Report for SO-Lively
Human Activity Recognition Detection

James Carlyle

December 2025

University of Southampton

COMP6246 Assessment

SO-Lively is a health-tech startup focused on developing AI-driven solutions for personal wellness, fitness monitoring, and elder-care support. This report looks at alternative machine-learning methods for activity recognition based on data from body-worn sensors.

The goal was to develop a machine-learning based Human Activity Recognition (HAR) system to accurately classify physical activities (e.g., walking, sitting, standing) from sensor data from wearable devices proprietary to the company. HAR is a classical problem in Machine Learning, but no solution has been tested with the SO-Lively wearable devices and the company wants to know what solutions can be used.

1 Introduction

Human Activity Recognition (HAR) data was made available for assessment and labelling by activity type (walking, sitting etc.) with a number of machine learning methods, including unsupervised and supervised learning. This report presents the data loading, analysis, cleansing and windowing exercises undertaken, and then evaluates activity labelling using K-Means, Random Forest (RF) and Convolutional Neural Network (CNN) approaches.

2 Data pre-processing and exploration

2.1 Data loading

Functions were defined for data-loading, and the data was loaded into a training and test dataset.

```
Training dataframe shape: (5568946, 9) (5.5 million rows, 9 columns)
Testing dataframe shape: (1122375, 9)
Columns: Index(['timestamp', 'back_x', 'back_y', 'back_z',
    'thigh_x', 'thigh_y', 'thigh_z', 'label', 'sensor']
```

Several different approaches were taken to structure the data frames, including a multi-index using [sensor, timestamp] and a dictionary using [sensor: data frame], in both cases to ensure that window processing took place within the context of a single sensor (i.e. a 2-second activity window cannot span more than one sensor). In the end, a single external integer index was adopted, with sensor demarcation taking place within the window functions. After the cycling labelled records were removed, and separate stairs classes 4 and 5 (up and down) were merged to a new class 9, the dataset sizes were:

```
Training dataframe shape after cycling and merging stairs: (5059040, 9)
Testing dataframe shape after cycling and merging stairs: (1105427, 9)
```

For the training dataset, the following distribution of labels was seen before and after the initial cleaning:

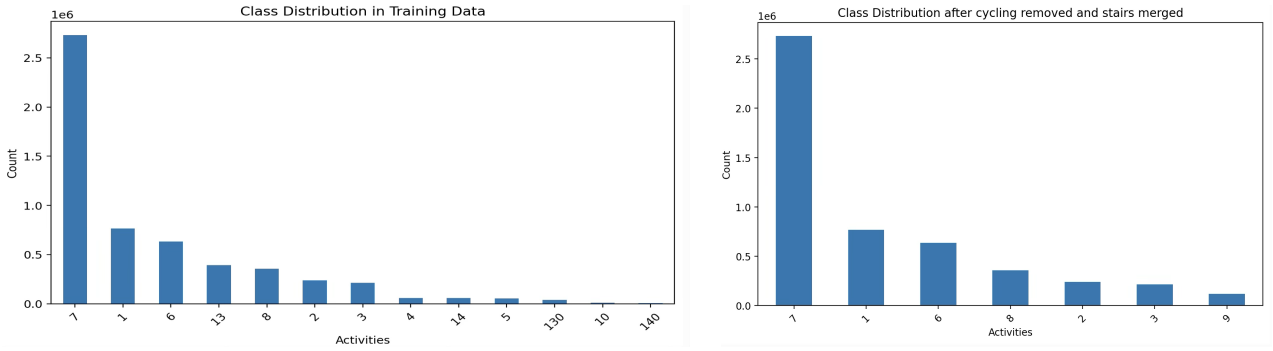

Figure 1: Class distribution in original training data (left) and after cleaning of cycling and stair labels (right).

2.2 Sampling and visualisation for Walking

Sensor 12 was sampled using a filter of [label==1], steps of 1 and 2, and the first 400 rows. This was plotted for back and thigh sensors, but truncated in the graphs below to thigh-only for brevity. Additionally, a Savitzky-Golay filter was applied to de-noise and more easily show patterns in the data, by smoothing with a short window of 5 rows (0.1s).

One interesting feature seen at the end of the sample is a period of 1s when no data was collected by the sensor, i.e. the plot shows a flat section. This has an impact in later windowing: the specification called for a fixed window length of 2s, but the neural network processing requires windows with a fixed number of rows: When the data have gaps, a fixed-row window must be expanded to more than 2 elapsed seconds.

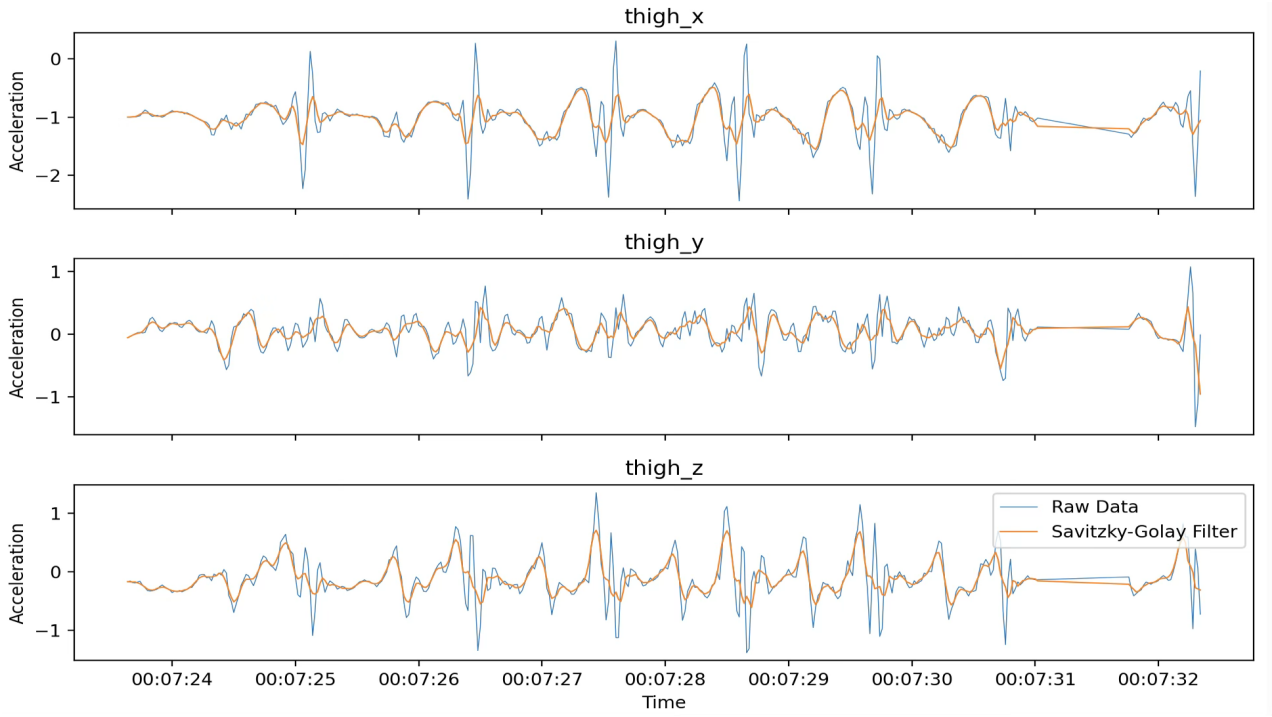

Figure 2: Visualisation of walking data.

3 Data Cleaning and Preparation

3.1 Data quality

There is an issue with all sensors: Some accelerometer data is constant across timestamps. This is particularly true for S007, because all axes of both sensors show a constant value. This is identified by looking for continuities of exact values over contiguous timestamps, grouping by sensor and then axis. Once groups of values were identified, each group was aggregated to capture the first and last timestamps of the series, and the number of rows in between them. There are around 250 instances where a sensor shows an exactly constant acceleration for more than 50 rows (1 second at a sampling frequency of 50Hz). It would be impossible for a human to achieve this, and must be caused by the sensor getting 'lodged' in a particular position.

sensor	axis	_start_time	_end_time	_count
S007	back_z	2019-01-17 00:00:41.840	2019-01-17 00:01:25.660	4383
S007	thigh_x	2019-01-17 00:00:41.840	2019-01-17 00:01:25.660	4383
S007	thigh_z	2019-01-17 00:00:41.840	2019-01-17 00:01:25.660	4383
S007	thigh_y	2019-01-17 00:00:41.840	2019-01-17 00:01:25.660	4383
S007	back_x	2019-01-17 00:00:41.840	2019-01-17 00:01:25.660	4383
S007	back_y	2019-01-17 00:00:41.840	2019-01-17 00:01:25.660	4383
S007	back_x	2019-01-17 01:00:28.580	2019-01-17 01:00:53.580	2501
S027	back_x	2019-01-12 00:01:18.260	2019-01-12 00:01:33.480	762
S027	back_x	2019-01-12 00:01:37.920	2019-01-12 00:01:51.000	655
S023	thigh_x	2019-01-12 00:07:25.360	2019-01-12 00:07:38.260	646

This table shows the top 10 instances of this problem. For example, sensor S007 has 4383 rows where all axes show a constant acceleration between 00:00:41 and 00:01:25, and sensor S023 has 646 rows where the thigh x acceleration is constant, starting at 00:07:25. Across all sensors, there are 70,897 rows with static readings out of 505,000 in the training set.

3.2 Spurious outlier data

S007 accelerometer data show unreasonable outliers across all axes and for both back and thigh sensors, seen in these violin plots; the AX3 sensor is rated to +-8g, so should never report -60g. However, the means

and distribution of the data from sensors without outliers is similar, showing that only a few timestamps are erroneous.

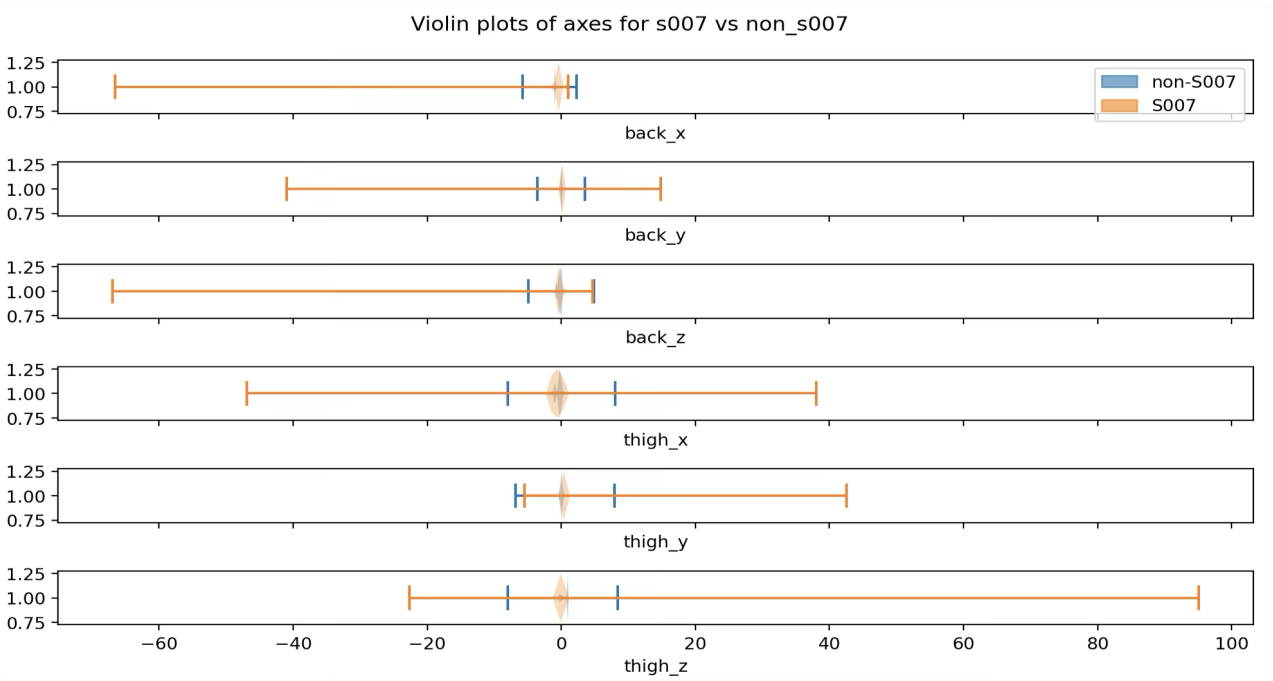

Figure 3: Spurious outliers in unmodified data from all sensors, aggregated, with sensor S007 separated.

3.3 Remove spurious outliers

Outliers are removed for any sensor acceleration values where the value is above the rated maximum (8g) for the AX3 accelerometer, in any axis, by setting to null and then interpolating all nulls linearly. Forward- and backward-fills ensure that reasonable values exist at both ends of the datasets.

With the outliers removed, the distribution for the entire dataset is shown below (thigh-only for brevity). There are still extremities at +-8g, but the significant distributions are much more centralised at +-2g. This is the cleaned dataset used for all subsequent analysis.

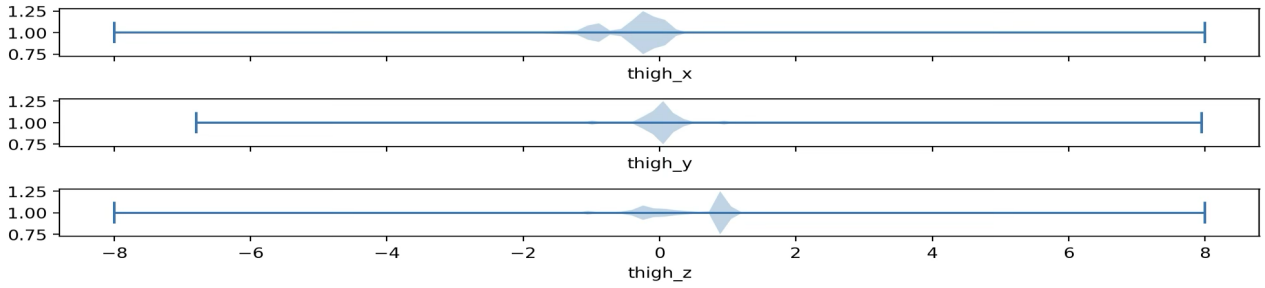

Figure 4: Data distribution from all sensors, aggregated, after outliers removed.

The following box-plot shows the distributions by activity. It provides an indication of which sensors will provide discrimination for each activity - for example, sensor back-x shows a marked difference in data for activity 8 (lying) compared to activity 7 (sitting), but this is not seen in sensor back-z. Additionally, the box-plot clearly shows that for both back and thigh sensors, the x-axis (aligned along the limb or upper body) raw median data are displaced to -1g, showing the constant effect of gravity. The exceptions are for activity 8 (lying) where both sensors are aligned horizontally, and for activities 7 (sitting) where only the thigh sensor is horizontal.

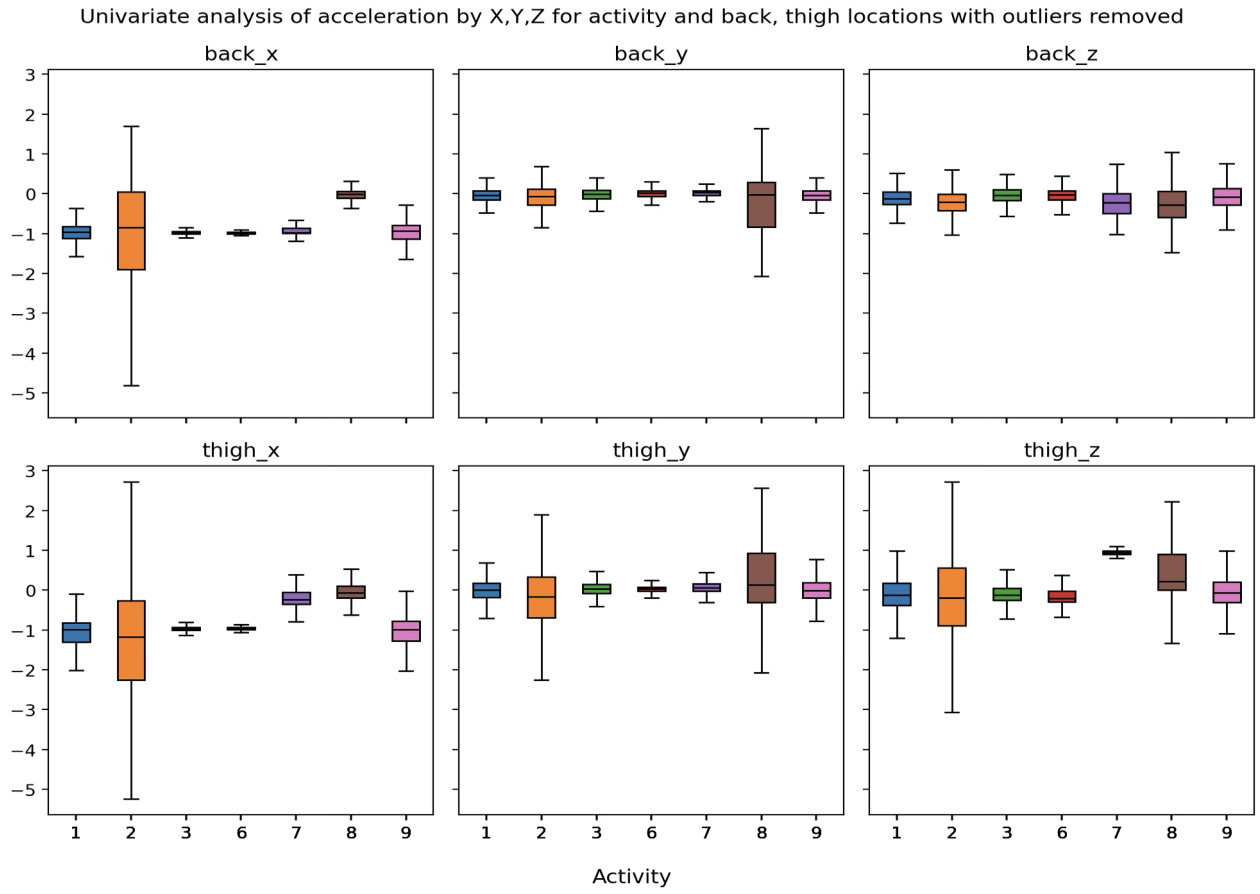

Figure 5: Sensor data distribution by activity.

3.4 Information loss with ENMO

There are many data rows where the Euclidean Norm Minus One (ENMO) is less than zero.

```
Percentage of back sensor readings with ENMO < 0 (normally truncated): 47.91%
```

This could be possible during periods of downward step, where the x-axis (vertical) measured acceleration is less than 1 (i.e. the thigh is being forced down) while y- and z-axis acceleration is close to zero, or could be due to incorrect sensor calibration. Truncating negative values to zero (standard in much research) causes information loss and reduces the ability to identify activities. It was decided that for label fitting, plain ENMO (the Euclidean mean minus one, regardless of whether the resulting value was less than zero) should be calculated, not the formula max(ENMO, 0).

3.5 Discontinuities in the data

As previously seen in the sensor S012 walking data, there are gaps in the data timestamps when considering a regular 50Hz sampling rate. This means that subsequently generated windows can either be a fixed number of rows or length of time, not both. For example, sensor S006 shows a 3-second gap at 00h03m08s:

```
                    timestamp    back_x    back_y    back_z  label sensor
18670 2019-01-12 00:03:08.360 -0.976309  0.023846  0.155131  6   S006
18671 2019-01-12 00:03:11.610 -0.994317 -0.009432  0.064764  6   S006
```

It was decided not to attempt to interpolate and fill this missing data, because the time period was so long that any interpolation would have hidden the true pattern seen in other data segments. This issue was subsequently addressed by using two different windowing functions, to generate windows of fixed time interval or fixed data length (although of course fixed data length windows may span longer than 2s).

3.6 Remove particularly egregious S007 constant data

In order to preserve as much data as possible, it was decided to remove a single section of invalid sensitive data from the S007 sensor, using the following mask:

```
_mask = ~(
  (merged_stairs.sensor == 'S007') &
  (merged_stairs.timestamp >= pd.Timestamp('2019-01-17 00:00:41.840')) &
  (merged_stairs.timestamp <= pd.Timestamp('2019-01-17 00:01:25.660'))
)
```

3.7 Report on dataset size

The complete cleaned dataset size (in terms of rows and 9 columns of timestamp, sensor, 6 axes, label) is:

```
(5054657, 9)
```

3.8 Windowing function for 2s of data

The windows were generated with strict 2s duration and 1s overlap. The windows were generated after grouping by sensor; for cross-validation, grouping by time or subject prevents overlap leakage in sliding windows. The following windows were created for training and test data, using the same approach to cleaning outliers in both sets:

```
Train data window count: 94707
Test data window count: 22256
```

4 Pipelines

4.1 Establish a baseline with an initial set of 10 features

Cross validation: The 'holdout' method was used for cross validation. This relied on the existing split in training and test data which had been supplied. Although the holdout method is simple, it works well with large datasets such as this, and does allow the full training and test sets to be used. Care was taken to segregate the datasets and ensure that training data was never used to test, and test data never used to train, to minimise the risk of over-fitting. Note that it assumes that there is no bias in the way in which training and test datasets had been split originally.

A superset of features was built in one go. Looking at box-plot univariate analysis, there are clear differences in max and min values by activity, but some of the data are spurious and truncated at +-8g, so max and min was less discriminating (and also susceptible to participant body dimensions). Instead, for a given sensor and axis, the 10-percentile and 90-percentile value within the window was taken, as well as the median and standard deviation of values in the window. The initial set of features chosen was from the thigh sensor (which showed greater variety and differentiability than the back sensor):

```
THIGH_X_FEATURES = ['thigh_x_10p', 'thigh_x_median', 'thigh_x_90p', 'thigh_x_std']
THIGH_Y_FEATURES = ['thigh_y_10p', 'thigh_y_median', 'thigh_y_90p', 'thigh_y_std']
THIGH_Z_FEATURES = ['thigh_z_10p', 'thigh_z_median', 'thigh_z_90p', 'thigh_z_std']
THIGH_ENMO_FEATURES = ['thigh_enmo_10p', 'thigh_enmo_median', 'thigh_enmo_90p', 'thigh_enmo_std']
THIGH_10_FEATURES = THIGH_X_FEATURES + THIGH_Z_FEATURES + THIGH_ENMO_MS
```

With these decisions, each window of data was converted into an aggregate view per axis, along with ENMO (Euclidean Mean Minus One) calculated from all three axes together. In total, 30 features were created for later experimentation, although initially, only THIGH-10-FEATURES were used.

```
Training data window summaries: (94707, 34)
Test data window summaries: (22256, 34)
```

5 K-Means

K-Means++ was chosen as the unsupervised method. It creates computed clusters, iteratively assigning points to closest clusters and then updating cluster centres, often used to annotate unlabelled data for subsequent supervised training. In this case, however, data is already labelled, so only the training data were used, and K-Means was assessed by comparing the proposed against the ground-truth label.

5.1 Baseline

Initially the inertia (sum of squares of distance from points to corresponding cluster centres) was plotted to identify the best number of clusters. The elbow method of inertia was evaluated to optimise clusters, and showed an inflection point at 3 clusters. In an unsupervised setting, 3 clusters would be optimum, but since the

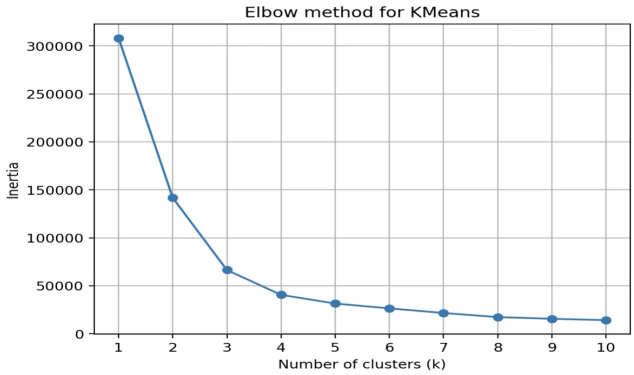

Figure 6: K-Means elbow plot.

data was labelled with 7 ground-truth labels, 7 clusters were chosen. Additionally, because K-Means assigns an arbitrary label to clusters, those identified were mapped to labels on a best-fit basis using op.linear-sum-assignment(). For THIGH-10-FEATURES, performance was poor:

```
K-Means 7 clusters, THIGH_10_FEATURES, n_clusters=7, init='k-means++', n_init='auto', random_state=42.
For label 1: Precision 0.75, Recall 0.45   For label 2: Precision 0.99, Recall 0.75
For label 3: Precision 0.00, Recall 0.00   For label 6: Precision 0.62, Recall 0.96
For label 7: Precision 0.95, Recall 0.98   For label 8: Precision 1.00, Recall 0.55
For label 9: Precision 0.06, Recall 0.17
```

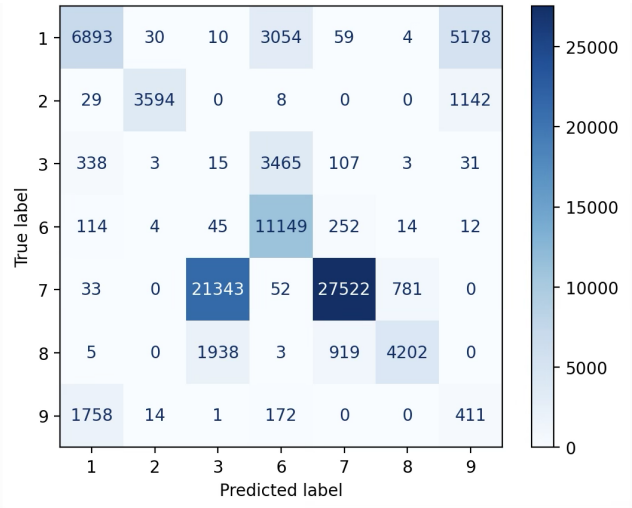

Figure 7: K-Means baseline confusion matrix.

Commentary: precision and recall were poor for activities 3 (shuffling) and 9 (stairs). Shuffling was associated with standing, while sitting was misclassified as shuffling. Walking was frequently classified as stairs. Looking at the sensor box plot for shuffling/standing and walking/stairs, there is a clear similarity between max/min, percentile and median for these pairs and corresponding body movements for given activities.

5.2 Fine-tuning

A large number of different combinations were tested to see which demonstrated improvement, particularly for classes 3 and 9. The combinations involved different groups of features, including up to 32 (all THIGH, all

BACK, all ENMO, as well as different initialisations and algorithms:

```
K-Means 7 clusters, BACK_ENMO_FEATURES, n_clusters=7, init='random', n_init=100
For label 3: Precision 0.09, Recall 0.15 For label 9: Precision 0.11, Recall 0.19
K-Means 7 clusters, BACK_ENMO_FEATURES, n_clusters=7, init='k-means++', n_init=100
For label 3: Precision 0.09, Recall 0.14 For label 9: Precision 0.11, Recall 0.18
K-Means 7 clusters, BACK_10_FEATURES, n_clusters=7, init='k-means++', n_init=100
For label 3: Precision 0.00, Recall 0.01 For label 9: Precision 0.00, Recall 0.00
K-Means 7 clusters, ALL_FEATURES, n_clusters=7, init='k-means++', n_init=100
For label 3: Precision 0.00, Recall 0.00 For label 9: Precision 0.00, Recall 0.00
K-Means 7 clusters, THIGH_10_FEATURES, n_clusters=7, init='k-means++', n_init=100
For label 3: Precision 0.00, Recall 0.00 For label 9: Precision 0.06, Recall 0.17
K-Means 7 clusters, BACK_10_FEATURES + THIGH_10_FEATURES, n_clusters=7, n_init=100
For label 3: Precision 0.00, Recall 0.00 For label 9: Precision 0.00, Recall 0.00
K-Means 7 clusters, BACK_10_FEATURES + THIGH_10_FEATURES, n_clusters=7, n_init=1
For label 3: Precision 0.00, Recall 0.00 For label 9: Precision 0.20, Recall 0.74
K-Means 7 clusters, BACK_10_FEATURES + THIGH_10_FEATURES, n_clusters=7, n_init=1, algorithm='elkan'.
For label 3: Precision 0.00, Recall 0.00 For label 9: Precision 0.20, Recall 0.74
```

The best performance was with BACK-10-FEATURES + THIGH-10-FEATURES, with the following performance and confusion matrix:

```
KMeans(n_clusters=7, init='k-means++', n_init='auto', random_state=42)
For label 1: Precision 0.80, Recall 0.38   For label 2: Precision 0.99, Recall 0.81
For label 3: Precision 0.00, Recall 0.00   For label 6: Precision 0.62, Recall 0.96
For label 7: Precision 0.97, Recall 0.75   For label 8: Precision 0.84, Recall 0.60
For label 9: Precision 0.20, Recall 0.74
```

Commentary: The performance objective was met. It can be seen that the performance of label 9 (stairs) has improved dramatically, with lower misclassification of stairs as walking, but with a remaining misclassification of walking as stairs.

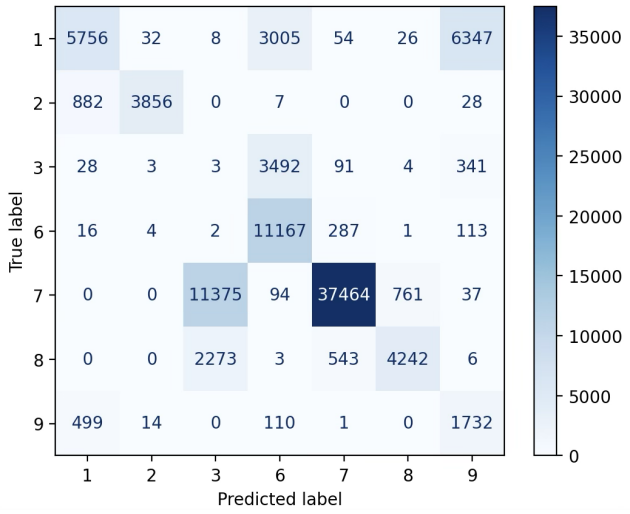

Figure 8: K-Means best configuration confusion matrix.

6 Random Forest

Random Forest was selected as the first supervised learning method.

6.1 Baseline

The training X and y datasets were prepared for both training and test data. Label and timestamp columns were removed.

```
_rf_x_train = window_summaries.drop(['timestamp', 'label'], axis=1, inplace=False)
_rf_y_train = window_summaries['label']
```

The following performance was observed initially using THIGH-X-FEATURES:

```
For label 1: Precision 0.86, Recall 0.94   For label 2: Precision 0.90, Recall 0.96
For label 3: Precision 0.35, Recall 0.37   For label 6: Precision 0.87, Recall 0.84
For label 7: Precision 0.85, Recall 0.99   For label 8: Precision 0.87, Recall 0.08
For label 9: Precision 0.00, Recall 0.00
```

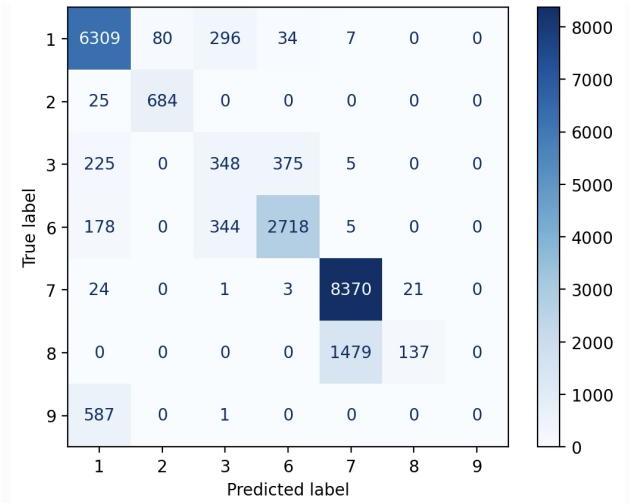

Figure 9: Random Forest baseline confusion matrix.

Commentary: precision is again lower for 3 (shuffling) and 9 (stairs), although early performance was better than with K-Means. Of note, stairs was predicted as walking, and lying was predicted as sitting (perhaps understandable, since the thigh sensor x-axis orientation would be the same for sitting and lying).

6.2 Fine-tuning

As with K-Means, a large number of combinations of features and forest initiations were tried in order to understand factors affecting classification.

```
n_estimators=100, ALL_FEATURES
For label 3: Precision 0.33, Recall 0.56    For label 9: Precision 0.44, Recall 0.58
n_estimators=10, ALL_FEATURES
For label 3: Precision 0.30, Recall 0.53    For label 9: Precision 0.42, Recall 0.52
n_estimators=100, THIGH_10_FEATURES
For label 3: Precision 0.37, Recall 0.45    For label 9: Precision 0.59, Recall 0.52
n_estimators=100, THIGH_X_FEATURES
For label 3: Precision 0.31, Recall 0.38    For label 9: Precision 0.32, Recall 0.23
n_estimators=100, criterion='log_loss', THIGH_X_FEATURES
For label 3: Precision 0.32, Recall 0.38    For label 9: Precision 0.32, Recall 0.23
n_estimators=100, criterion='gini', max_depth=5, THIGH_X_FEATURES
For label 3: Precision 0.37, Recall 0.43    For label 9: Precision 0.00, Recall 0.00
n_estimators=100, THIGH_X_FEATURES, min_samples_leaf=1000
For label 3: Precision 0.35, Recall 0.37    For label 9: Precision 0.00, Recall 0.00
```

Commentary: The best performance on the hard-to-classify activities of 3 (shuffling) and 9 (stairs) was produced by simply using a large number of features (ALL-FEATURES). The confusion matrix is shown. Changing criterion, tree depth, and minimum number of leaf samples did not provide a noticeable improvement. The performance objective was met:

```
For label 1: Precision 0.86, Recall 0.95   For label 2: Precision 0.99, Recall 0.97
For label 3: Precision 0.40, Recall 0.39   For label 6: Precision 0.87, Recall 0.86
For label 7: Precision 1.00, Recall 1.00   For label 8: Precision 1.00, Recall 1.00
For label 9: Precision 0.00, Recall 0.00
```

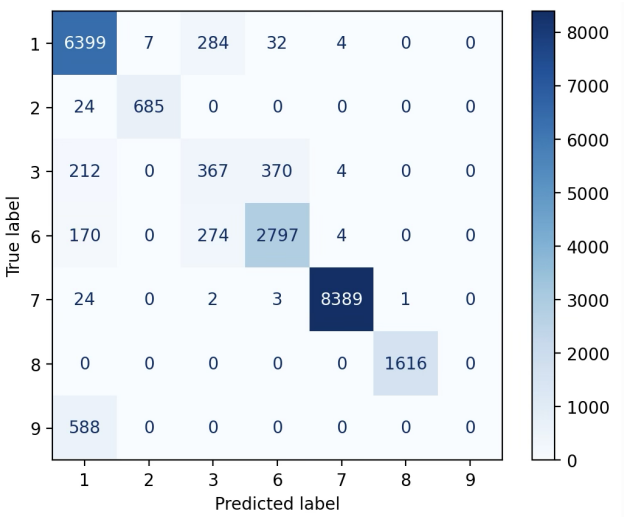

Figure 10: Random Forest optimised confusion matrix.

7 Convolutional Neural Network (CNN)

7.1 Baseline

The initial implementation of a Convolutional Neural Network was as simple as possible and involves a 1D convolutional layer, a pooling and flattening layer, and then 2 fully-connected layers. The structure and confusion matrix are shown:

Layer (type)	Output Shape	Param #
conv1d_62 (Conv1D)	(None, 100, 32)	608
max_pooling1d_51 (MaxPooling1D)	(None, 50, 32)	0
flatten_10 (Flatten)	(None, 1600)	0
dense_37 (Dense)	(None, 128)	204,928
dense_38 (Dense)	(None, 7)	903

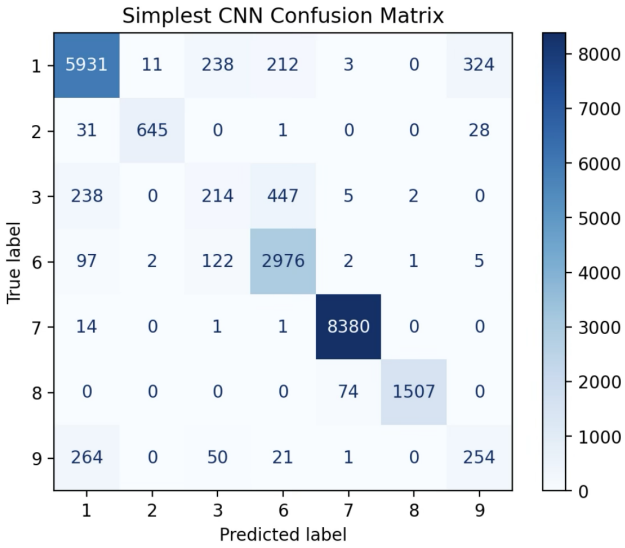

Figure 11: Simplest CNN structure and CNN baseline confusion matrix.

The training was seen to progress properly, with an improvement in accuracy and a reduction in loss:

```
Epoch 1/10 3159/3159  4s 994us/step - accuracy: 0.9167 - loss: 0.2360
Epoch 2/10 3159/3159  3s 986us/step - accuracy: 0.9363 - loss: 0.1804
Epoch 3/10 3159/3159  3s 981us/step - accuracy: 0.9425 - loss: 0.1613
Epoch 4/10 3159/3159  3s 977us/step - accuracy: 0.9471 - loss: 0.1479
Epoch 5/10 3159/3159  3s 954us/step - accuracy: 0.9510 - loss: 0.1363
Epoch 6/10 3159/3159  3s 968us/step - accuracy: 0.9546 - loss: 0.1253
Epoch 7/10 3159/3159  3s 954us/step - accuracy: 0.9573 - loss: 0.1165
Epoch 8/10 3159/3159  3s 958us/step - accuracy: 0.9615 - loss: 0.1075
```

```
Epoch 9/10 3159/3159  3s 971us/step - accuracy: 0.9646 - loss: 0.0988
Epoch 10/10 3159/3159  3s 974us/step - accuracy: 0.9674 - loss: 0.0911
```

The performance objective was met, with confusion by label shown:

```
For label 1: Precision 0.92, Recall 0.90   For label 2: Precision 0.95, Recall 0.95
For label 3: Precision 0.40, Recall 0.25   For label 6: Precision 0.81, Recall 0.93
For label 7: Precision 0.98, Recall 1.00   For label 8: Precision 1.00, Recall 0.93
For label 9: Precision 0.51, Recall 0.52
```

7.2 Fine-tuning: More Complex CNN

A more complex CNN model was developed, involving multiple convolutional layers. The improved performance is depicted:

```
For label 1: Precision 0.95, Recall 0.83   For label 2: Precision 0.99, Recall 0.88
For label 3: Precision 0.42, Recall 0.64   For label 6: Precision 0.92, Recall 0.85
For label 7: Precision 0.99, Recall 1.00   For label 8: Precision 1.00, Recall 0.93
For label 9: Precision 0.40, Recall 0.88
```

The structure and confusion matrix are shown:

Layer (type)	Output Shape	Param #
conv1d_63 (Conv1D)	(None, 100, 32)	1,568
batch_normalization_51 (BatchNormalization)	(None, 100, 32)	128
max_pooling1d_52 (MaxPooling1D)	(None, 50, 32)	0
dropout_45 (Dropout)	(None, 50, 32)	0
conv1d_64 (Conv1D)	(None, 50, 64)	10,304
batch_normalization_52 (BatchNormalization)	(None, 50, 64)	256
max_pooling1d_53 (MaxPooling1D)	(None, 25, 64)	0
dropout_46 (Dropout)	(None, 25, 64)	0
conv1d_65 (Conv1D)	(None, 25, 128)	24,704
batch_normalization_53 (BatchNormalization)	(None, 25, 128)	512
global_average_pooling1d_10 (GlobalAveragePooling1D)	(None, 128)	0
dropout_47 (Dropout)	(None, 128)	0
dense_39 (Dense)	(None, 128)	16,512
dropout_48 (Dropout)	(None, 128)	0
dense_40 (Dense)	(None, 7)	903

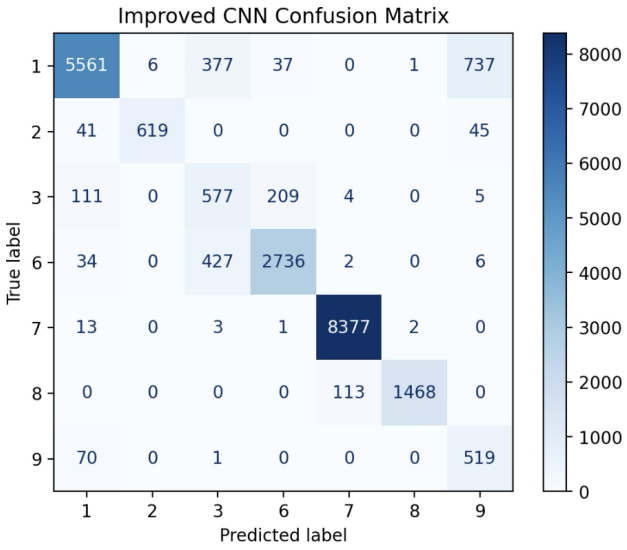

Figure 12: More complex CNN structure and improved confusion matrix.

Commentary: Compared with the simpler CNN, the deeper model had marginally higher precision scores across all labels, but did not meet the original 75% target for all activities. The recall was particularly improved for tricky activities 3 (shuffling) and 9 (stairs), and demonstrated a 10% improvement here.

7.3 Fine-tuning: Hybrid CNN / RNN

A hybrid model was tested. This was a combination of initial convolutional layers, followed by a LSTM RNN structure. It was hoped that the convolutional layers would capture the spatial structure of the sensor data, while the LSTM would capture the time dependencies.

This more advanced network design did not improve performance markedly:

```
For label 1: Precision 0.96, Recall 0.83   For label 2: Precision 0.99, Recall 0.88
For label 3: Precision 0.43, Recall 0.46   For label 6: Precision 0.79, Recall 0.93
For label 7: Precision 0.99, Recall 0.96   For label 8: Precision 1.00, Recall 0.93
For label 9: Precision 0.41, Recall 0.90
```

Layer (type)	Output Shape	Param #
conv1d_59 (Conv1D)	(None, 100, 64)	1,984
batch_normalization_48 (BatchNormalization)	(None, 100, 64)	256
re_lu_15 (ReLU)	(None, 100, 64)	0
max_pooling1d_48 (MaxPooling1D)	(None, 50, 64)	0
conv1d_60 (Conv1D)	(None, 50, 128)	41,088
batch_normalization_49 (BatchNormalization)	(None, 50, 128)	512
re_lu_16 (ReLU)	(None, 50, 128)	0
max_pooling1d_49 (MaxPooling1D)	(None, 25, 128)	0
conv1d_61 (Conv1D)	(None, 25, 128)	49,280
batch_normalization_50 (BatchNormalization)	(None, 25, 128)	512
re_lu_17 (ReLU)	(None, 25, 128)	0
max_pooling1d_50 (MaxPooling1D)	(None, 12, 128)	0
reshape_2 (Reshape)	(None, 12, 128)	0
bidirectional_4 (Bidirectional)	(None, 12, 256)	263,168
bidirectional_5 (Bidirectional)	(None, 128)	164,352
dropout_44 (Dropout)	(None, 128)	0
dense_36 (Dense)	(None, 7)	903

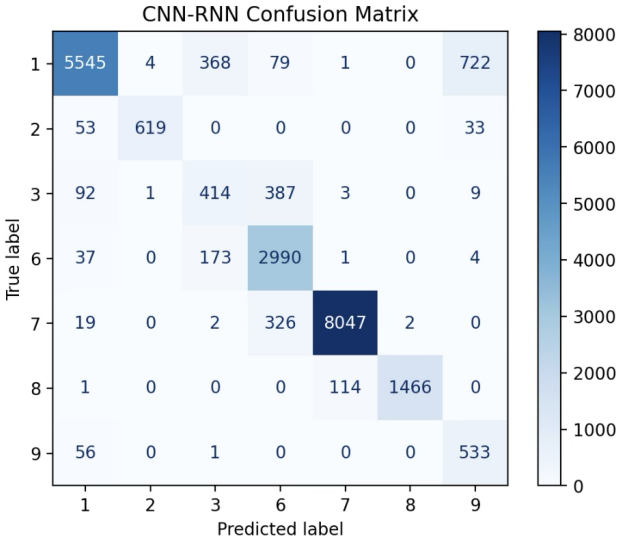

Figure 13: Hybrid CNN-RNN structure and resulting confusion matrix.

Commentary: The scores are marginally improved again compared with the plain deep CNN, except for a decline in recall for activity 3 (shuffling) and a decline in precision for activity 6 (standing).

8 Conclusion

All three approaches (unsupervised K-Means, supervised Random Forest and Convolutional Neural Network) showed a 10% improvement over the baseline when different features and model configuration were tried.

The best performing model was the deeper CNN, which comfortably met the performance objective of [Precision per class = 75%; Recall per class = 50%]. The addition of LSTM to the CNN did not improve performance across the board compared to the increase in complexity; while some activities' recognition improved, others declined.

The following additional approaches were coded and tested, but not included for brevity:

- Using a high-pass filter to remove the gravity element from sensor readings (particularly the vertically-oriented x-axis). This was tried when the ENMO calculation produced large quantities of zero based on max(0, ENMO), where ENMO itself was negative. In the end, for simplicity, the EMNO max() element was removed.
- Determining the dominant frequency from the signals using Fourier analysis (np.fft.fftfreq library). Although this was thought to be a good determinant of activity, it did not provide significant discrimination when mapped to ground-truth labels, because the frequency of steps for walking, shuffling and stairs is too similar.

Given further freedom to experiment outside the stipulated 2s windows, it would be beneficial to derive windows based on cycle peaks (e.g. for walking, one window per left/right leg stride), in the same way that an oscilloscope triggers on repetitive cycles to show a waveform. The windows would not be of identical length. Then the waveform could be analysed by shape, as seen in the depiction of the walking activity in figure 2.