Machine Learning Classification for Human Activity Recognition

Andy Malinsky, Dheemanth Rajakumar, Greg Moore

Shiley-Marcos School of Engineering, University of San Diego

AAI-501: Introduction to Artificial Intelligence

Professor Andrew Van Benschoten, PhD.

Professor David Friesen, MS

08/12/2024

Machine Learning Classification for Human Activity Recognition

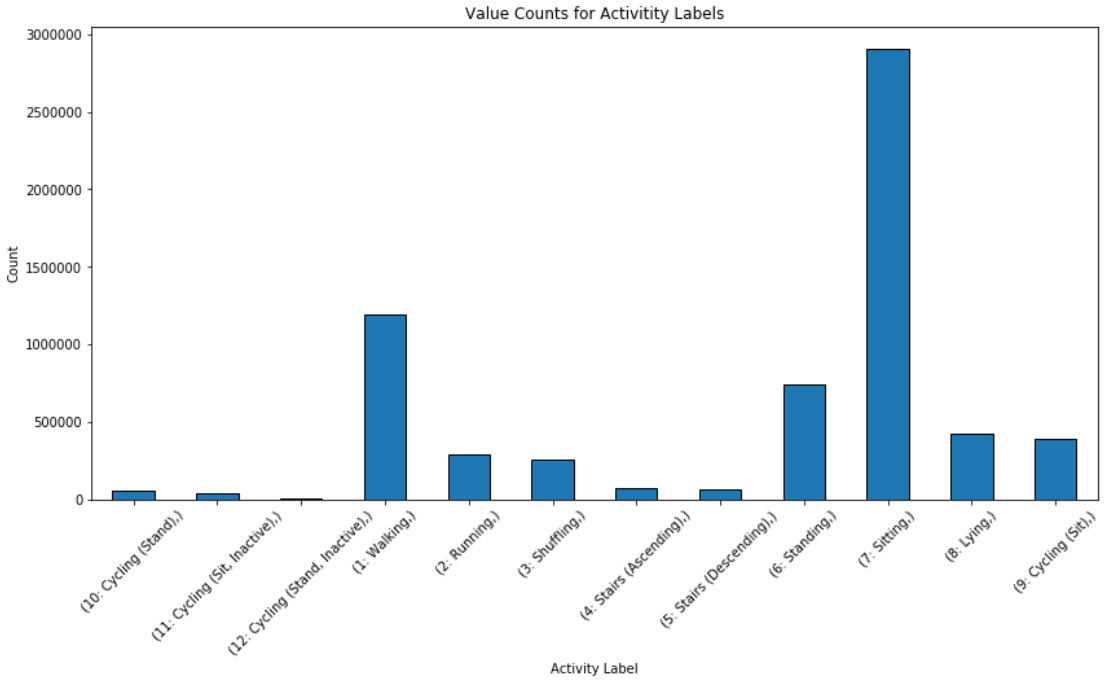
This report reviews and compares three different machine learning models using the Human Activity Recognition Trondheim (HARTH) dataset from UC Irvine (Aleksej Logacjov et al., 2023), which is a culmination of three-axial accelerometer sensor data on 22 test subjects (people). The sensor data was collected from two different sensors, one on the thigh and one on the hip, from people doing normal things during a 2-hour period in their homes. The data has been annotated to assist in the development of machine learning classifier models for processing timestamped HAR sensor data. From the 22 subject files combined, there are 6,461,328 rows of sensor data that the three models developed for this final project were used to train and evaluate the machine learning models. Code for this project is made public on GitHub (Malinsky, Rajakumar, & Moore, 2024). The three models developed to process the HARTH data are a Support Vector Machine (SVM) model, a Convolutional Neural Network (CNN) model, and a Bidirectional Long Short-Term Memory (bi-LSTM) model.

Dataset Exploration

Each of the 22 subject test files in the dataset contains 8 features or different sensor data that is timestamped at 50Hz to 100Hz and labeled. This data was used to train the models, test, and validate the model performance. The features in each dataset file are timestamp: date and time of recorded sample, back_x: acceleration of back sensor in x-direction (down), back_y: acceleration of back sensor in y-direction (left), back_z: acceleration of back sensor in z-direction (forward), thigh_x: acceleration of thigh sensor in x-direction (down), thigh_y: acceleration of thigh sensor in y-direction (right), thigh_z: acceleration of thigh sensor in z-direction (backward), and label: annotated activity code. The annotated activities in the dataset are: 1: Walking, 2: Running, 3: Shuffling, 4: Stairs (Ascending), 5: Stairs (Descending), 6:

Standing, 7: Sitting, 8: Lying, 13: Cycling (sit), 14: Cycling (stand), 130: Cycling (sit, inactive), and 140: Cycling (stand, inactive). Exploratory data analysis in Figure 1 shows that most labels are Sitting, Walking, or Standing. This makes sense as those activities are most common in daily life.

Figure 1
Value counts for activity labels



Project Goal: THE METALYZER

The application we chose to use our model for is an idea for a product named “THE METALYZER” that involves a hip or back based accelerometer sensor that attaches to a person. The METALYZER will record and process sensor data real-time and by implementing our classification model into the sensor device, it will predict what activity the person is doing. When the person moves to another activity, the METALYZER will process the previous activity data using timestamps, MET table, and weight (kg) and will convert the time duration of the activity to calories burned using a Metabolic EquivalenT translation table for the activities. One MET is equal to one minute of the amount of energy (calories) the human body burns when at

rest, so we chose this standardized translation table for our sensor product. The formula for this calculation is:

Burned Calories = MET level of activity * 3.5 * Weight(kg) * minutes of activity / 200

The METALYZER will start accumulation like a stopwatch recording and will store previous sessions in non-volatile memory for historical analysis and processing. The base Python code for this type of sensor product has been designed and lightly tested using some of the HARTH sensor data. The METALYZER can be used by anyone, but it could be very useful in the managed healthcare of elderly people in retirement homes. These devices could also be RF connected to a control station at the retirement home for monitoring. For example, if the METALYZER determines that a person is lying down in the daytime, a wellness check on the person can be made to ensure they have not fallen. If there is a classification for a person not lying down during sleeping hours, then a wellness check can also be done. This type of sensor device will greatly help in the management of healthcare and safety for individuals. The METALYZER will calorie-process the timestamped sensor data streaming to our classification model to determine the recorded activity until another activity is determined. The METALYZER can run on a small, replaceable battery and will sport a small display with a few buttons with indicators for low battery to ensure continuous use. The METALYZER can be used not only for classifying human activity and calorie counting, but also for safety purposes and healthcare.

Support Vector Machine (SVM) Model

An SVM is an algorithm that creates hyperplanes, or decision boundaries, with the largest possible distance to sample points (Cortes & Vapnik, 1995). It is useful for supervised learning tasks, such as classification, which lends itself well to this dataset. The SVM model chosen for this project is implemented using the SGDClassifier estimator from the sklearn Python library

(Buitinck et al., 2013). By default, the SGDClassifier fits a linear SVM with stochastic gradient descent learning. This means the gradient of the loss is estimated at each sample and updated based on the provided learning rate parameter. Based on the library documentation, this implementation works well with floating point values for the features, and the features of the HARTH dataset are all floating-point values, based on accelerometer sensor data. For better results on the SVM model we apply normalization to the independent features. This scales the values of the independent features to fall within the range of 0 to 1. The data was split into a training and test set, with a test size of 20%.

We first run the model using the default parameters to get a baseline performance score. Default parameters of note used by the model include the loss function known as 'hinge' which gives a linear SVM, penalty is given a value of 'l2' which is the standard regularization term for linear SVMs, an alpha of '0.001' which is the constant that multiplies with the regularization term and is used to compute the learning rate, the learning_rate which is set to 'optimal' where the change in rate is determined by a preset heuristic, max_iter value of 1000 which is the number of epochs over the training data, and early_stopping which is set to False which is whether or not to terminate training when validation score is not improving. We also set a default random_state value to support reproducibility and the n_jobs parameter to -1 for faster compute time. We fit the baseline model on the training set, then ran predictions through the test set and comparing the predicted labels with the actual labels. The baseline model scored an accuracy score of 0.61, with a 5-fold cross validation resulting in an average baseline accuracy score of 0.59. The model had near zero values for precision, recall, and f1 for less prevalent activities such as Cycling (Stand), Cycling (Sit, Inactive), Cycling (Stand, Inactive), Stairs (Ascending), and Stairs (Descending), which shows that there needs to be a more even distribution of activity labels for a better

performing model. To increase performance, we utilize a hyperparameter tuning by performing a grid search using 3-fold cross validation. We selected 3 alpha values of 1e-1, 1e-4, and 1e-7 to compare. We also tested the 'hinge' loss vs 'log' loss. The log loss function gives logistic regression, a probabilistic classifier loss function. We also tested early_stopping. The best parameters that were found from the grid search were alpha: 1e-7, loss: log, and early_stopping: False. The model with these updated parameters predicted on the test set with an accuracy of 0.66, with a 5-fold cross validation resulting in an average accuracy score of 0.64, which was an increase of 5% from the baseline model. For our calorie experiment, we run the trained SVM model on each of the 22 separated datasets to predict the labels. Each activity label corresponds to a different metabolic calorie burn value which is calculated based on the metabolic equivalent formula for total calories burned. Taking the predicted labels from the model, Figure 2, Figure 3, and Figure 4 compare respectively compare the cosine similarity, label variance, and total predicted error.

Figure 2

Scatterplot of cosine similarity vs label variance

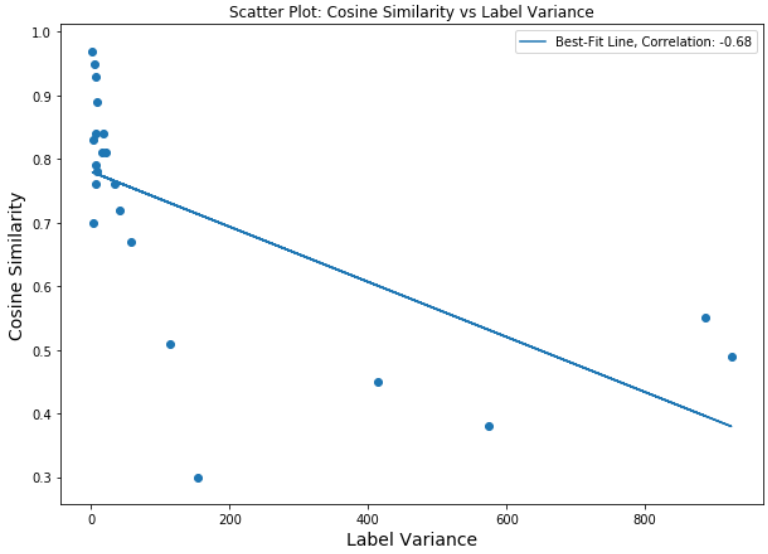


Figure 3

Scatterplot of cosine similarity vs total error

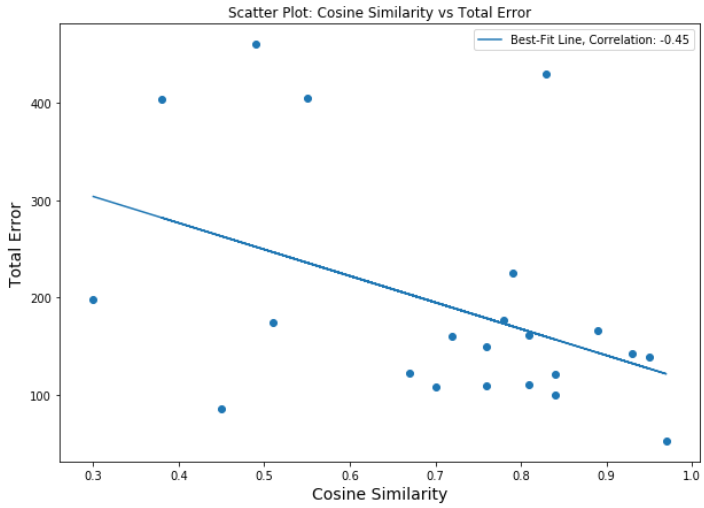


Figure 4

Scatterplot of label variance vs total error

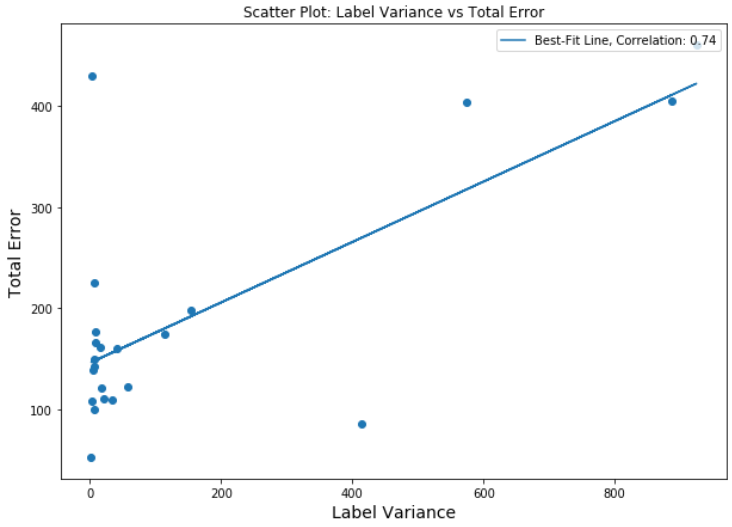


Figure 2 displays that there is a strong negative correlation between label variance and cosine similarity. Figure 3 displays a negative relationship between total absolute error in calorie burn count vs cosine similarity. Figure 4 demonstrates a positive relationship between label variance and total error. These results suggest that the model was able to predict better on datasets with fewer distinct activity labels. For example, dataset ‘012’ contained 311,722

samples of Sitting with a model prediction accuracy of 0.97, while dataset ‘029’ contained only 3,406 sitting samples and had a model prediction score of 0.3. These results show that the model is most likely suffering from overfitting with a bias toward most common label of Sitting. This supports the finding that there were not enough training examples across all activities, and that the activities with the most labels are easier for the model to predict. Potential future directions to increase performance could in include utilizing bootstrapping or data augmentation to strive for a more even distribution of activity labels.

Convolutional Neural Network (CNN) Model

CNNs are a powerful class of deep learning models primarily designed for image processing, but their principles can be extended to tabular data with innovative adaptations. At their core, CNNs operate by sliding filters across data to detect and extract important features, preserving spatial relationships in image data. When applied to tabular data, this concept can be adapted to recognize patterns and relationships between columns and rows, akin to spatial dependencies in images. The convolution operation in CNNs allows the model to focus on local patterns within the data, which can be crucial for capturing interactions between features in a table. Pooling layers, typically used to reduce the dimensionality of images while retaining essential features, can be employed in tabular data processing to down-sample the data, highlighting the most significant patterns. ReLU activation functions introduce non-linearity, enabling the model to handle complex, non-linear relationships within tabular data. The flexibility of CNNs lies in their ability to learn hierarchical features, where initial layers might capture simple interactions, while deeper layers identify more complex patterns. Figure 5 displays the structure of the CNN model. This makes CNNs particularly effective for tasks where feature interactions play a crucial role, such as predicting outcomes based on structured data. The

adaptability of CNNs to various types of data, including tabular data, highlights their versatility in deep learning, extending their applications beyond traditional image recognition to domains like healthcare, finance, and more, where structured data is prevalent.

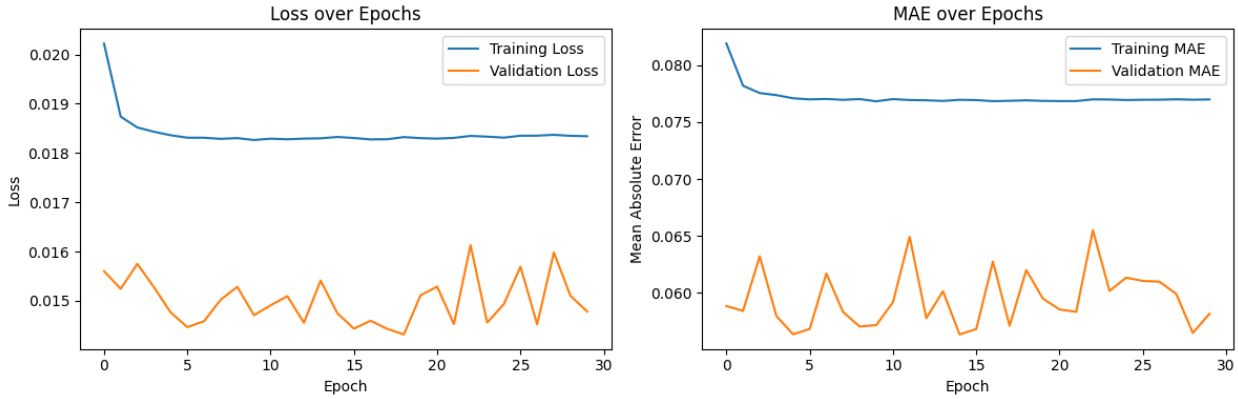
Figure 5

CNN Model Structure

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 5, 32)	96
max_pooling1d (MaxPooling1D)	(None, 5, 32)	0
dropout (Dropout)	(None, 5, 32)	0
conv1d_1 (Conv1D)	(None, 4, 64)	4,160
max_pooling1d_1 (MaxPooling1D)	(None, 4, 64)	0
dropout_1 (Dropout)	(None, 4, 64)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 128)	32,896
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Figure 6

Graphs of MAC, EPOCH, and Loss



The combined graphs of "Loss over Epochs" and "MAE over Epochs," as seen in Figure 6, reinforce the robustness and efficiency of the CNN model. Here are the key takeaways from the graphs: Key takeaways from the graphs include stable performance, strong generalization, and minimal overfitting. The training loss and MAE curves show rapid improvement in the initial epochs, followed by a stable phase, indicating that the model quickly learns and fine-tunes its

parameters effectively. The validation curves consistently remain below the training curves, demonstrating the model's ability to generalize well to unseen data, which is critical for reliable predictions in real-world applications. The relatively close alignment between the training and validation curves in both loss and MAE suggests that the model avoids overfitting, maintaining its predictive accuracy without being overly tailored to the training data.

Figure 7

Plot of Actual vs Predicted Calorie Burn

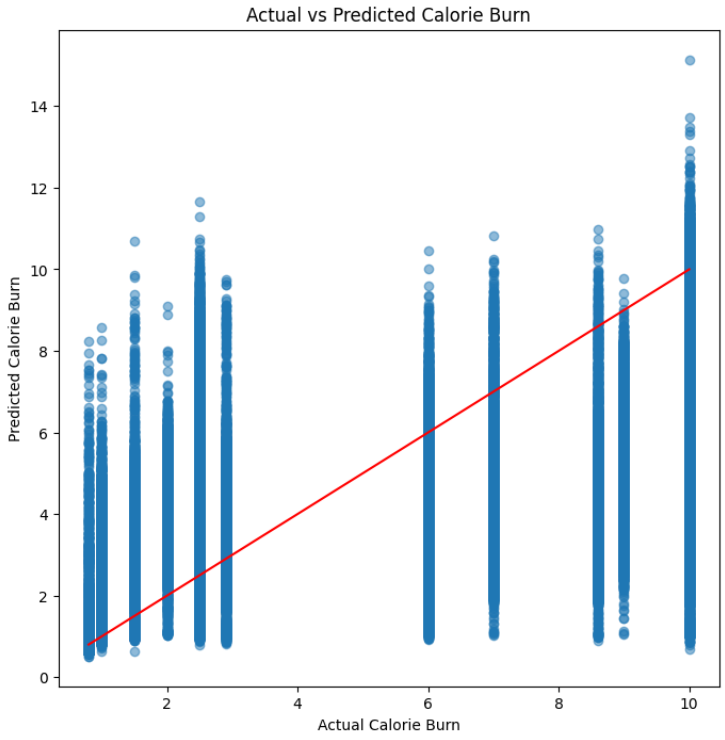


Figure 7 shows the results of the actual vs predicted calorie burn using the CNN model. Reported Mean Absolute Error (MAE) was 0.58, Mean Squared Error (MSE) was 1.48, and R-squared (R2) was 0.74. The combination of low MAE, acceptable MSE, and a solid R2 value reinforces that the CNN model is well-suited for predicting outcomes in this dataset. It provides accurate, consistent predictions with a strong ability to generalize to new data, making it a reliable choice for tasks like calorie burn estimation based on sensor data.

Bidirectional Long Short-Term Memory (bi-LSTM) Model

The LSTM is a Recurrent Neural Network (RNN) that keeps previous short-term memory and uses it for long-term training. There are three primary gates as part of an LSTM structure which include input, output, and forget type gates. The bi-LSTM can look at data forward and backwards to learn data from the future and past which makes it a more complex algorithm. The bi-LSTM is built by first defining a sequential model having a stack of layers where each layer has an input and an output. The first bidirectional layer gives the LSTM the ability to process data in forward and backwards directions finding patterns from past and future. The LSTM in the model uses 64 units that can sequence predict by learning long term patterns. A parameter was included in building the model to ensure that the input and output data are of the same length which is required for stacked LSTM layers that need full sequences as input. A dropout layer was added to prevent overfitting along with another bi-LSTM layer and another dropout layer. A fully connected dense layer was added as the final layer of the classification model to map the output to a probability. Lastly, the model was compiled, trained, and validated. The bi-LSTM is used for sequenced data analyzing the data in both directions. The K-Fold Cross Validation method was used for this model which splits the data into 5 different folds.

This bi-LSTM model was used to analyze the HARTH dataset comprised of 22 different sets of human activity accelerometer sensor data. The bi-LSTM model was trained and validated on the data and tested using a combined portion of known labeled HAR data. Several versions of the models were compiled, trained, and validated, but the average accuracy was typically around 0.87 to 0.89 as shown in Figure 8. The results of testing predictions on some of the known data had mixed results as seen below. There was some good prediction performance seen on some of the activity labels such as standing, lying down, and cycling while seated as seen in Figure 9. But

there were false predictions on these activities as shown in Figure 10. There also seemed to be different results using smaller and larger windows of sensor data for testing the model. There are areas that should be explored more to improve the model through model tuning parameters and understanding the data window sizes and how they affect the performance of the bidirectional LSTM model.

Figure 8

Bidirectional LSTM Model Summary

Average Accuracy: 0.8770

Model: "sequential_4"

Layer (type)	Output Shape	Param #
bidirectional_8 (Bidirectional)	(None, 1, 128)	36,352
dropout_8 (Dropout)	(None, 1, 128)	0
bidirectional_9 (Bidirectional)	(None, 128)	98,816
dropout_9 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 141)	18,189

Total params: 460,073 (1.76 MB)
Trainable params: 153,357 (599.05 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 306,716 (1.17 MB)

Figure 9

Bidirectional LSTM Model Correct Prediction for 6 - Standing

```
Name of HARTH Sensor Data File: S008.csv
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Predicted vs True Sensor Data Activity for 6-Standing
Predicted Activity Index: 566
True Activity: 6-Standing
Predicted Activity: 6-Standing
Total MET Calories Burned for 6-Standing :12.48
```

Figure 10

Bidirectional LSTM Model Summary False Prediction for 13 – Cycling (Sit)

```
Name of HARTH Sensor Data File: S026.csv
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 707us/step
Predicted vs True Sensor Data Activity for 13-Cycling (Sit)
Predicted Activity Index: 1338379
True Activity: 13-Cycling (Sit)
Predicted Activity: 7-Sitting
Total MET Calories Burned for 7-Sitting :41.01
```

References

Aleksej Logacjov, Atle Kongsvold, Kerstin Bach, Hilde Bremseth Bårdstu, & Paul Jarle Mork. (2023). *HARTH* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5NC90

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., Vanderplas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). *API design for machine learning software: Experiences from the scikit-learn project*. https://doi.org/10.48550/ARXIV.1309.0238

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Malinsky, A, Rajakumar, D, & Moore, G. (2024). *Machine Learning Classification for Human Activity Recognition* [Computer software]. https://github.com/ Root18D/AAI-501--Machine-Learning