ML4QS:
Predicting User Activities using Sensory Data
Group 28

Zijie Tang (2801446)
Lars de Wolf (2820753)
Vajiheh Moshtagh(2801823)

Vrije Universiteit Amsterdam

1 Introduction

This assignment investigates the utilization of machine learning algorithms to predict user activities by analyzing sensory data captured through daily movements. Our dataset comprises data from accelerometers, gyroscopes, proximity sensors, and location trackers. The objective is to develop machine learning models that can accurately classify various activities, such as walking, running, cycling, driving, and commuting by train. Through this analysis, we aim to uncover the patterns that these sensors capture during different activities, which can have practical implications in diverse fields such as healthcare for patient monitoring and urban planning for enhancing traffic and transit systems. The report documents our methods and processes, starting from data preprocessing, moving through exploratory data analysis, and culminating in the training and validation of our predictive models.

2 Data Exploration

This section presents an exploratory data analysis (EDA) of the bike sensor dataset, chosen as a representative example to illustrate the analytical processes applied to similar datasets, including those for cars, trains, and activities such as running and walking. Due to space constraints, we focus here on the bike dataset, which encompasses data from Accelerometers, Gyroscopes, Linear Accelerometers, Location, and Proximity sensors. The primary objective of this analysis is to explore the data distributions, identify distinct patterns, and detect any anomalies or outliers that could influence subsequent analyses. All data was recorded on the highest frequency possible for each sensor, using the PhyPhox[4] application.

2.1 Descriptive Statistics Summary

The following table summarizes the key descriptive statistics for each sensor in the bike dataset. In a similar manner, we have applied these statistical analysis techniques to other datasets, including those for running, walking, cars, and trains, enabling a comprehensive investigation of our data across various contexts.

Sensor	Axis	Mean	Std Dev	Min	Max
Accelerometer	X (m/s^2)	-1.64	5.26	-35.60	20.31
Accelerometer	Y (m/s^2)	0.12	5.94	-30.12	29.93
Accelerometer	Z (m/s^2)	4.17	4.77	-31.19	59.08
Gyroscope	X (rad/s)	-0.004	0.81	-8.60	7.68
Gyroscope	Y (rad/s)	0.001	0.65	-10.44	14.34
Gyroscope	Z (rad/s)	-0.006	0.64	-6.68	8.98
Linear Accelerometer	X (m/s^2)	-0.018	1.61	-28.28	16.95
Linear Accelerometer	Y (m/s^2)	-0.14	1.31	-21.45	22.16
Linear Accelerometer	Z (m/s^2)	0.06	2.12	-39.57	56.40
Location	Latitude (°)	52.22	0.27	51.59	52.35
Location	Longitude (°)	4.95	0.12	4.87	5.23
Location	Height (m)	3.76	4.22	-6.71	16.65
Location	Velocity (m/s)	2.83	1.36	0.00	6.33
Proximity	Distance (cm)	2.83	2.52	0.00	5.00

Table 1: Descriptive Statistics Summary for Bike Sensor Data

2.2 Visualization and Analysis

To gain further insights, we visualized the data from each sensor. Below are the combined time series and box plots for the Accelerometer, Gyroscope, and Linear Accelerometer. For a general overview, all data from the bike datasets have been compiled and analyzed.

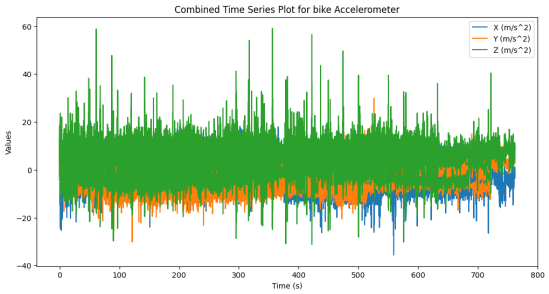


Fig. 1: Combined Time Series Plot for all bike Accelerometer

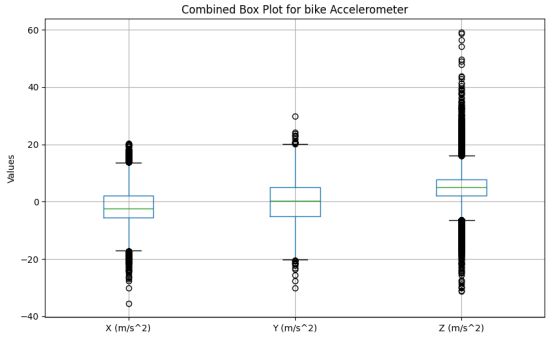


Fig. 2: Combined Box Plot for all bike Accelerometer

Accelerometer Data Analysis: The accelerometer data shows a wide range of values for each axis, indicating varied movement and acceleration patterns during the bike's operation, as seen in Figure 1. The mean acceleration values suggest slight bias in the X and Z directions, while the Y axis shows minimal bias. The box plots, shown in Figure 2, reveal significant outliers, particularly in the Z axis, which may correspond to bumps or sudden movements.

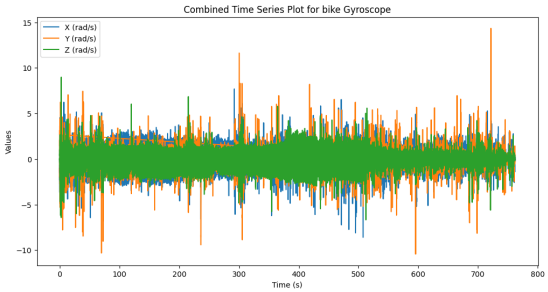


Fig. 3: Combined Time Series Plot for all bike Gyroscope

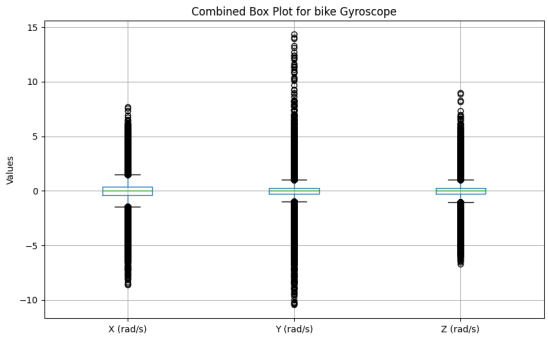


Fig. 4: Combined Box Plot for all bike Gyroscope

Gyroscope Data Analysis: The gyroscope data, as seen in Figure 3, indicates that the bike undergoes various rotational movements. The box plot, Figure 4, highlights the spread and outliers in the rotational speed across all three axes. The X axis shows the least variation, suggesting relatively stable rotation about this axis. The Y axis exhibits higher peak values and more variance, which might be attributed to frequent turns or tilts, indicative of directional changes or adjustments in the bike's path. The Z axis, while similar in range to the X axis, also shows some outliers that may represent occasional abrupt changes in the bike's orientation. These plots collectively illustrate the dynamic nature of the bike's movements and the challenges in capturing rotational data accurately without noise.

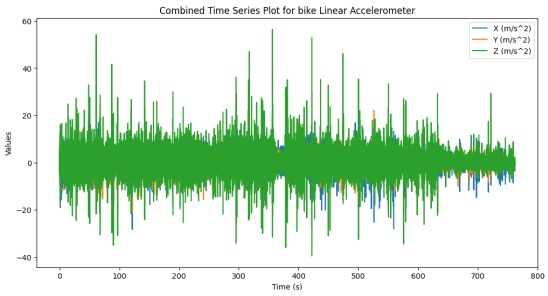


Fig. 5: Combined Time Series Plot for all bike Linear Accelerometer

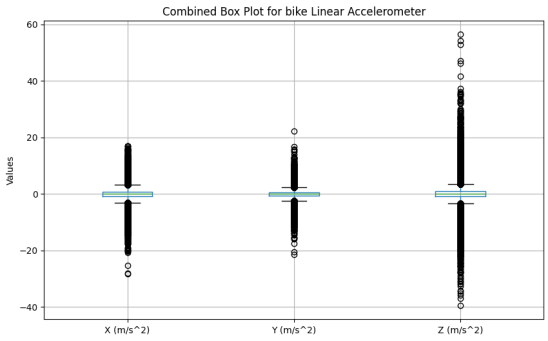


Fig. 6: Combined Box Plot for all bike Linear Accelerometer

Linear Accelerometer Data Analysis: The linear accelerometer data, as depicted in Figure 5, indicates that the bike's movement generally stays within a narrow range for most of the time. However, there are instances of high acceleration, particularly in the Z axis, as shown in Figure 6.

These instances could correspond to rapid acceleration or deceleration events, likely related to dynamic changes in the bike's motion.

2.3 Conclusion

The exploratory data analysis of the bike sensor dataset provides valuable insights into the data distribution and sensor performance. Each sensor type contributes unique and significant information about the bike's dynamics and environment. These insights are crucial for guiding subsequent analysis steps, including data cleaning, feature engineering, and modelling. Similarly, comprehensive analyses have been performed on additional datasets related to activities such as running, walking, car travel, and train journeys. These investigations adhere to the same rigorous analytical standards demonstrated here. Due to space constraints, detailed discussions on these additional datasets are not included in this report. However, they have been thoroughly examined to ensure consistency and comprehensiveness across all analyzed activities. Our final data collection contains six datasets for each activity, with each dataset containing approximately 10 minutes of recorded sensory data. By maintaining consistent data proportions across all activities, we ensure a balanced representation of our collection of data. Therefore, we minimize class imbalance, which is crucial for training robust and unbiased models.

2.4 Additional Time Series Plots for Quick Consideration

Below are the time series plots for activities such as car, run, walk and train. These plots provide a quick visual comparison across different activities.

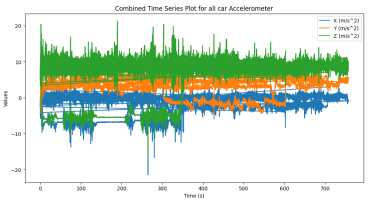


Fig. 7: Car Accelerometer

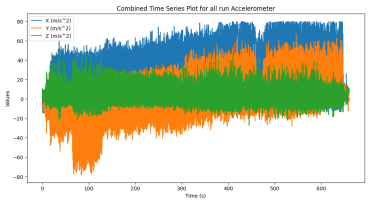


Fig. 8: Run Accelerometer

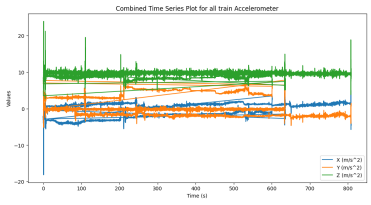


Fig. 9: Train Accelerometer

Fig. 10: Walk Accelerometer

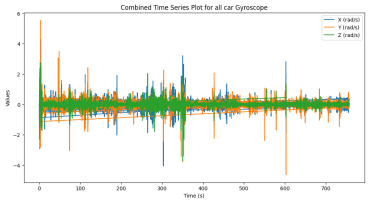


Fig. 11: Car Gyroscope

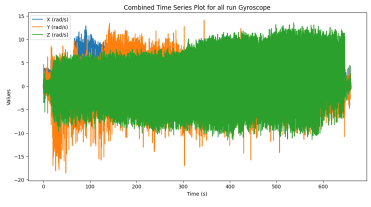


Fig. 12: Run Gyroscope

Fig. 13: Walk Gyroscope

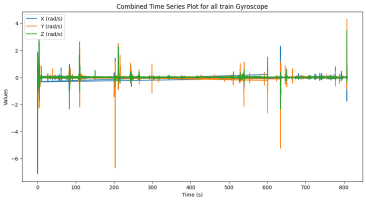


Fig. 14: Train Gyroscope

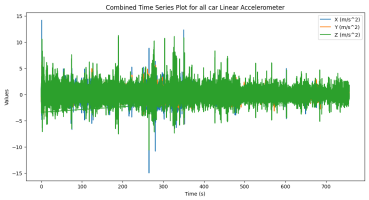


Fig. 15: Car Linear Accelerometer

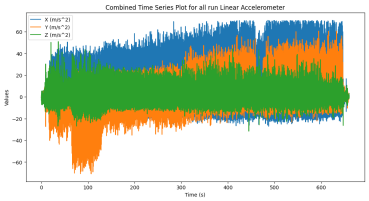


Fig. 16: Run Linear Accelerometer

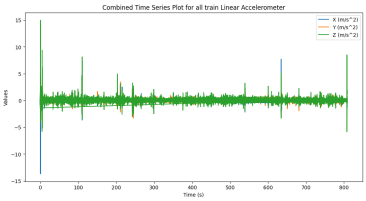


Fig. 17: Train Linear Accelerometer

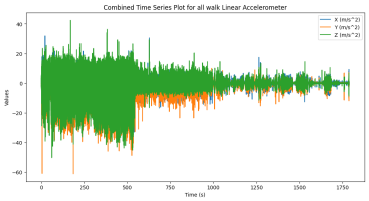


Fig. 18: Walk Linear Accelerometer

3 Data Cleaning

After initial exploration, we applied a series of data-cleaning techniques to each dataset in our data collection. The objective was to clean and smooth the data to ensure its reliability for subsequent analyses. Each dataset in our collection is stored as a pandas DataFrame, where the columns are all of the metrics collected by each sensor (e.g. x, y, z directions for acceleration). The rows represent measurements collected at timestamp t for each of the columns.

3.1 Preliminary Analysis

Initial data loading and analysis were conducted to identify any missing values and to understand the statistical properties of the data. We confirmed that most of the datasets were complete, with little to no missing entries found throughout all of the sensor records. However, during this analysis, we concluded that the datasets specific to the 'train' activity contained long, and sometimes complete series of missing values for the velocity, longitude, and direction metrics collected by the location sensor. Replacing these values is not trivial, as imputing a long series of absent data might introduce bias to all of the 'train' datasets. Therefore, we decided to drop these location metrics throughout all of the datasets in our collection, ensuring that the integrity and consistency of our analysis would not be compromised by the potential biases introduced through imputation.

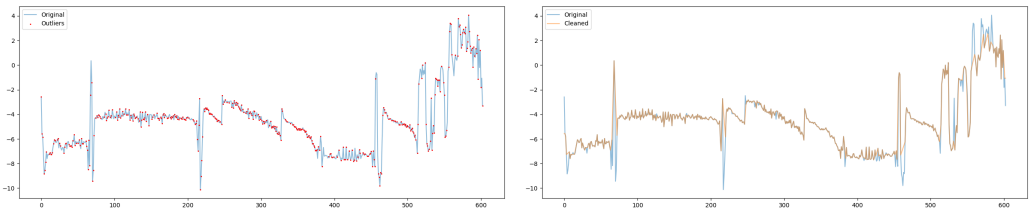
3.2 Outlier Detection

We detect outliers using a distance-based approach, where we consider the distance between the values for each of the individual sensor metrics. By using a distance-based approach, we can disregard

making any normality assumptions about our extensive dataset, contrary to other distribution-based methods. We use the local outlier factor (LOF) algorithm, introduced by Breunig et al[1]. LOF calculates the k-nearest neighbors for each data point, denoting the degree of isolation of the data point compared to its neighbors. It then derives a local reachability density (LDR) by first calculating the reachability distances, which are the maximum distances between points and their respective k-nearest neighbors. The LDR is then derived by taking the inverse of the average reachability distances, which gives us a local density estimate for each data point with respect to their k-nearest neighbors. We detect outliers by comparing the local density estimates for each data point and remove the outlying values by replacing them with NumPy NaN values. In our experiments, we used a k-value and contamination value of 20 and 0.1 respectively.

3.3 Missing Value Imputing

After the outlier removal, we end up with a total of 198.785 missing values in our data collection. We utilize a univariate imputer, which uses the mean statistic to replace missing values across all of the sensor metric columns. Figure 19 shows a complete overview of our data-cleaning process for the first dataset in our collection, by (a) visualizing the LOF outlier detection from the previous section and (b) showing the dataset after outlier removal and mean imputation. As this process is identical to each sensor metric in each dataset in our collection, Figure 19 only shows this process for the accelerometer X (m/s^2) metric on a 1-second granularity.



(a) Outlier Detection (b) Removal and Imputation

Fig. 19: Our Data cleaning process, where (a) shows the outliers detected using LOF and (b) shows the dataset after removal of outliers and mean interpolation

4 Feature Engineering

4.1 Feature Extraction

We extract a variety of statistical and frequency features, using data from the accelerometers, linear accelerometers, gyroscopes, and location sensors. The raw data has 14 numerical columns, and a total of 98 columns of features are obtained after we extract the features. We calculate for each of the sensor columns the following data over predefined time windows:

- **Minimum Value**: The minimum value of each column within the time window.
- **Maximum Value**: The maximum value of each column within the time window.

- **Mean Value**: The average value of each column within the time window.
- **Standard Deviation**: The standard deviation of each sensor within the time window, showing the spread of the data.
- **Slope**: The standard deviation of each sensor within the time window.
- **Fourier Transform**: Perform the Fast Fourier Transform to each sensor data, generating the frequency feature including maximum and mean frequency components.

We initially considered adding patterns with sufficient support. However, the way we collect the data does not suit this pattern-generation method, as we record data strictly according to each type of activity. Each dataset corresponds to a single activity, and the data does not capture changes between different activities. Therefore, we did not add more features.

In addition, we used a sliding window technique with overlap to control the data's similarity and enhance the features' robustness. By setting the overlap rate, we ensure that the model does not take similar data points into consideration repeatedly, which helps prevent overfitting. Without overlap, the model might become too specialized in recognizing specific patterns. We closely follow the recommendations described in [2], by setting the overlap to .5.

4.2 Feature Processing

We perform clustering to gain a better overview of our data structure after generating the features. The clustering algorithms we chose are:

- **K-means Clustering**: It portions data into K clusters, where each data point belongs to the cluster with the nearest mean.
- **Hierarchical Clustering**: It builds a hierarchy of clusters through either agglomerative (bottom-up) or divisive (top-down) methods.

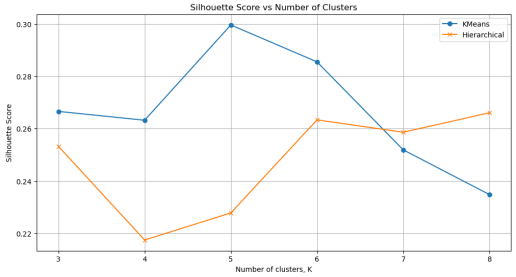
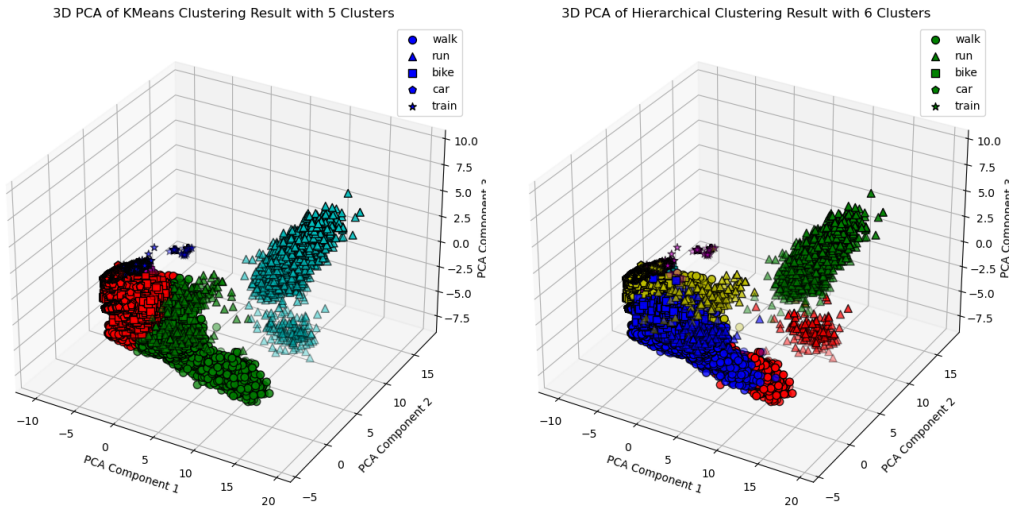


Fig. 20: The Performance of Clustering with Different k Values

We use the Silhouette Score to evaluate the quality of the clustering results. Figure 20 shows the performance of the clustering methods as the number of clusters varies. K-means performs best with 5 clusters, while hierarchical clustering performs best with 6 clusters. Overall, K-means demonstrates better performance.

To better visualize the results, we apply PCA to reduce the features to three dimensions. Figure 21 shows the results of two clustering methods. Specifically, Table 2 presents the outcomes of



(a) K-means Clustering (b) Hierarchical Clustering

Fig. 21: Different Clustering Methods with the Best k

the K-means clustering method. Cluster 1 is highly specialized in identifying the "train" activity and also Cluster 4 and Cluster 5 are associated with "run" and "bike" respectively. These are the clusters with clear and distinct categorization. For Cluster 2 and Cluster 3, there is considerable overlap within "walk", "bike", "run" and "car". Cluster 2 is predominantly associated with "bike" and "car" activities while Cluster 3 is mainly connected with "walk". This phenomenon is understandable because these activities sometimes have very similar speeds.

label	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
train	5547	-	-	-	-
walk	-	5466	1798	-	-
bike	-	1361	5700	-	1275
run	-	222	25	2299	-
car	7	-	6995	-	-

Table 2: Results for K-means Clustering with k=5

5 Algorithm & Model

Our dataset collection comprises sensory data collected from multiple users under various circumstances. Consequently, our dataset reflects diverse conditions and scenarios. Therefore, we consider two different methods for splitting the dataset collection into training, development, and test sets. By using these two methods, we aim to explore the effectiveness and reliability of our models in different data partitioning scenarios.

Measurement Level Split The first method splits the data on a measurement level. The whole dataset is shuffled and divided into training, testing, and development sets. In this method, we set the training set size to 75%, the test set size to 15%, and the development set size to 10%.

Activity Level Split The second method splits the data based on activity level. In this approach, we consider each dataset as an activity record. For each activity, we select one dataset as the test and development sets, while the rest are used for training. The selected test and development data are then split evenly between the test and development sets. Rather than predicting labels for each row of records, we consider a single dataset as a movement to predict.
In the following sections, we will provide detailed descriptions of each model's architecture and hyperparameter tuning efforts.

5.1 Decision Tree

A decision tree is a non-parametric supervised learning method which goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. For our decision tree model, we used the CART (Classification and Regression Trees) algorithm. This algorithm works by splitting the dataset into subsets based on the attribute that results in the largest information gain (or Gini impurity reduction for classification tasks). The process is recursive and continues until all the data points are perfectly classified or another stopping criterion is met, such as a maximum tree depth.

Hyperparameter Tuning We performed hyperparameter tuning using grid search combined with K-Fold cross-validation (k=5) to find the optimal parameters for our decision tree model. The primary hyperparameters tuned include:

- `max_depth`: None, 5, 10, 20, 50
- `min_samples_split`: 0.5, 4, 8, 16
- `min_samples_leaf`: 1, 2, 4, 8

After full hyperparameter tuning, we obtain the following optimized parameter values: **'max_depth': 20; 'min_samples_leaf': 1; 'min_samples_split': 4**

5.2 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm that can be used for both classification and regression. The KNN algorithm classifies a new data point based on the majority class of its k nearest neighbors or predicts a value based on the average of its k nearest neighbors for regression. The KNN algorithm works by calculating the distance between the test point and all training points using a distance metric, such as Euclidean distance. The k nearest neighbors are then selected, and their labels are used to determine the prediction for the test point.

Hyperparameter Tuning Hyperparameter tuning for the KNN model involved selecting the optimal number of leaf sizes, weight functions, and the appropriate distance metric. We used grid search combined with K-Fold cross-validation (k=5) to find the best value for our parameters. For the KNN model, we optimized across the following parameter values:

1. `leaf_size:` 10, 20, 30, 40
2. `weights:` 'uniform', 'distance'
3. `metric:` 'euclidean', 'manhattan'

For the KNN model, we obtain the following parameter values: **'leaf_size': 10; 'metric': 'manhattan'; 'weights': 'distance'**

5.3 LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture capable of learning long-term dependencies in sequential data. The LSTM aims to solve the vanishing gradient problem by utilizing memory cells to store information. These cells contain so-called input, output, and forget gates, and control the information flow within the memory cell. For our research, we implement a multi-layer LSTM classification model in PyTorch. Our model includes a single dense linear layer added to the hidden outputs of the LSTM layers. This linear layer serves as a mapping from the hidden dimension to the final output space (in our case 5), transforming the hidden states to output logits for each class. Models are trained using the Cross-Entropy Loss, which compares the true label distribution with the predicted label distribution. This predicted label distribution is obtained by applying a Soft-Max function on the output logits and is denoted as $\hat{y}$ in the following loss function:

$$Loss = -\sum_{i=1}^{c} y_i \, log(\hat{y}_i) \tag{1}$$

Where y_i is the ground truth label distribution, model training is optimized using the Adam[3] optimizer.

Hyperparameter Tuning We optimize our parameters using an exhaustive search over all combinations of the parameter values. We utilize K-Fold validation with a k-value of 5 to ensure an unbiased selection of optimal parameters. For our LSTM implementation, we optimized over the following parameter values:

1. `hidden_size:` 50, 100, 200, 400
2. `drop_out:` 0.3, 0.5, 0.8
3. `learning_rate:` 1e-1, 1e-2, 1e-3

After optimization, we end up with the following parameters: **'hidden_size': 400, 'drop_out': 0.3, 'lr': 0.001** All models were trained with 3 hidden layers, thus we ended up with our final model containing 3 hidden layers, each containing 400 hidden units. We use a low drop-out rate of 0.3 and a slow learning rate of 0.001.

6 Evaluation

6.1 Metrics

We use four statistical metrics to evaluate our models. They are:

- **Accuracy**: This metric measures the overall correctness of the model considering all correctly predicted instances.
- **Precision**: This metric measures the accuracy of the positive predictions.
- **Recall**: This metric measures the ability of the model to correctly identify all positive instances.
- **F1 score**: This metric is the harmonic mean of precision and recall, providing a balance between the two.

6.2 Experiment Setting

To evaluate the performance of those two models under various granularities, window sizes, and sensor combinations, we design a series of experiments. First, the data is processed according to different granularities (i.e., 100ms, 500ms, 1s). Then we use different window sizes of 5, 10, and 50 to generate the features. For each granularity and window size combination, the following steps are performed:

- **Data Preparation**: Cleaninig of the data and extraction of statistic features.
- **Model Training**: Each model is trained for ten epochs, and repeated three times.
- **Evaluation and Metrics Recording**: The metrics are recorded for each run and the average metrics of each model during multiple training are also computed and saved.
- **Best Model Recording**: The model with the highest accuracy for each granularity and window size combination is selected as the best model, which is saved for further analysis.

We perform our experiments by repeating these steps for each dataset split (measurement/activity). Models are evaluated on the development sets after each epoch, and the best-performing model on the development sets (based on accuracy) is then evaluated on the test set.

6.3 Results

In this section, we present the detailed results for both the measurement level split and activity level split using all the models. Table 3 and Table 4 show the experiment results. For measurement level split, the LSTM model performs the best most of the time. The KNN model also shows excellent performance, especially at 1s granularity with a window size of 5. The Decision Tree model also performs well, just slightly lower compared to the other ones.

Table 3: Measurement Level Split Results

Δt	Window Size	Model	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
100ms	5	DT	99.32	99.44	99.42	99.43
	10	DT	99.33	99.47	99.44	99.46
	50	DT	99.46	99.56	99.53	99.55
	5	KNN	99.69	99.74	99.61	99.68
	10	KNN	99.50	99.65	99.46	99.55
	50	**KNN**	**99.87**	**99.91**	**99.88**	**99.89**
	5	**LSTM**	**99.68**	**99.72**	**99.71**	**99.73**
	10	**LSTM**	**99.54**	**99.60**	**99.59**	**99.59**
	50	LSTM	99.59	99.65	99.58	99.61
500ms	5	DT	98.90	98.98	99.08	99.03
	10	DT	99.11	99.25	99.35	99.25
	50	DT	98.53	98.70	98.74	98.72
	5	KNN	98.89	99.11	98.77	98.93
	10	KNN	99.01	98.94	98.98	98.96
	50	KNN	98.43	98.60	98.65	98.61
	5	**LSTM**	**99.57**	**99.61**	**99.61**	**99.61**
	10	**LSTM**	**99.55**	**99.57**	**99.58**	**99.58**
	50	**LSTM**	**99.54**	**99.66**	**99.57**	**99.58**
1s	5	DT	99.26	99.35	99.37	99.36
	10	DT	98.52	98.56	98.77	98.66
	50	DT	98.73	99.03	98.99	99.00
	5	**KNN**	**100.00**	**100.00**	**100.00**	**100.00**
	10	KNN	98.42	98.61	98.79	98.68
	50	KNN	89.92	88.55	88.85	88.62
	5	LSTM	99.57	99.62	99.59	99.61
	10	**LSTM**	**99.58**	**99.61**	**99.59**	**99.60**
	50	**LSTM**	**99.60**	**99.65**	**99.63**	**99.64**

For activity level split, the LSTM model maintains the strong performance of the best in general. The Decision Tree model generally performs better than the KNN model, especially at higher granularities.

Table 4: Activity Level Split Results

Δt	Window Size	Model	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
100ms	5	DT	52.36	60.36	42.05	42.85
	10	**DT**	**71.22**	**67.77**	**54.68**	**55.87**
	50	**DT**	**89.11**	**71.39**	**67.72**	**69.11**
	5	KNN	64.59	51.29	40.42	42.18
	10	KNN	69.04	52.65	42.82	44.61
	50	KNN	89.66	73.74	67.68	70.34
	5	**LSTM**	**81.14**	**54.52**	**49.58**	**50.85**
	10	LSTM	77.46	54.51	47.82	49.92
	50	LSTM	78.79	55.57	48.66	50.77
500ms	5	DT	79.19	67.34	57.51	60.97
	10	DT	53.12	60.51	42.12	30.30
	50	**DT**	**77.56**	**70.38**	**59.38**	**59.38**
	5	KNN	72.34	57.69	45.56	49.50
	10	KNN	67.88	58.35	43.34	47.67
	50	KNN	51.43	58.50	33.80	39.62
	5	**LSTM**	**81.53**	**55.67**	**50.02**	**51.80**
	10	**LSTM**	**84.39**	**57.40**	**51.75**	**53.78**
	50	LSTM	79.27	55.95	49.04	51.54
1s	5	DT	60.23	60.01	46.19	41.18
	10	**DT**	**75.84**	**65.49**	**57.24**	**53.99**
	50	DT	57.84	41.80	36.14	34.37
	5	KNN	63.62	59.24	41.49	44.94
	10	KNN	65.17	60.00	42.29	46.16
	50	KNN	47.06	58.00	31.40	36.54
	5	**LSTM**	**83.02**	**56.71**	**50.98**	**52.63**
	10	LSTM	80.02	55.39	49.20	51.17
	50	**LSTM**	**82.32**	**54.13**	**50.14**	**50.98**

6.4 Abbreviation Study

In the following abbreviation study, we will examine the performance of the models when fewer features (sensors) are available. To do so, we train models on a fixed granularity (100ms) and fixed window size (10) while varying the sensors used. Figure 22 shows both the model's performance on the measurement runs and activity runs. From these plots, we observe the following:

DT Model: The accuracy is highest when all sensors are used (combination 1) and decreases as fewer sensors are included. The drop is significant when reducing to just a few sensors and DT performs the worst when only considering the gyroscope data. We can observe a slight edge in performance in the measurement runs for DT, compared to the activity runs.

KNN Model: Shows a similar trend to the DT model, maintaining higher accuracy with more sensors. However, it shows a more gradual decline in performance compared to the DT model. We also observe a great decrease in performance when using the location sensor between the measure-

ment and activity runs.
LSTM Model: Exhibits robustness with fewer sensors compared to DT and KNN, maintaining relatively high accuracy even with reduced sensor combinations. However, we observe a significant dent in performance when only utilizing the location sensor, compared to the DT and KNN models. LSTM seems to perform slightly better on the activity dataset.

Overall, we can conclude that the models' performance is significantly impacted by the number of sensors used. We observe that the acceleration and location sensors play a vital role in activity classification, while the location and gyroscope sensors may have a less impactful role. However, we can conclude that in general, the model's performance is still reasonable, even when sensors are dropped.

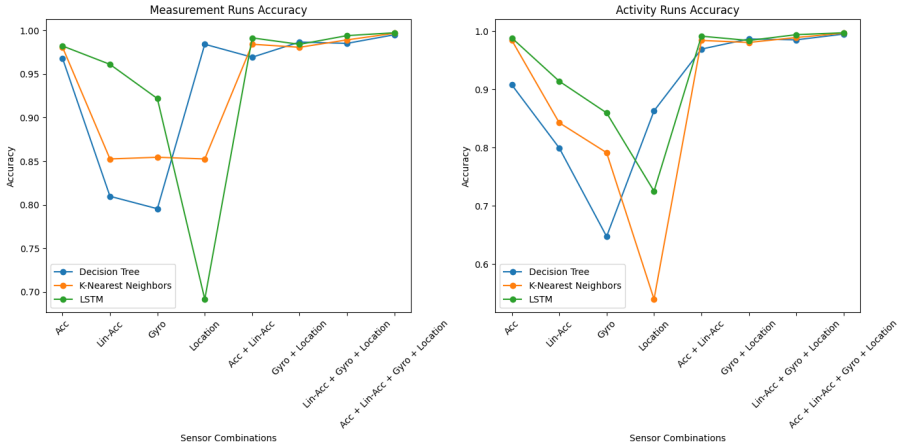


Fig. 22: Abbreviation study on both dataset levels

7 Discussion

In this project, we use different models including Decision Tree, KNN and LSTM to predict the activities. Our results indicate that the LSTM model consistently outperforms the other models, demonstrating the strong capability of deep learning. This may also be related to LSTM's ability to capture long-term dependencies in sequential data.

While evaluating the models at the measurement level, we observed exceptionally high results. We believe this phenomenon could be due to the nature of our data collection process. Since the data was recorded in segments without activity transitions, there is a high degree of continuity and similarity within the dataset. Consequently, even though the data was randomly split for training and testing, the segments used for testing may be very similar to those seen during training, leading to inflated performance metrics. Future work should consider more diverse datasets with varying activity transitions to better evaluate the generalization capabilities of the models.

References

1. Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 93–104, New York, NY, USA, 2000. Association for Computing Machinery.
2. Mark Hoogendoorn and Burkhardt Funk. *Machine learning for the quantified self: On the art of learning from Sensory Data*. Springer, 2018.
3. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
4. Sebastian Staacks. Your smartphone is a mobile lab. — phyphox.org. https://phyphox.org/. [Accessed 23-06-2024].