Human Activity Recognition Using Smartphone Sensor Data: Analysis and Classification using Azure ML and Power BI

Séamus Knightly
1MECE1
University Of Galway
Galway, Ireland
s.knightly1@universityofgalway.ie

Seán Kelly
1MECE1
University Of Galway
Galway, Ireland
s.kelly178@universityofgalway.ie

***Abstract*—EE5127 - Internet Of Things System Design Deliverable IIa and IIb written report. Covering the analysis of the dataset in use, experimentation with Machine Learning in Azure ML, deployment of an Azure ML model, and finally visualisation of data in Power BI.**

***Index Terms*—IoT, Remote Patient Monitoring, Accelerometer, Gyroscope, Classification, Assisted Living**

I. Introduction

Human Activity Recognition (HAR) is an important area in the domain of machine learning and medical computing. Focusing on identification and classification of human physical activities based on sensor data. Data for HAR is collected by a wide range of devices, from stationary room sensing video or radar, to wearable or mobile devices. HAR has applications in fields such as remote patient monitoring and assisted living.

There are a number of large, labelled datasets available for human activity recognition. For this project, the *Human Activity Recognition Using Smartphones* dataset [1] is used. This dataset contains sensor data collected from smartphone accelerometers and gyroscopes, as participants performed activities such walking, sitting, and standing. The observations are represented by multiple time and frequency features extracted from the motion signals. Sensor signals from the device's accelerometer and gyroscope were recorded and processed into 561 time and frequency features.

This dataset is chosen over other datasets, such as HARTH dataset [2] [3] for its simplicity, requiring only 1 worn sensor. And for the simpler set of classifications, which correspond better with scenarios that would arise in a care facility or other healthcare setting.

This paper aims to analyse and classify human activities using the UCI HAR dataset by developing and evaluating multiple machine learning models in Azure Machine Learning Studio. Power BI is employed to visualise data characteristics, feature distributions, and model performance metrics. The results provide insights into the use of sensor based data and machine learning in Human Activity Recognition.

The paper will then additionally describe the steps to deploy and visualise an Azure ML model, which will receive data from an IoT device.

II. Dataset Description

A. Dataset Source

The dataset used in this study is the *Human Activity Recognition Using Smartphones* dataset [1], first introduced in 2013. Created as part of a study on human centered computing [4], the dataset consists of sensor data collected from 30 volunteers aged between 19 and 48 years, performing six basic activities of daily living.

Each participant carried a Samsung Galaxy S II smartphone on the waist. The smartphone's embedded accelerometer and gyroscope were used to capture linear acceleration and angular velocity at a constant sampling rate of 50 Hz. The experiment was carried out in a controlled laboratory environment, with participants following a standardised sequence of activities to ensure consistency across samples. The signals were normalised and bounded within [1, -1], and each feature vector is also standardised.

The introductory paper [4] demonstrated that a multiclass Support Vector Machine (SVM) model could achieve an overall classification accuracy of 96% on this dataset, comparable to or exceeding the performance of systems using specialised wearable sensors. Their work highlighted the feasibility of using simpler accelerometer and gyroscope sensors, such as smart phones, as unobtrusive, affordable, and reliable sensing tools for HAR.

B. Feature Overview

The dataset provides both raw sensor readings and pre-processed feature vectors based on that raw data. Each observation represents a 2.56 second window of sensor data with 50% overlap between adjacent windows, there are 128 readings per window. From each window, 561 features were extracted from the time and frequency domains.

The features include statistical and signal based measures such as mean, standard deviation, median absolute deviation,

signal magnitude area, and correlation between sensor axes. Frequency domain features were obtained using Fast Fourier Transform (FFT) on the windowed signals, used to capture additional important motion characteristics for each activity.

Additionally, subject identifiers (1–30) are included, allowing for subject specific analysis.

In the dataset, each of the recorded activities is encoded numerically as follows: 1–*Walking*, 2–*Walking Upstairs*, 3–*Walking Downstairs*, 4–*Sitting*, 5–*Standing*, and 6–*Laying*.

C. Data Preparation

Prior to analysis, experimentation and model training, a subset of the original dataset features was selected to simplify the analysis and reduce computational overhead. The full UCI HAR dataset contains 561 features derived from both time and frequency transformations. This feature set provides detailed motion characterisation, but it is computationally expensive to process, particularly on the edge in the Rasbperry Pi.

To address this, a representative subset of features was selected, focusing on descriptive statistics (mean, maximum, and minimum values) from the accelerometer and gyroscope signals.

The retained features include the following categories:

- **Time domain features:** body acceleration, body acceleration jerk, gyroscope, and gyroscope jerk (mean, max, and min for each axis)
- **Magnitude features:** signal magnitude for acceleration, jerk, and gyroscope signals
- **Frequency domain features:** selected mean, max, and min values for acceleration, jerk, and gyroscope signals

This reduced dataset comprises 101 variables in total. This reduction improves computational efficiency on the edge where limited processing power and energy consumption are key considerations.

The original study [4] employed an SVM classifier, which required extensive pre-processed features to achieve high accuracy. This work explores modern machine learning techniques capable of capturing non-linear relationships and feature interactions. It is therefore expected that the reliance on an extensive set of features is reduced without sacrificing accuracy.

On the Azure Machine Learning side, the `subject` feature was excluded from model training to generalise the data.

III. Data Analysis

A. Data Characteristics

The class distribution, illustrated in Fig. 1, shows that the dataset is well balanced across the six activity categories. Each activity contains roughly a similar number of samples, ensuring that the classification problem is not biased toward any single class.

Figure 2 plots the mean gyroscope signal along the X-axis against the corresponding mean accelerometer signal. The data forms a horizontally elongated cluster centered near 0.25 on the gyroscope axis and 0 on the accelerometer axis. This indicates that angular velocity varies more widely across activities than linear acceleration. The limited vertical spread

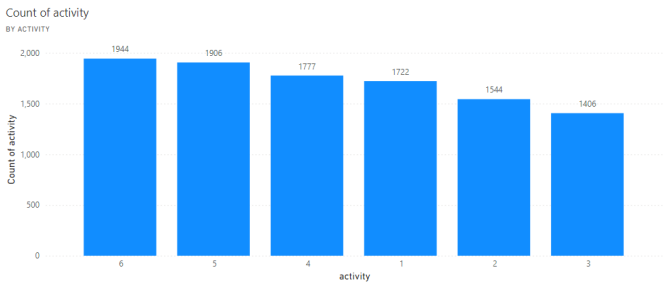


Fig. 1. Distribution of recorded samples across the six activity classes.

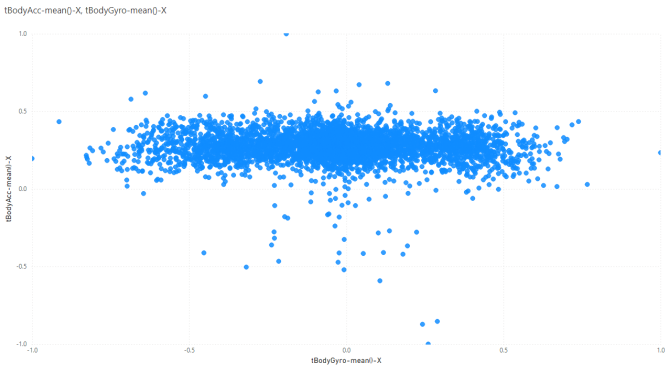


Fig. 2. Scatter plot showing the relationship between the mean body gyroscope signal (`tBodyGyro-mean()-X`) and the mean body acceleration signal (`tBodyAcc-mean()-X`).

reflects the fact that participants do not remain perfectly upright. The positive offset of the gyroscope mean likely arises because most activities involve forward body motion.

Figure 3 shows the distributions of several accelerometer and gyroscope mean features. The features are bounded within $[-1, 1]$, confirming that normalisation was applied to the dataset to remove sensor bias and ensure comparable scaling. The histograms verify that the dataset is in a state suitable for machine learning analysis.

The frequency domain gyroscope features (`fBodyGyro-mean()-X` and `fBodyGyro-mean()-Z`) in the top left of Fig. 3 display an unusual wedge shaped distribution, in contrast to the more symmetric time domain features. This reflects the characteristics of FFT outputs, low frequency components appear more consistently, and high frequency components occur less frequently but with greater variability.

The data shows that there is a complementary relationship between time and frequency domain, with time features capturing instantaneous motion and orientation, frequency features show periodicity and energy characteristics. Including both will give the model better information about the motion patterns.

B. Trends and Patterns

Figure 4 illustrates the mean body acceleration and gyroscope signals across all activities. The mean acceleration

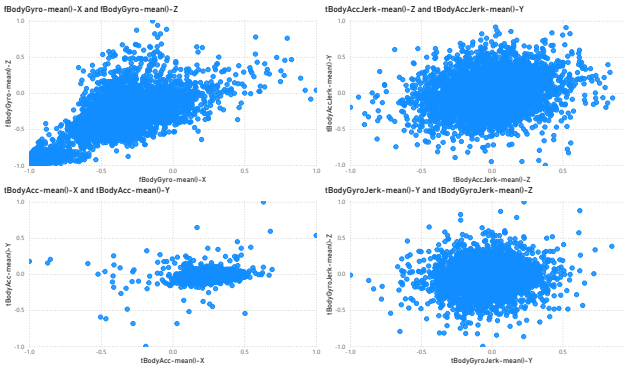


Fig. 3. Histograms of representative accelerometer and gyroscope mean features.

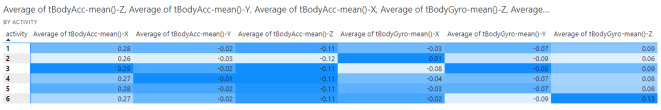


Fig. 4. Heatmap of selected mean accelerometer and gyroscope features across activity classes.

values are relatively uniform across classes. Subtle variations are visible in the gyroscope means, particularly along the Z-axis, corresponding to torso rotation and orientation differences during activities such as walking or laying. These differences are expected given that the device was worn at the waist, where small postural changes and rotational motion are captured more strongly in the gyroscope signals. Overall, the heatmap confirms that the dataset's body motion features capture the small variations between activity types.

C. Feature Comparison

Figure 5 compares the mean magnitudes of selected motion features across all activity classes. These features, including the body acceleration magnitude, gyroscope magnitude, and jerk magnitudes, represent the overall intensity of body motion.

A separation is visible between dynamic and static activities. *Walking*, *Walking Upstairs*, and *Walking Downstairs* (classes 1–3) exhibit notably higher magnitude values across all features, indicating stronger overall motion and rotational activity. Among these, *Walking Downstairs* shows significantly higher jerk magnitudes, due to the more abrupt deceleration and impact forces during descent.

In contrast, static postures such as *Sitting*, *Standing*, and *Laying* (classes 4–6) display consistently lower mean magnitudes across all sensors. The near zero variation among these static classes suggests limited body movement and stable device orientation. These results confirm that magnitude based features can capture the motion intensity of each activity and give discriminative information for classification.

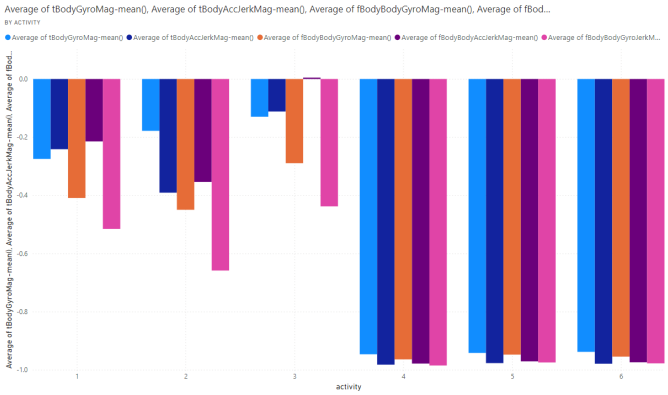


Fig. 5. Comparison of mean motion magnitude features across activities.

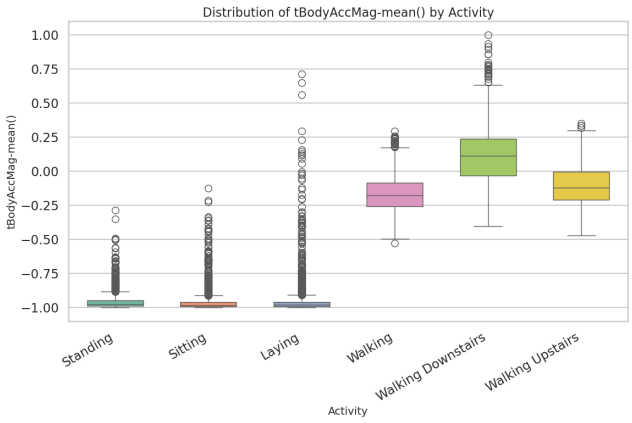


Fig. 6. Box plot of mean body acceleration magnitude (`tBodyAccMag-mean()`) across activity classes. The plot was generated using Matplotlib, as Power BI Online does not currently support box plot creation.

D. Box Plot Analysis

Box plots were generated to compare the distributions of selected features across activity categories (Fig. 6). The mean body acceleration magnitude (`tBodyAccMag-mean()`) has a clear separation between static and dynamic activities. Static postures, *Standing*, *Sitting*, and *Laying*, cluster tightly around -1, indicating near zero net body acceleration after gravity removal. In contrast, dynamic activities (*Walking*, *Walking Upstairs*, and *Walking Downstairs*) show higher medians and broader interquartile ranges, as they have higher motion intensity and variation.

For the dynamic categories, *Walking Downstairs* shows the highest median acceleration magnitude, while *Walking Upstairs* has the greatest spread, consistent with the more irregular motion of climbing. Although this feature alone distinguishes static from dynamic activities, the overlap between static classes (e.g., *Sitting* and *Standing*) suggests that multiple features are required for finer classification.

E. Findings and Discussion

The data analysis highlights several meaningful trends across the dataset. First, the activity classes are well balanced, ensuring that subsequent classification experiments are not biased toward any single class. Normalisation has been applied to remove sensor bias and facilitate comparison across variables.

Feature comparisons show separations between static and dynamic activities across both time and frequency domains. Accelerometer features mostly capture translational body motion, making them effective for distinguishing movement, while gyroscope features show the rotation, which separates similar patterns such as *Walking Upstairs* and *Walking Downstairs*. Frequency features provide extra information about periodicity and energy of movement, which will be useful for classification.

As a whole, the data analysis confirms that the selected features make sense, are physically interpretable, and have distinct patterns across activity classes. These characteristics make the dataset suitable for supervised machine learning models in Azure ML studio.

IV. Machine Learning Model Development and Evaluation

Machine learning models were developed and evaluated in Azure ML Studio using the processed dataset described in Section II-C. A multi-class classification problem was defined, where the target label corresponds to one of six human activity classes. All models were trained using an 70/30 train/test split, and performance was assessed through the use of the `Evaluate Model` component in ML Studio.

The experiments were conducted according to the project specification.

A. Single Feature Models

In the first experiment, 99 models were trained using a single feature each from the dataset. This was done using the *Execute Python Script* component in Azure ML Studio, which was useful to automate the mass training and testing of the models. Logistic Regression was used for all, the code for this can be found in Appendix A

The resulting accuracies ranged from ≈17%, roughly equivalent to a random choice, to ≈ 52%, depending on the selected feature. The most informative features were primarily derived from mean accelerometer and gyroscope signals in the time domain, specifically: `fBodyAcc-max()-X`, `tGravityAcc-min()-Y`, `tGravityAcc-mean()-Y`, `tGravityAcc-max()-Y`, `tBodyAcc-max()-X`, and `fBodyGyro-max()-X`. These activities had accuracies around ≈ 50%, With Gravity related features being over-represented. This matches the physical expectation however, as these features will distinguish the static and dynamic categories. This instantly gives a boost as it reduces the chance to 1 in 3 on a random choice as 3 of the 6 activities are dynamic.

The top performing single feature model achieved an accuracy of approximately 52.18%, using `fBodyAcc-max()-X`. This indicates that horizontal body acceleration alone carries substantial information about activity state, but cannot independently distinguish similar postures such as *Sitting* and *Standing*.

The weakest performing feature was `tBodyAcc-mean()-Y`, which achieved only 17.1% accuracy, which is roughly equivalent to random guessing on the 6 activities. This feature captures minimal variation between activities because the Y-axis component of linear acceleration is effected by the constant gravitational force of the earth.

B. Combination of Features

In the second experiment, a small representative subset of features was used to evaluate how combining motion signals from multiple axes influences classification performance. The selected features represent the mean time domain linear acceleration and angular velocity along the three body axes:

- `tBodyAcc-mean()-X`, `tBodyAcc-mean()-Y`, `tBodyAcc-mean()-Z`
- `tBodyGyro-mean()-X`, `tBodyGyro-mean()-Y`, `tBodyGyro-mean()-Z`

These six features were chosen because they correspond directly to the type of raw sensor readings that would be available on a typical IoT edge device.

A *Multiclass Logistic Regression* classifier was trained using this six-dimensional feature set. The resulting model achieved test accuracy, precision, and recall values of approximately 27%, a poor improvement above random guessing. This relatively poor performance indicates that while these features capture fundamental movement patterns, they do not provide sufficient discriminatory information on their own to differentiate between activities.

The limited accuracy can be attributed to several factors. Firstly, mean values omit temporal variation and frequency information that are important for distinguishing activities with similar average motion but different periodic structures, like *Walking Upstairs* and *Walking Downstairs*. Secondly, Logistic Regression, being a linear model, is not able to capture the non linear interactions between accelerometer and gyroscope signals that make up complex human movements, however, incorporating additional derived features like signal magnitude, minimum and maximum values, and frequency domain statistics, can bring out and encode some of these non linear relationships, allowing linear models to achieve improved performance.

C. All Features

In the third experiment, all 99 available features in the dataset were used to train a *Multiclass Logistic Regression* classifier. This configuration represents the upper bound of model performance using the full set of derived accelerometer and gyroscope features, including both time and frequency statistics such as mean, standard deviation, magnitude, and

TABLE I
FEATURE SELECTION METHOD PERFORMANCE SUMMARY.

Method	Accuracy (%)
Permutation Feature Importance (Decision Forest)	96.1
Filter Based (Pearson Correlation)	59.6
Filter Based (Chi-Squared)	81.7

energy. These features better capture the motion characteristics of human activity.

The trained model achieved an overall test accuracy of 92.8%, with Micro Precision and Micro Recall values of 0.928, and Macro Precision and Macro Recall values of approximately 0.931 and 0.930, respectively. These results indicate good performance across all six activity classes.

Compared to the previous experiments, the improvement in accuracy demonstrates the importance of including the set of pre-processed features. As discussed previously, the additional features expose the non-linear relationships in the motion signals.

This experiment highlights that high performance can be achieved even with a relatively simple classification algorithm when a sufficiently rich feature set is provided. In IoT, this emphasises the benefit of performing lightweight feature extraction either on the gateway or in the cloud, rather than attempting classification directly on limited raw sensor data at the edge.

D. Feature Selection Methods

To reduce dimensionality and identify the most informative features, three selection techniques were evaluated in Azure ML Studio: *Permutation Feature Importance*, and *Filter Based Feature Selection* using the *Pearson Correlation* and *Chi-Squared* criteria (using a `Permutation Feature Importance` component and `Filter Based Feature Selection` components) Each method was followed by model retraining to assess its effect on performance.

Permutation Feature Importance (PFI) was applied using a trained *Multiclass Decision Forest*, achieving an accuracy of 96.1%. The most influential features included `tGravityAcc-min()-X`, `tGravityAcc-min()-Y`, `tBodyAcc-max()-X`, and `fBodyAccMag-max()`.

Using **Filter Based Selection**, the *Pearson Correlation* method selected ten features with the highest linear correlation to activity class, achieving 59.6% accuracy. In contrast, the *Chi-Squared* method achieved a higher accuracy of 81.7%, identifying both time and frequency domain features such as `tBodyAcc-max()-X`, `tGravityAccMag-max()`, and `fBodyAccMag-mean()` as most relevant.

A summary of results is given in Table I. The PFI method with a Decision Forest provided the best overall accuracy. Chi-Squared filtering offered a good compromise between simplicity and accuracy, making it suitable for resource limited IoT applications.

TABLE II
SUMMARY OF MODEL PERFORMANCE ACROSS ALL EXPERIMENTS.

Model / Feature Set	Accuracy (%)
Single Feature (best: `fBodyAcc-max()-X`)	52.2
Six Mean Features (Accelerometer + Gyroscope)	27.0
All 99 Features (Logistic Regression)	92.8
Filter Based (Pearson Correlation)	59.6
Filter Based (Chi-Squared)	81.7
Permutation Feature Importance (Decision Forest)	96.1

V. EXPERIMENT RESULTS COMPARISON AND DISCUSSION

Table II summarises the performance of all experiments conducted in Azure ML Studio. The progression from single features to the full feature set demonstrates a clear relationship between feature richness, model complexity, and classification accuracy.

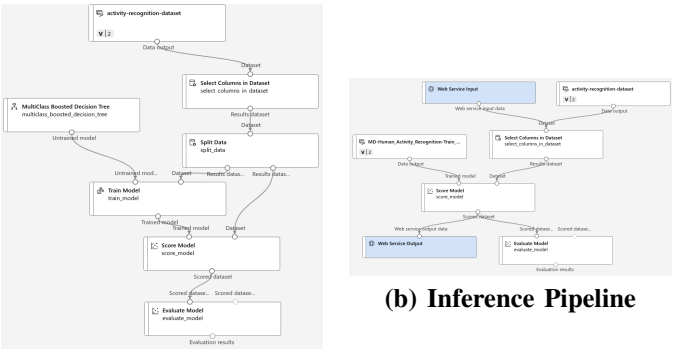
The single-feature and small-subset experiments highlight that limited raw sensor data provide only partial information about human activity. Individual features could distinguish static from dynamic motion but failed to capture finer variations between activities such as *Walking Upstairs* and *Walking Downstairs*. The poor accuracy of the six-feature model (27%) reinforces this limitation and reflects the reduced representational power of mean values that discard temporal and frequency variation.

In contrast, the full feature model achieved 92.8% accuracy, showing that the engineered features available in the dataset significantly improve separability. Derived statistical, magnitude, and frequency domain attributes expose latent non-linear relationships that even a linear classifier like Logistic Regression can exploit. This demonstrates that well-designed feature extraction can compensate for model simplicity.

The feature-selection experiments further confirmed this relationship. The Chi-Squared filter retained the most relevant subset of features, achieving 81.7% accuracy, close to the full model but with much lower dimensionality. The Pearson correlation filter, limited to linear dependencies, performed poorly at 59.6%. The Permutation Feature Importance method, evaluated using a Decision Forest, produced the highest overall accuracy (96.1%), reflecting the capability of ensemble models to capture non-linear feature interactions without explicit feature engineering.

From an IoT system design perspective, these results highlight several key trade offs:

- **Edge vs. Cloud Processing:** Lightweight models on raw sensor data achieve limited accuracy and are suitable only for coarse classification. More complex processing, including feature extraction and model inference, should be offloaded to a cloud platform or high-performance gateway to achieve reliable recognition.
- **Feature Dimensionality:** While the full 99-feature model delivers high accuracy, feature selection can substantially reduce computation and bandwidth costs with minimal performance loss—important for constrained IoT deployments.



(a) Pipeline **(b) Inference Pipeline**

Fig. 7. Pipeline to inference workflow

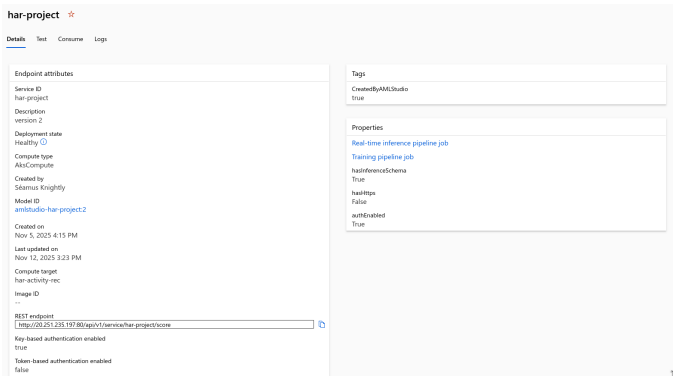


Fig. 8. Details Page for Endpoint

- **Model Choice:** Linear models such as Logistic Regression are interpretable and efficient but limited in expressiveness. Non-linear models like Decision Forests or Neural Networks provide superior performance at the cost of higher computational load.

Overall, the experiments demonstrate that classification accuracy in Human Activity Recognition depends primarily on the availability of informative features. Combining moderate feature selection with non-linear classifiers offers an optimal balance between computational efficiency and predictive accuracy, aligning with the design goals of scalable IoT systems.

VI. Documentation of Web Service Deployment, Use, and Visualisation

A. Deploying a Machine Learning Model as a Web Service

To deploy a machine learning model as a service, the first step is to create a real time inference pipeline from the pipeline that generated the model. Figure 7 shows the design of the original pipeline, and the resultant real time inference pipeline. Two components, `Web Service Input` and `Web Service Output`, have been added, to allow for querying an API and receiving a response.

Once the Real Time Inference Pipeline has been created and completes its submission job, it will need to be deployed to a an endpoint, which will expose the model's prediction logic through a REST API. The endpoint must be created first, and then the inference pipeline must be deployed to it. Once the endpoint is ready, it will be in the final "Healthy" state and inspecting the details page will give a similar output to that shown in Figure 8

After the endpoint becomes healthy, the REST API can be queried directly. Azure also exposes an auto-generated Swagger specification for the endpoint, which documents the request and response formats that the service expects. This specification is useful for understanding the required JSON structure, supported operations, and example payloads. The start of the Swagger documentation for this deployment is shown in Figure 9.

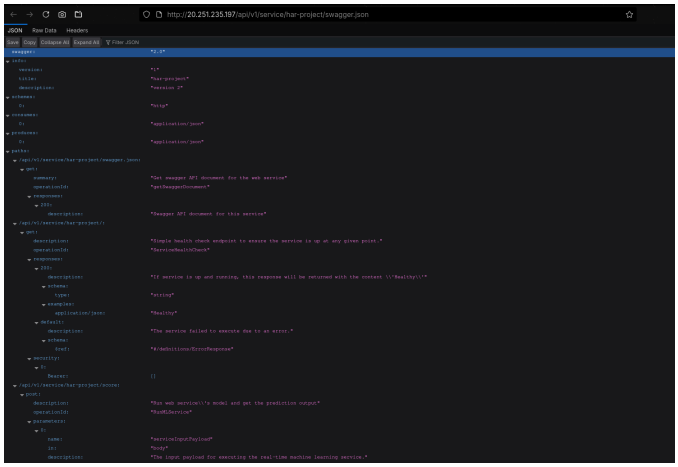


Fig. 9. Swagger Output

B. Creating and Running a Stream Analytics Job

With the model now exposed through an endpoint, the next stage is to create a Azure Stream Analytics (ASA) job.

After creating the ASA job, it needs to be given an input to take in IoT device data, an output to send the data on to Power BI for visualisation, a function to query the REST API, and an SQL query to transfer the information from input to output while also including the function output.

First, create an input, ensure that `IoT Hub` is chosen as the input type, and give it a unique name. The creation window will look like Figure 10. Give it a unique name, and then choose the IoT Hub that the IoT device is sending to.

Next, create a function. When adding the function, choose "Azure ML Service". The creation window will look like Figure 11. Give it a unique name, and take note of the function signature, as this will be used to craft the SQL query.

Before completing the next task, create a workspace in Power BI. Take note of the workspace name. Figure 12 shows what the creation window looks like in Power BI.

Creating the workspace is necessary because Azure Stream Analytics writes its output directly into a Power BI dataset. The workspace selected determines where the real time dataset will appear and where the final visualisations will be built.

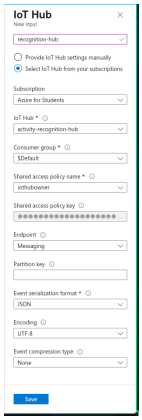


Fig. 10. Creating an ASA input.

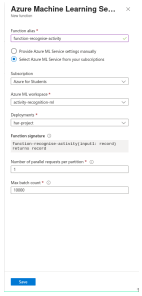


Fig. 11. Creating an ASA function.

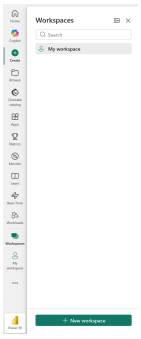


Fig. 12. Power BI Workspace Creation.

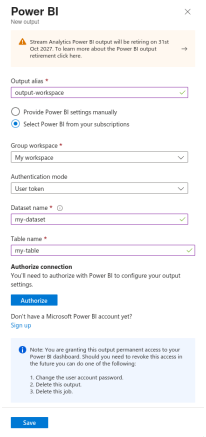


Fig. 13. Adding a Power BI output.

Next, back in the Stream Analytics Job, create an output. Choose Power BI when adding the output. Give a unique name, and choose the workspace from the Power BI account. Give a name to the dataset and the table. There will also be a need to Authorise the connection to Power BI. The creation window is shown in Figure 13.

The Power BI output defines the destination for the model predictions produced by the Azure ML function. Once the connection is authorised, Stream Analytics will push each prediction into the specified dataset and table in real time, allowing Power BI to automatically update dashboards and reports as new IoT messages arrive.

Finally, create an SQL query. This is done last as the names of the inputs, outputs, and functions must be known in order to create the query.

The SQL query acts as the transformation and routing layer between the IoT telemetry, the Azure ML endpoint, and the final output consumed by Power BI. It must reshape the raw device messages into the JSON structure expected by the REST API, call the machine learning function using that payload, and then format the prediction results into a table that can be visualised in Power BI. The full query used in this project is shown in Appendix B, where each clause ensures the correct mapping of fields from the IoT device to the model input and the extraction of the returned prediction.

To note here, Power BI only supports a maximum of 75 columns when it is receiving the data, so the data sent to Power BI only includes a subset of features, and the prediction label. Also note that it is important to send the timestamps for the data, to allow for visualising in Power BI.

The ASA Job is now ready to be started. After starting the job, the overview page will look like Figure 14

Additionally, if storage is required, a blob storage can be added to the outputs section, this is done by choosing Azure blob storage instead when adding an output. The SQL query will also need to be updated to output to both the Power BI visualisation and the blob storage.

C. Visualising in Power BI

Once the IoT device has started transmitting, and the previous steps have been completed, the dataset specified will start receiving data in Power BI.

First, go to the workspace that the data is being sent to, there will now be a new dataset, with the same name as what was specified previously. Figure 15 shows what the workspace might look like now in Power BI.

Next, create a report on the dataset.

With the editing view, the data pane is the right, the data should be visible in the format specified in the SQL query created as part of the ASA job creation step, this can be seen in Figure 16

Now, various visuals for the data are available in the `Visualizations` tab. For this project, two `Line Charts`

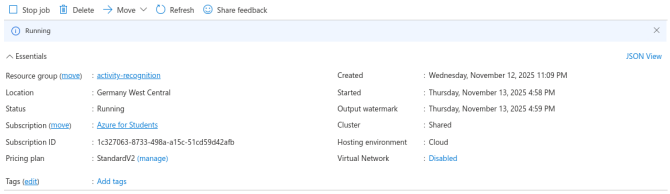


Fig. 14. Stream Analytics Job Overview Page when Job is running.

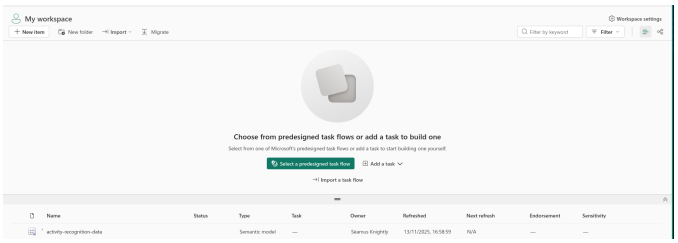


Fig. 15. Power BI Workspace showing the new dataset from the ASA Job.

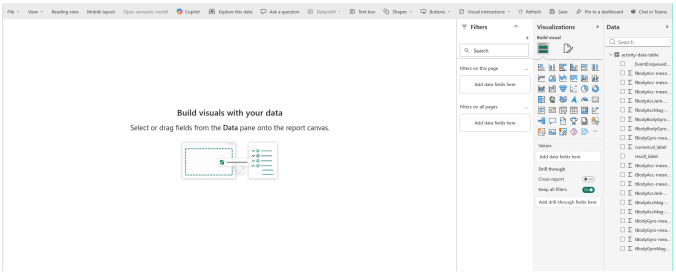


Fig. 16. Power BI Report Creation Workspace with data on the right.

and a `Card` are chosen. These can be dragged into the workspace.

For the `Line Charts`, in the case of the data for this project, the `EventEnqueuedUtcTime` time was chosen for the X-axis for both charts. Then each chart displayed the X, Y and Z axes for Gyroscopic and Accelerometer Data respectively. To make the dashboard display the newest information, rather than showing all historical data, a filter was applied to both of these dashboard to only display data from the past minute. This was done by applying a filter, in the `Filters` tab, on `EventEnqueuedUtcTime` for both charts.

The `Card` was then used to display the last `activity_label`, therefore displaying the current activity.

The final visualisation can be seen in Figure 17.

REFERENCES

[1] J. Reyes-Ortiz, D. Anguita, A. Ghio, L. Oneto, and X. Parra. "*Human Activity Recognition Using Smartphones*", UCI Machine Learning Repository, 2013 [Online]. https://doi.org/10.24432/C54S4K.

[2] A. Logacjov, K. Bach, A. Kongsvold, H. B. Bårdstu, and P. J. Mork, "HARTH: A Human Activity Recognition Dataset for Machine Learning," *Sensors*, vol. 21, no. 23, Art. no. 23, Jan. 2021, doi: 10.3390/s21237853.

[3] K. Bach, A. Logacjov, A. Kongsvold, H. B. Bårdstu, and P. J. Mork, "A Machine Learning Classifier for Detection of Physical Activity Types and Postures During Free-Living," *Journal for the Measurement of Physical Behaviour*, vol. 1, no. aop, pp. 1–8, Dec. 2021, doi: 10.1123/jmpb.2021-0015.

[4] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A Public Domain Dataset for Human Activity Recognition Using Smartphones," *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2013)*, Bruges, Belgium, pp. 437–442, Apr. 2013.

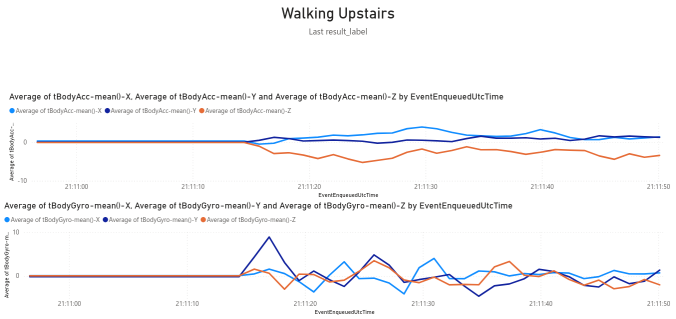


Fig. 17. Final Power BI Real Time Visualisation

APPENDIX A
SCRIPT USED FOR SINGLE FEATURE EVALUATION IN EXECTUTE PYTHON SCRIPT COMPONENT

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# The script MUST contain a function named azureml_main
# which is the entry point for this module.


# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be None.
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):

df = dataframe1.copy() # copy dataset
y = df['activity'] # activity is the category
X = df.drop(columns=['activity']) # Dataset

results = []

# iterate over the features available
for feature in X.columns:
        X_single = X[[feature]]
        X_train, X_test, y_train, y_test = train_test_split(
                X_single, y, test_size=0.2, random_state=42
        )
        # Train a logistic regression model
        model = LogisticRegression(max_iter=1000)
        model.fit(X_train, y_train)
        # grab the accuracy
        acc = accuracy_score(y_test, model.predict(X_test))
        results.append({'Feature': feature, 'Accuracy': acc})

results_df = pd.DataFrame(results).sort_values(by='Accuracy', ascending=False)

# output the results dataframe
return results_df,
```

APPENDIX B
AZURE STREAM ANALYTICS JOB QUERY

```
-- This will include device data and the enqueued time.
WITH FeaturesWithTime AS (
        SELECT
                EventEnqueuedUtcTime,
                activity,
                [tBodyAcc-mean()-X],
                [tBodyAcc-mean()-Y],
                [tBodyAcc-mean()-Z],
                [tBodyAcc-max()-X],
                [tBodyAcc-max()-Y],
                [tBodyAcc-max()-Z],
                [tBodyAcc-min()-X],
                [tBodyAcc-min()-Y],
                [tBodyAcc-min()-Z],
                [tBodyAccJerk-mean()-X],
                [tBodyAccJerk-mean()-Y],
                [tBodyAccJerk-mean()-Z],
                [tBodyAccJerk-max()-X],
                [tBodyAccJerk-max()-Y],
                [tBodyAccJerk-max()-Z],
```

```
[tBodyAccJerk-min()-X],
[tBodyAccJerk-min()-Y],
[tBodyAccJerk-min()-Z],
[tBodyGyro-mean()-X],
[tBodyGyro-mean()-Y],
[tBodyGyro-mean()-Z],
[tBodyGyro-max()-X],
[tBodyGyro-max()-Y],
[tBodyGyro-max()-Z],
[tBodyGyro-min()-X],
[tBodyGyro-min()-Y],
[tBodyGyro-min()-Z],
[tBodyGyroJerk-mean()-X],
[tBodyGyroJerk-mean()-Y],
[tBodyGyroJerk-mean()-Z],
[tBodyGyroJerk-max()-X],
[tBodyGyroJerk-max()-Y],
[tBodyGyroJerk-max()-Z],
[tBodyGyroJerk-min()-X],
[tBodyGyroJerk-min()-Y],
[tBodyGyroJerk-min()-Z],
[tBodyAccMag-mean()],
[tBodyAccMag-max()],
[tBodyAccMag-min()],
[tBodyAccJerkMag-mean()],
[tBodyAccJerkMag-max()],
[tBodyAccJerkMag-min()],
[tBodyGyroMag-mean()],
[tBodyGyroMag-max()],
[tBodyGyroMag-min()],
[tBodyGyroJerkMag-mean()],
[tBodyGyroJerkMag-max()],
[tBodyGyroJerkMag-min()],
[fBodyAcc-mean()-X],
[fBodyAcc-mean()-Y],
[fBodyAcc-mean()-Z],
[fBodyAcc-max()-X],
[fBodyAcc-max()-Y],
[fBodyAcc-max()-Z],
[fBodyAcc-min()-X],
[fBodyAcc-min()-Y],
[fBodyAcc-min()-Z],
[fBodyAccJerk-mean()-X],
[fBodyAccJerk-mean()-Y],
[fBodyAccJerk-mean()-Z],
[fBodyAccJerk-max()-X],
[fBodyAccJerk-max()-Y],
[fBodyAccJerk-max()-Z],
[fBodyAccJerk-min()-X],
[fBodyAccJerk-min()-Y],
[fBodyAccJerk-min()-Z],
[fBodyGyro-mean()-X],
[fBodyGyro-mean()-Y],
[fBodyGyro-mean()-Z],
[fBodyGyro-max()-X],
[fBodyGyro-max()-Y],
[fBodyGyro-max()-Z],
[fBodyGyro-min()-X],
[fBodyGyro-min()-Y],
[fBodyGyro-min()-Z],
[fBodyAccMag-mean()],
[fBodyAccMag-max()],
[fBodyAccMag-min()],
[fBodyBodyAccJerkMag-mean()],
[fBodyBodyAccJerkMag-max()],
[fBodyBodyAccJerkMag-min()],
[fBodyBodyGyroMag-mean()],
```

```
                [fBodyBodyGyroMag-max()],
                [fBodyBodyGyroMag-min()],
                [fBodyBodyGyroJerkMag-mean()],
                [fBodyBodyGyroJerkMag-max()],
                [fBodyBodyGyroJerkMag-min()]
        FROM
                [activity-recognition-hub]
),
-- This will be passed into the function.
Features AS (
        SELECT
                activity,
                [tBodyAcc-mean()-X],
                [tBodyAcc-mean()-Y],
                [tBodyAcc-mean()-Z],
                [tBodyAcc-max()-X],
                [tBodyAcc-max()-Y],
                [tBodyAcc-max()-Z],
                [tBodyAcc-min()-X],
                [tBodyAcc-min()-Y],
                [tBodyAcc-min()-Z],
                [tBodyAccJerk-mean()-X],
                [tBodyAccJerk-mean()-Y],
                [tBodyAccJerk-mean()-Z],
                [tBodyAccJerk-max()-X],
                [tBodyAccJerk-max()-Y],
                [tBodyAccJerk-max()-Z],
                [tBodyAccJerk-min()-X],
                [tBodyAccJerk-min()-Y],
                [tBodyAccJerk-min()-Z],
                [tBodyGyro-mean()-X],
                [tBodyGyro-mean()-Y],
                [tBodyGyro-mean()-Z],
                [tBodyGyro-max()-X],
                [tBodyGyro-max()-Y],
                [tBodyGyro-max()-Z],
                [tBodyGyro-min()-X],
                [tBodyGyro-min()-Y],
                [tBodyGyro-min()-Z],
                [tBodyGyroJerk-mean()-X],
                [tBodyGyroJerk-mean()-Y],
                [tBodyGyroJerk-mean()-Z],
                [tBodyGyroJerk-max()-X],
                [tBodyGyroJerk-max()-Y],
                [tBodyGyroJerk-max()-Z],
                [tBodyGyroJerk-min()-X],
                [tBodyGyroJerk-min()-Y],
                [tBodyGyroJerk-min()-Z],
                [tBodyAccMag-mean()],
                [tBodyAccMag-max()],
                [tBodyAccMag-min()],
                [tBodyAccJerkMag-mean()],
                [tBodyAccJerkMag-max()],
                [tBodyAccJerkMag-min()],
                [tBodyGyroMag-mean()],
                [tBodyGyroMag-max()],
                [tBodyGyroMag-min()],
                [tBodyGyroJerkMag-mean()],
                [tBodyGyroJerkMag-max()],
                [tBodyGyroJerkMag-min()],
                [fBodyAcc-mean()-X],
                [fBodyAcc-mean()-Y],
                [fBodyAcc-mean()-Z],
                [fBodyAcc-max()-X],
                [fBodyAcc-max()-Y],
                [fBodyAcc-max()-Z],
                [fBodyAcc-min()-X],
```

```
                        [fBodyAcc-min()-Y],
                        [fBodyAcc-min()-Z],
                        [fBodyAccJerk-mean()-X],
                        [fBodyAccJerk-mean()-Y],
                        [fBodyAccJerk-mean()-Z],
                        [fBodyAccJerk-max()-X],
                        [fBodyAccJerk-max()-Y],
                        [fBodyAccJerk-max()-Z],
                        [fBodyAccJerk-min()-X],
                        [fBodyAccJerk-min()-Y],
                        [fBodyAccJerk-min()-Z],
                        [fBodyGyro-mean()-X],
                        [fBodyGyro-mean()-Y],
                        [fBodyGyro-mean()-Z],
                        [fBodyGyro-max()-X],
                        [fBodyGyro-max()-Y],
                        [fBodyGyro-max()-Z],
                        [fBodyGyro-min()-X],
                        [fBodyGyro-min()-Y],
                        [fBodyGyro-min()-Z],
                        [fBodyAccMag-mean()],
                        [fBodyAccMag-max()],
                        [fBodyAccMag-min()],
                        [fBodyBodyAccJerkMag-mean()],
                        [fBodyBodyAccJerkMag-max()],
                        [fBodyBodyAccJerkMag-min()],
                        [fBodyBodyGyroMag-mean()],
                        [fBodyBodyGyroMag-max()],
                        [fBodyBodyGyroMag-min()],
                        [fBodyBodyGyroJerkMag-mean()],
                        [fBodyBodyGyroJerkMag-max()],
                        [fBodyBodyGyroJerkMag-min()]
                FROM
                        FeaturesWithTime
        ),

        -- To call the function and get the ML prediction
        Scored AS (
                SELECT
                        FWT.*,
                        ([do-recognise-activity](F)).[Scored Labels] AS result
                FROM FeaturesWithTime FWT
                JOIN Features F
                ON DATEDIFF(second, FWT, F) BETWEEN 0 AND 1
        )

        -- Send to Power BI
        SELECT
                -- Power BI only supports 75 rows.
                Scored.EventEnqueuedUtcTime,
                Scored.[tBodyAcc-mean()-X],
                Scored.[tBodyAcc-mean()-Y],
                Scored.[tBodyAcc-mean()-Z],
                Scored.[tBodyAccJerk-mean()-X],
                Scored.[tBodyGyro-mean()-X],
                Scored.[tBodyGyro-mean()-Y],
                Scored.[tBodyGyro-mean()-Z],
                Scored.[tBodyAccMag-mean()],
                Scored.[tBodyAccMag-max()],
                Scored.[tBodyGyroMag-mean()],
                Scored.[fBodyAcc-mean()-X],
                Scored.[fBodyAcc-mean()-Y],
                Scored.[fBodyAcc-mean()-Z],
                Scored.[fBodyAccJerk-mean()-X],
                Scored.[fBodyGyro-mean()-X],
                Scored.[fBodyBodyGyroMag-mean()],
                Scored.[fBodyAccMag-mean()],
```

```
        Scored.[fBodyBodyGyroJerkMag-mean()],
        result as numerical_label,
        CASE result
                WHEN 1 THEN 'Walking'
                WHEN 2 THEN 'Walking Upstairs'
                WHEN 3 THEN 'Walking Downstairs'
                WHEN 4 THEN 'Sitting'
                WHEN 5 THEN 'Standing'
                WHEN 6 THEN 'Laying'
                ELSE 'Unknown'
        END AS result_label
INTO [My-Workspace]
FROM Scored;

-- Send to Azure Blob storage
SELECT
        Scored.EventEnqueuedUtcTime,
        Scored.[tBodyAcc-mean()-X],
        Scored.[tBodyAcc-mean()-Y],
        Scored.[tBodyAcc-mean()-Z],
        Scored.[tBodyAccJerk-mean()-X],
        Scored.[tBodyGyro-mean()-X],
        Scored.[tBodyGyro-mean()-Y],
        Scored.[tBodyGyro-mean()-Z],
        Scored.[tBodyAccMag-mean()],
        Scored.[tBodyAccMag-max()],
        Scored.[tBodyGyroMag-mean()],
        Scored.[fBodyAcc-mean()-X],
        Scored.[fBodyAcc-mean()-Y],
        Scored.[fBodyAcc-mean()-Z],
        Scored.[fBodyAccJerk-mean()-X],
        Scored.[fBodyGyro-mean()-X],
        Scored.[fBodyBodyGyroMag-mean()],
        Scored.[fBodyAccMag-mean()],
        Scored.[fBodyBodyGyroJerkMag-mean()],
        result as numerical_label,
        CASE result
                WHEN 1 THEN 'Walking'
                WHEN 2 THEN 'Walking Upstairs'
                WHEN 3 THEN 'Walking Downstairs'
                WHEN 4 THEN 'Sitting'
                WHEN 5 THEN 'Standing'
                WHEN 6 THEN 'Laying'
                ELSE 'Unknown'
        END AS result_label
INTO
        [iot-output-blob]
FROM
        Scored;
```