (February 2012) 22–35

[4] Allen, F.R., Ambikairajah, E., Lovell, N.H., Celler, B.G.: Classification of a known sequence of motions and postures from accelerometry data using adapted gaussian mixture models. Physiological Measurement 27(10) (2006) 935

[5] Mannini, A., Sabatini, A.M.: Machine learning methods for classifying human physical activity from on-body accelerometers. Sensors 10(2) (2010) 1154–1175

[6] Ravi, N., D, N., Mysore, P., Littman, M.L.: Activity recognition from accelerometer data. In: In Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence(IAAI, AAAI Press (2005) 1541–1546

[7] Kwapisz, J.R., Weiss, G.M., Moore, S.A.: Activity recognition using cell phone accelerometers. SIGKDD Explor. Newsl. 12(2) (March 2011) 74–82

[8] LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker,H., Guyon, I., Mller, U., Sckinger, E., Simard, P., Vapnik, V.: Comparison of learning algorithms for handwritten digit recognition. In: International Conference on Artificial Neural Networks.

(1995) 53–60

[9] Ganapathiraju, A., Hamaker, J., Picone, J.: Applications of support vector machines to speech recognition. Signal Processing, IEEE Transactions on 52(8) (aug. 2004) 2348 – 2355

[10] Xi, Xiaopeng, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. "Fast time series classification using numerosity reduction." In Proceedings of the 23rd international conference on Machine learning, pp. 1033-1040. 2006.

[11] https://github.com/gornes/Human_Activity_Recognition

[12] https://colah.github.io/posts/2015-08-Understanding-LSTMs/