Machine Learning for the Quantified Self Assignment *Group 24*

Pengju Ma[2764123], Zhuofan Mei[2757904], and Lin Tian[]

Vrije Universiteit Amsterdam
{p.ma2,lin.t,z.mei}@student.vu.nl

Abstract. In this work, we collected the sensor data ourselves and described a refinement pipeline consisting of data pre-processing, feature engineering and selection. We then evaluated machine learning and deep learning methods and compared them based on prediction results.

Keywords: Machine Learning · Sensory Data · Multi-Classification Tasks · Data Collection · Feature Engineering

1 Introduction

In this assignment, we work on implementing a machine learning multi-classification task based on sensor data. We collect data from the real world by using a mobile application capable of collecting sensor data. Our goal is to use this data to train an accurate and efficient machine learning model that is able to accurately classify different classes

Research Question

Research Question: *"Can the type of activity of a user be accurately predicted by analyzing multifaceted sensor data during exercise?"*

We will use the mobile application Phyphox to collect sensor data during various different types of activities performed by the user (e.g. walking, running, sitting, etc.). These activity types will be used as our target variables, which is what we will be predicting.

We will collect accelerometer, gyroscope, magnetometer and position data during each activity. This data will provide information about the user's movement, orientation, position, and space. By analyzing this data, we hope to build an accurate machine learning model that can predict the type of activity the user is in based on the sensor data.

To answer this research question, we will collect data from multiple users to ensure the diversity and generalization of the data. We will analyze and process this data, performing cleaning and pre-processing to ensure the quality and reliability of the data. We will then use this processed data to train and evaluate machine learning models and use the models to make classification predictions on new sensor data.

2 Dataset Description

2.1 Dataset Statistic

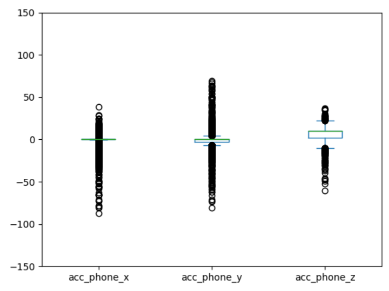
As mentioned in the previous section, this dataset is collected by Phyphox at a sampling rate of 50 Hz. We performed a total of four different exercises: `sit`, `walk`, `run`, and `cycling`, as well as the corresponding attributes: `accelerometer`, `gyroscope`, `magnetometer`, and location (including `latitude`, `longitude`, and `speed`). We merged the accelerometer, gyroscope, magnetometer and position data based on a common identifier, i.e., the timestamp. And the dataframe aggregation with Δt=250ms and 1s respectively. We then calculated the statistical metrics corresponding to each attribute and the missing values to complete the initial exploration, as shown in Table 2.1. The location-related attributes contain a large percentage of missing values, so we can consider not applying these attributes in further steps.

Attributes	**Mean** (250ms/1s)	**Std**	**Count**	**Missing**
`acc_x`	-0.028/-0.010	5.066/5.345	15641/19542	0%/0%
`acc_y`	-1.708/-1.727	6.309/6.639	15641/19542	0%/0%
`acc_z`	6.664/6.656	5.475/5.566	15641/19542	0%/0%
`gyr_x`	-0.013/-0.018	1.030/1.007	15639/6648	0.013%/65.981%
`gyr_y`	-0.011/0.024	1.694/1.866	15639/6648	0.013%/65.981%
`gyr_z`	0.000/ -0.002	0.982/0.963	15639/6648	0.013%/65.981%
`mag_x`	-8.644/-9.988	18.456/14.740	15639/6648	0.013%/65.981%
`mag_y`	7.659/8.219	19.765/19.129	15639/6648	0.013%/65.981%
`mag_z`	-28.948/-29.648	15.060/14.716	15639/6648	0.013%/65.981%
`latitude`	52.334/52.334	0.001/0.001	1091/377	93.024%/98.071%
`longitude`	4.892/4.891	0.023/0.023	1091/377	93.024%/98.071%
`speed`	-0.012/0.009	1.068/1.092	1091/377	93.024%/98.071%

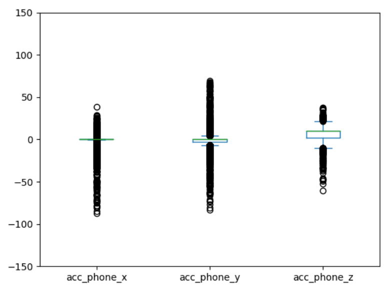
Table 1. The attributes

2.2 Exploratory Data Analysis

Figure 1(a) and 1(b), show the distribution of accelerometer data, we can see that the data has a large range of variation and a wide distribution of data points, and there are also some extreme values. This indicates that there is a large oscillation in the acceleration data and that the frequency of the object's oscillation is highly variable or unstable. And the signal graphs over time are shown in Figure 2(a) and 2(b), between 19:00 and 20:00, the indicators (`accelerometer`, `gyroscope`, `magnetometer`, and location) fluctuate more since we were doing the sports such as running and cycling.

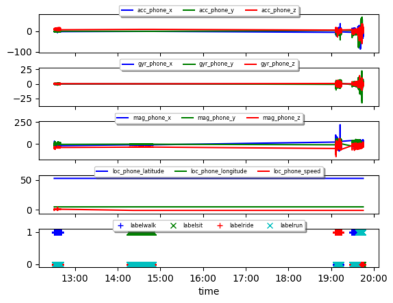


(a) Accelerometer data with Δt=250ms

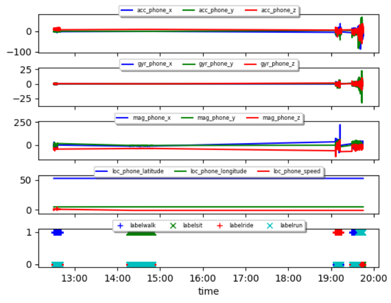


(b) Accelerometer data with Δt=1s

Fig. 1.



(a) Measured signals Δt=250ms



(b) Measured signals Δt=1s

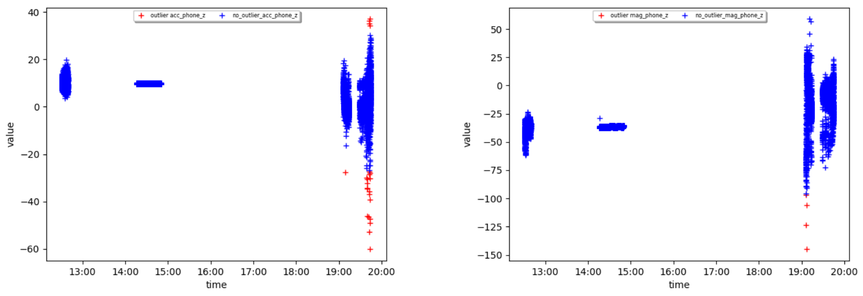
Fig. 2.

3 Noise Handling

3.1 Outliers Removing

For this section, we used Distance-based outlier detection [1], for the reason that we believe that this method can effectively detect the encouraging points and unusually sparse density areas in the data, and in addition, we found that our data distribution is relatively consistent, i.e. not enough overly sparse data clusters, so the Simple distance based method is sufficient to meet the requirements [2]. We also observed and compared different parameter combinations, and finally found that the combination of `dmin=0.2` and `fmin=0.99` is more satisfactory.

Take Figure 3(a) and Figure 3(b) as examples, we can find that outliers are mainly concentrated in the interval from 19:00 to 20:00 where the data density distribution is large. This part of the data was collected from the time interval of the user's physical activity. Therefore, the data collected by the sensor may



(a) Outliers in `acc_phone_z` (b) Outliers in `mag_phone_z`

Fig. 3.

generate some incoherent outliers due to drastic changes, and from the figure, we can also see that these outliers are effectively removed.

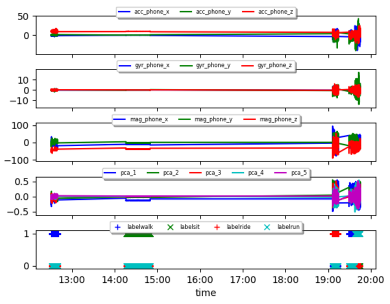
3.2 Missing Values Imputation

For the three location-related attributes, we did not do anything with them in this section because their data distribution is too sparse and therefore the value inference is not meaningful for these attributes. As for the `acc`, `gyr` and `mag` classes, by looking at the dataset we found that only two rows of the 15,000+ total data have missing values for all attributes in these three classes at the same time. Therefore, the method used to fill these two rows has little impact on the final model training, and we simply used the mean fill method.

Subsequently, low-pass filters were applied to columns containing periodic data, specifically accelerometer and gyroscope measurements, to remove frequencies above 1.5 Hz. Removing high-frequency noise and isolating low-frequency motion can improve the performance of machine learning algorithms trained on the data. The low-pass filtered data may better represent the actual motion. Then, principal component analysis was also performed on the filtered data to reduce the dimensionality of the data, and the optimal principal component score was considered to be 5. Figure 4(a) shows the results after performing PCA on the dataset, extracting a specified number of principal components, transforming the original data and adding the principal components as new columns to the dataset.

4 Feature Engineering

In this section, we used our raw sensor data to generate some useful features to discover the potential characteristics of the original data. Our method begins with numerical abstraction, which involves generating aggregate statistics such


(a)

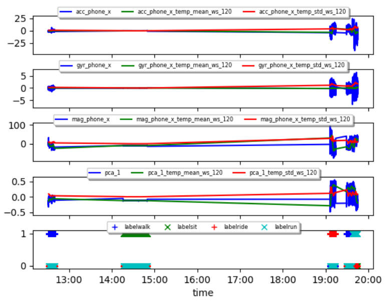

(b)

Fig. 4.

as average and standard deviation. These window-based features allow us to capture changes in sensor readings across certain time periods, indicating changes in the user's activities throughout that period.

Figure 4(b) shows a comparison of the data range changes for some typical raw features and associated aggregated features. There are some new properties that are applied based on the aggregation of specific window sizes, calculating mean and standard deviation at 120Hz sampling frequency. It can be seen that the filter in the original data is significantly reduced after aggregation, so it is possible to further explore the common characteristics in the data.

The second step is category abstraction, which focuses on label data. Our goal here is to convert our categorical labels into a more abstract representation, which will allow us to focus on broader behavioral patterns rather than specific cases. This approach is predicated on a list of binary columns representing the various categories. Several limitations are introduced throughout the abstraction process, such as minimum support, a window size between succeeding patterns, and a maximum size for the pattern set. These limits serve in balancing the complexity and interpretability of the created patterns, as well as the process's computational efficiency.

In addition, we also considered frequency domain features. Because human actions, like walking or running, inherently involve repeating patterns, it is useful to convert these time-domain data into frequency-domain signals to capture these periodic patterns. This is accomplished by performing a Fast Fourier Transform (FFT) [3] on the windowed sensor data. We can infer additional informative features from the frequency domain representation, which offers a measure of the frequency distribution's homogeneity.

5 Clustering

Clustering is the next phase in our data analysis process after feature engineering. This is an unsupervised machine learning technique in which we group together comparable data items. The purpose of this stage is to find intrinsic groupings in the data that may indicate different behavior states. It can reduce noise and outliers and highlight overall trends. For different movements, there may be distinct patterns or trends in the acceleration data over time. Clustering can make these trends more apparent by smoothing out short-term fluctuations. In addition, raw acceleration data can be very large, especially at high sampling frequencies. Aggregation can greatly reduce the size of the data while retaining important information, making it easier to store, analyse and visualise.

We used the k-means clustering algorithm for clustering. This technique accepts as input the dataset and a collection of features to be used for clustering, as well as parameters indicating the number of clusters, the initial centroid selection approach, and the maximum number of algorithm iterations. In our case, we chose to perform the clustering based on the accelerometer readings in the x, y, and z directions. These were selected because they capture essential movement characteristics that could differentiate between behaviors such as walking and running.

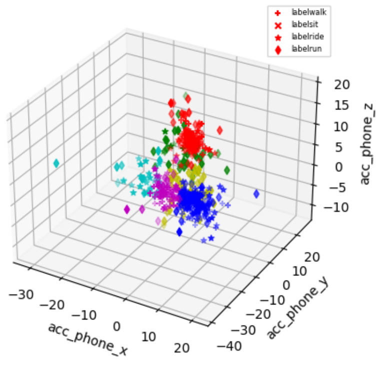


Fig. 5.

To evaluate the quality of our clusters, we employed 3D cluster figures in our data after clustering. We can see from Figure 5 that the sedentary data were almost all concentrated at the zero point of the coordinate, while the running and cycling-related data showed a wide range of dispersion around the zero point on the coordinate axis, with the running data varying more significantly. This is most likely due to the uneven speed of the subjects. Therefore the degree of data dispersion may be used as an important indicator for subsequent model training predictions.

6 Classical Machine Learning

In our approach, we began by preprocessing the dataset through feature engineering techniques.

6.1 Dataset Splitting

To ensure an appropriate train/test setup, we split the dataset for our classification problem. We paid attention to the class columns (class-labels) and maintained the class distribution during the split, avoiding any potential bias.

6.2 Feature Selection

For feature selection, we opted for the forward selection method, which sequentially adds features based on their performance. We set a predefined number of features that we aimed to select for our models.

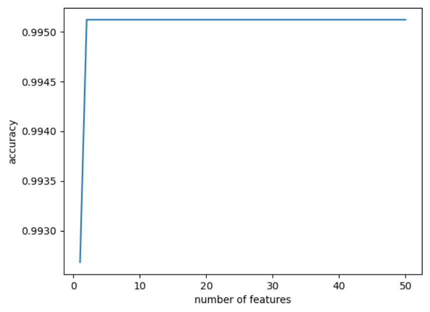


Fig. 6.

6.3 Machine Learning Algorithm

We considered various machine learning models for our classification task, including feedforward neural networks, SVM, k-nearest neighbors, decision trees, and naive Bayes. The selection of these models was based on their relevance and suitability for our specific classification problem.

Feedforward neural networks are powerful models that can learn complex features and patterns through the connections of multiple layers of neurons and nonlinear activation functions. They perform well with large datasets and high-dimensional features, exhibiting strong expressive capabilities.

SVM is a binary classification model based on maximizing the margin by mapping data to a high-dimensional space and constructing an optimal hyperplane. It performs well for both linearly separable and nonlinearly separable data, with good robustness to small samples and noise.

The k-nearest neighbors algorithm is an instance-based learning approach that classifies based on the proximity of samples. It is simple and intuitive, adapting well to nonlinear and complex decision boundaries. However, it can be computationally expensive for high-dimensional and large-scale datasets.

Decision trees make decisions through a series of branching nodes, using feature selection and conditional branching to construct a tree structure. They are easy to understand and interpret and exhibit good robustness to handling missing values and outliers. However, they can be prone to overfitting and produce unstable results.

Naive Bayes is a probabilistic model based on the Bayes theorem and the assumption of feature independence. It classifies by calculating posterior probabilities. Naive Bayes models have good computational efficiency and interpretability, performing well in tasks such as text classification and spam filtering. However, they are sensitive to feature dependencies.

6.4 Hyperparameter Tuning

To optimize the models' performance, we focused on hyperparameter tuning. Specifically, we experimented with different regularization parameters (`reg-parameters`). We systematically tested these parameters using techniques like grid search or cross-validation to identify the values that yielded the best results. We utilized an adaptive learning rate for our machine learning models. Adaptive learning rate algorithms dynamically adjust the learning rate during the training process based on the behavior of the model. This helps to optimize the learning process and improve convergence.

6.5 Re-selected Feature

To score the selected features, we utilized a random forest model. This allowed us to assess the importance of each feature for classification. The scoring process was based on the decision tree's splitting criteria, which inherently provide insights into feature importance, the results are shown in Table 2.

6.6 Conclusion

After obtaining the feature scores, we narrowed down our selection to the top 10 features. We used a specific criterion, such as a threshold score or a fixed number of top features, to make this final selection. Then, we re-run the model, and the accuracy of the training process finally reached nearly 94%. The outcome is a matrix which includes the accuracy of the prediction on the test set. In conclusion, our approach involved a thorough train/test setup, feature selection

Table 2. Feature Importance - Random Forest

Feature	**Importance**
gyr_phone_y_temp_std_ws_120	0.22585988953553662
pca_5_temp_std_ws_120	0.19978973369156883
acc_phone_x_temp_std_ws_120	0.10936901973854646
mag_phone_y	0.06614264266165276
pca_2_temp_std_ws_120	0.06318816959430572
acc_phone_y_temp_mean_ws_120	0.044795844016246526
loc_phone_speed_temp_mean_ws_120	0.04155196733623881
mag_phone_z_max_freq	0.04149222171444176
loc_phone_longitude_temp_mean_ws_120	0.035398105986918324
pca_5_temp_mean_ws_120	0.03126728365248412
acc_phone_x_freq_0.0_Hz_ws_40	0.017770157753141957
acc_phone_z	0.017651471905484818
mag_phone_x_freq_1.1_Hz_ws_40	0.014581158429487168
pca_5	0.007714724543382103
acc_phone_z_freq_weighted	0.0064578658518771835

using forward selection, consideration of various machine learning models, hyperparameter optimization, feature scoring using a decision tree, and retraining of models using the final set of selected features. This iterative process allowed us to apply classical machine learning techniques effectively and achieve optimal performance for the classification problem at hand.

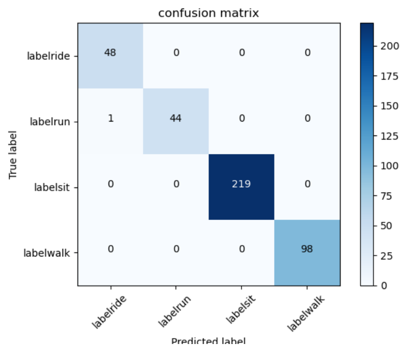


Fig. 7.

7 Deep Learning Model

Then, we selected the same feature of the dataset as mentioned in the previous section, and applied the Long Short-Term Memory (LSTM) Network for the classification tasks.

7.1 Long Short-Term Memory Network

LSTM is a recurrent neural network capable of learning long-term dependencies. Traditional recurrent neural networks struggle to learn long sequences due to the vanishing gradient problem. It overcomes this problem by incorporating a `unit state` - a memory cell that can store data for a significant period of time [4]. As the algorithm is shown below, the unit consists of three gates: a forgetting gate, which decides what information to remove from the unit state; an input gate, which decides what new information to add to the unit state; and an output gate, which decides what part of the unit state to use as output [5]. These gates allow the LSTM to remember or forget information in the unit, depending on what is relevant to the prediction at each step, facilitating efficient learning of dependencies in long sequences.

Algorithm 1 Long Short-Term Memory (LSTM) Algorithm

```
Require: Input sequence $\mathbf{X}$, initial states $\mathbf{h}_0$ and $\mathbf{c}_0$, LSTM parameters $\mathbf{W}$, $\mathbf{U}$, $\mathbf{b}$
Ensure: Output sequence $\mathbf{Y}$
 1: function LSTM($\mathbf{X}$, $\mathbf{h}_0$, $\mathbf{c}_0$, $\mathbf{W}$, $\mathbf{U}$, $\mathbf{b}$)
 2:     Initialize cell states $\mathbf{c}_t$ and hidden states $\mathbf{h}_t$ with $\mathbf{c}_0$ and $\mathbf{h}_0$
 3:     for $t = 1$ to $T$ do                                   ▷ Iterate over the time steps
 4:         Extract input $\mathbf{x}_t$ from $\mathbf{X}$
 5:         Compute input and forget gate activations:
 6:             $\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i)$
 7:             $\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f)$
 8:         Compute cell state update:
 9:             $\mathbf{u}_t = \tanh(\mathbf{W}_u\mathbf{x}_t + \mathbf{U}_u\mathbf{h}_{t-1} + \mathbf{b}_u)$
10:             $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$
11:         Compute output gate activation:
12:             $\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o)$
13:         Compute hidden state update:
14:             $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$
15:         Store $\mathbf{h}_t$ in output sequence $\mathbf{Y}$
16:     return $\mathbf{Y}$
```

The structure of the network is shown in Figure 7.1, there is an LSTM network to classify sequences into 4 classes. The architecture consists of an LSTM layer to process the sequential data, followed by some dense layers for feature extraction and classification. The Dropout layers help to reduce overfitting.

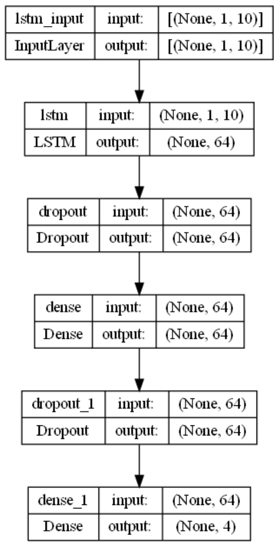

lstm_input | input: | [(None, 1, 10)]
InputLayer | output: | [(None, 1, 10)]
lstm | input: | (None, 1, 10)
LSTM | output: | (None, 64)
dropout | input: | (None, 64)
Dropout | output: | (None, 64)
dense | input: | (None, 64)
Dense | output: | (None, 64)
dropout_1 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_1 | input: | (None, 64)
Dense | output: | (None, 4)

Fig. 8.

7.2 Hyperparameter Settings

The general model structure has been discussed above, and the parameter settings and function selection for each layer are shown in Table 7.2. Dropout helps to reduce the dependence on specific features, the value of 0.2 means that 20% of the input units are randomly set to 0 during each training cycle. Adam [6] is a common optimisation algorithm used in deep learning. It combines the Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) to provide efficient gradient-based optimisation, which adapts the learning rate of each parameter based on the average of past gradients, helping to converge faster and handle sparse gradients. ReLU introduces non-linearity into the model, helping to learn complex patterns by allowing positive values to pass unchanged while setting negative values to zero. And by minimising the categorical cross-entropy loss, the model learns to assign higher probabilities to the correct categories and lower probabilities to the incorrect categories.

7.3 Evaluation and Results

The results of the training process are shown in Table 7.3, we apply the statistical indicators accuracy, recall, and f1 to determine the performance of the model. And the results of the test dataset are shown in Figure 7.3, compared to the machine algorithm we used before, the results of LSTM are slightly worse, which also suggests that we should further optimize the processing of selected features in our future work.

Parameter	**Value**
`learning rate`	0.001
`batch size`	32
`epochs`	50
`dropout`	0.2
`optimizer`	Adam
`activation function`	relu
`loss function`	categorical_crossentropy

Table 3. The hyperparameters

Indicators	**Value**
`accuracy`	93.85%
`recall`	91.45%
`f1`	89.73%

Table 4. Evaluation

8 Conclusion and Justification

In summary, we collected the motion data ourselves, including acceleration, position and other attributes, and performed a complete engineering pipeline including data collection, noise processing, feature engineering and selection. We then applied machine learning and deep learning to the processed data for the classification task and compared the results.

Based on the literature review, we found some outstanding work, which can give us some inspiration for future work. Saleh et al. [7] proposed a new method for categorising driving behaviour based on stacked LSTM recurrent neural networks, formulating the problem of classifying driving behaviour as a time-series classification task based on data from nine different sensors. On the UAH DriveSet, their proposed stacked LSTM model achieves top scores and 91% of the F1 measurements. Besides, to improve recognition performance, Mekruksav et al. [8] proposed a 4-layer CNN-LSTM hybrid LSTM network in addition to a general Human Activity Recognition (HAR) framework for smartphone sensor data. Besides, different combinations of sampling methods, validation protocols (10-fold and LOSO cross-validation) and Bayesian optimisation techniques were used to tune the hyperparameters of each network. In the end, good performance was achieved.

These works give us some inspiration for future work and improvements. We can improve the baseline LSTM model from a simple single-layer LSTM to a stacked LSTM or CNN-LSTM architecture with hyperparameter optimization to achieve higher performance, and we can introduce more datasets to help the training process.

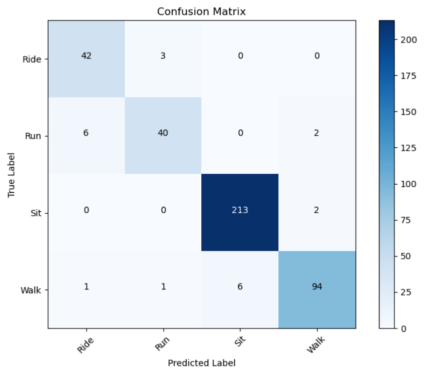


Fig. 9.

References

1. E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: algorithms and applications," *The VLDB Journal*, vol. 8, no. 3, pp. 237–253, 2000.
2. M. Hoogendoorn and B. Funk, "Machine learning for the quantified self," *On the art of learning from sensory data*, 2018.
3. W. T. Cochran, J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch, "What is the fast fourier transform?" *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664–1674, 1967.
4. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
5. Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.
6. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
7. K. Saleh, M. Hossny, and S. Nahavandi, "Driving behavior classification based on sensor data fusion using lstm recurrent neural networks," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.
8. S. Mekruksavanich and A. Jitpattanakul, "Lstm networks using smartphone data for sensor-based human activity recognition in smart homes," *Sensors*, vol. 21, no. 5, p. 1636, 2021.