Human Activity Recognition from Accelerometer Data

Sagar Raichandani
Data Science Initiative, Brown University
GitHub[8]

1. Introduction

Lower back pain is an increasingly common, expensive, and debilitating condition that affects all age groups. The inspiration for this project is drawn from the work of Sani, et al. on the SELFBACK system[1] that identifies physical activities from accelerometer data to help manage lower back pain. The dataset[2] was sourced from the UCI ML Repository. My objective through this project is to develop a subject independent classifier that identifies physical activity based on an accelerometer's readings. While the researchers predicted the L2 classes (Figure 1), the target variable for this project is L1 classification only.

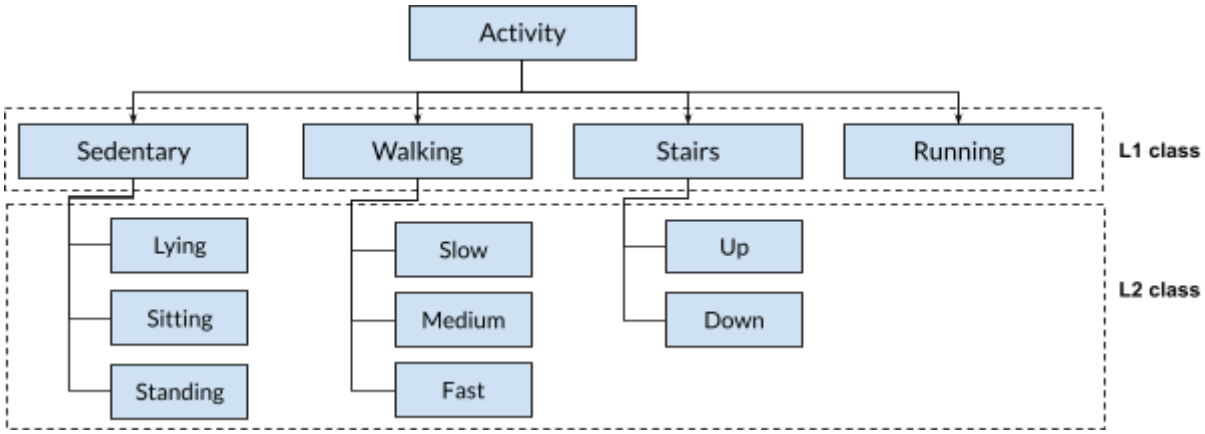

Fig 1. Flowchart of the activity classification present in the dataset.

The dataset contains 4 activity classes (L1 only) performed by 33 users; there are 4.8 million rows, and 4 features in addition to the target variable.

Feature	**Description**
Time	Time of signal measurement
x	Acceleration[g] along x-axis
y	Acceleration[g] along y-axis
z	Acceleration[g] along z-axis
Class	Activity type

Table 1. Dataset features and descriptions.

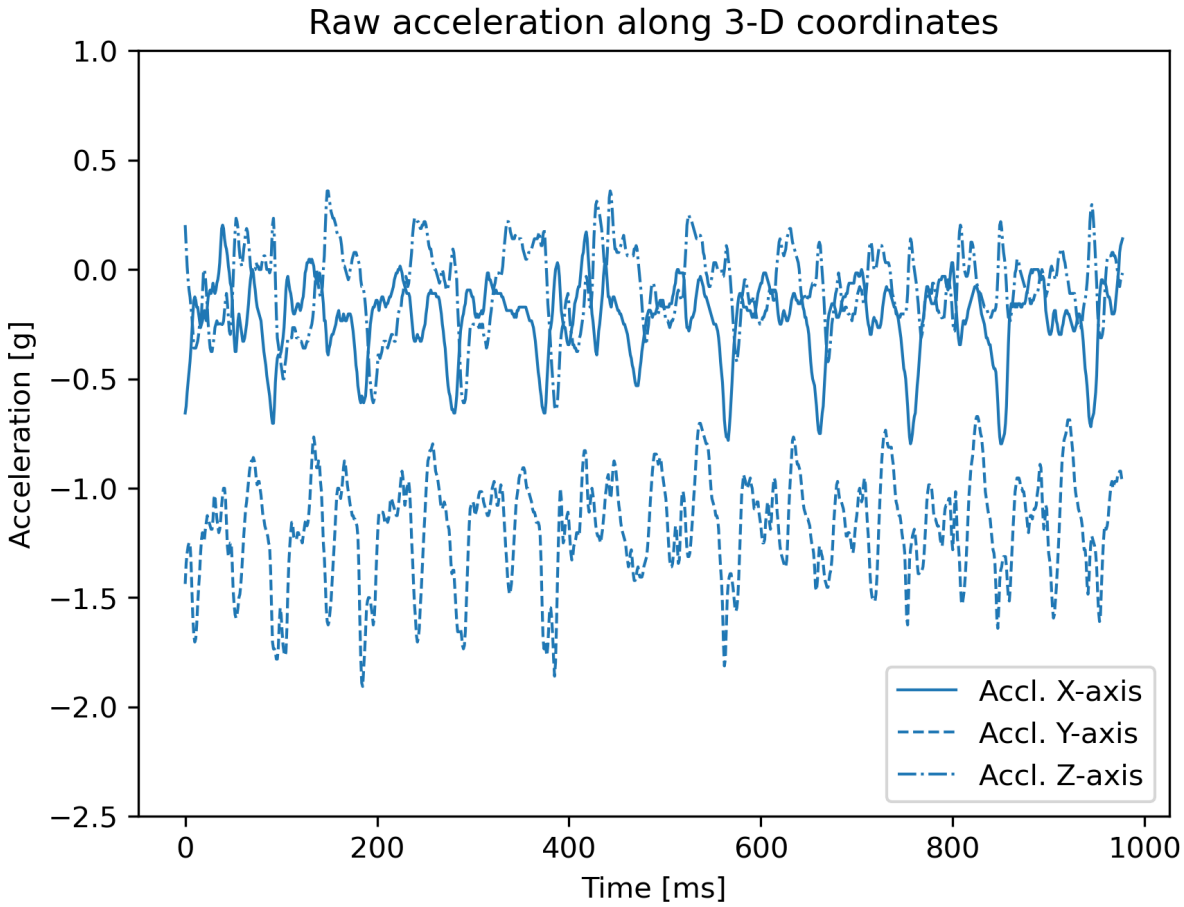

Fig 2. A plot of input acceleration along 3-D coordinates. Time in milliseconds along the horizontal, and acceleration [1g = 9.8 m/s^2] along the vertical.

Figure 2 presents a snapshot of the unprocessed readings of the accelerometer sampled at 100Hz. As evident, the readings are not IID, and hence require some processing before model development.

1.1 Windowing

Windowing [3] is the partitioning of time-series data into uniform "chunks" (10 secs[3, 4] in this case). Each window is constructed specific to a user and activity class such that each is identical and independent, forming the basis for feature engineering and classification.

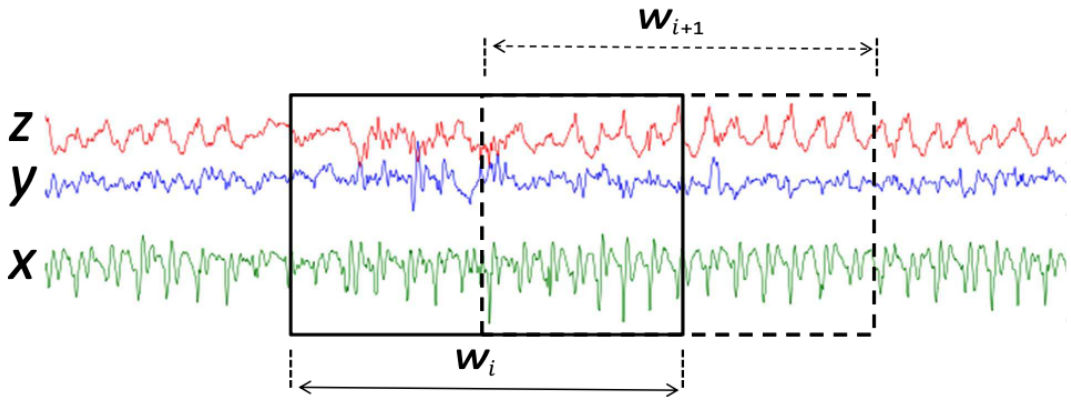

Fig 3. Illustration of accelerometer data windowing.

1.2 Feature Engineering

Feature Engineering extracts useful information from data in order to improve the model's performance. Here, I have focused on the categories presented below (figure 4):

1. **Time domain-** functions that depend upon and vary with time.
2. **Frequency domain-** functions that are defined by frequency.
3. **Discrete Cosine Transform-** DCT expresses a finite discrete signal as a sum of cosine functions at different frequencies. DCT compresses the energy of a signal into an ordered set of coefficients such that the most significant information is in the lowest indices.

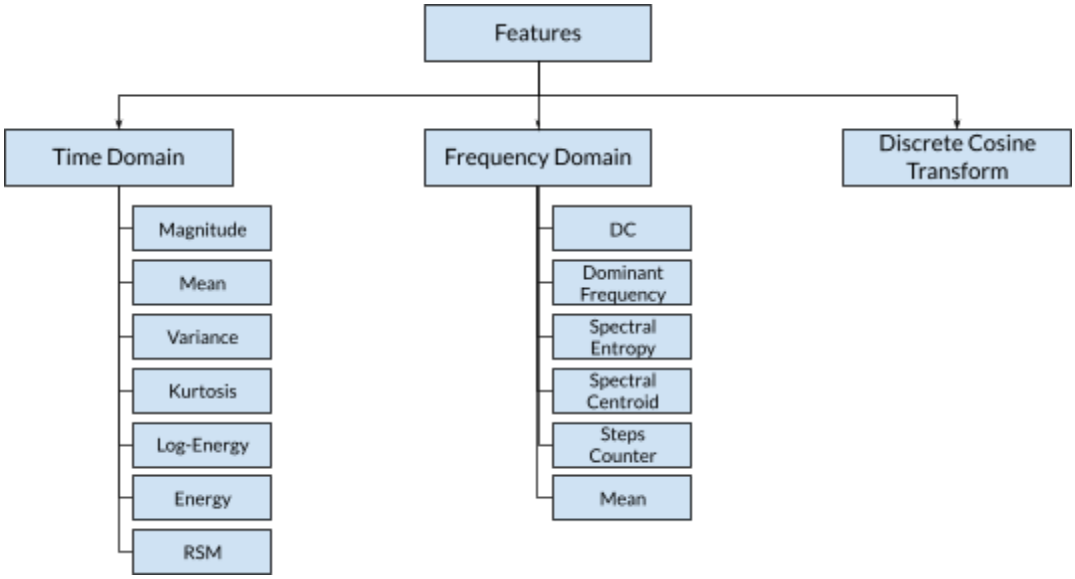

Fig 4. Flow chart of features extracted from the dataset.

To limit model complexity, the features are split into two groups and a comparative analysis is presented in section 4:

1. Combination of 80 time and frequency features; 2.
2. First 48 components[4] of the result of the Discrete Cosine Transform on the acceleration along X, Y, Z, and magnitude $\left(m = \sqrt{x^2 + y^2 + z^2}\right)$ resulting 196 features.

2. Exploratory Data Analysis

Having reduced rows and expanded usable features, we begin EDA to get a better understanding of the underlying class imbalance, distribution of extracted quantities, and relation among features and target variable. We start with class ratios to gauge imbalance (Figure 5)-- classes *sedentary* and *walking* have a significant majority with 37% of points in each.

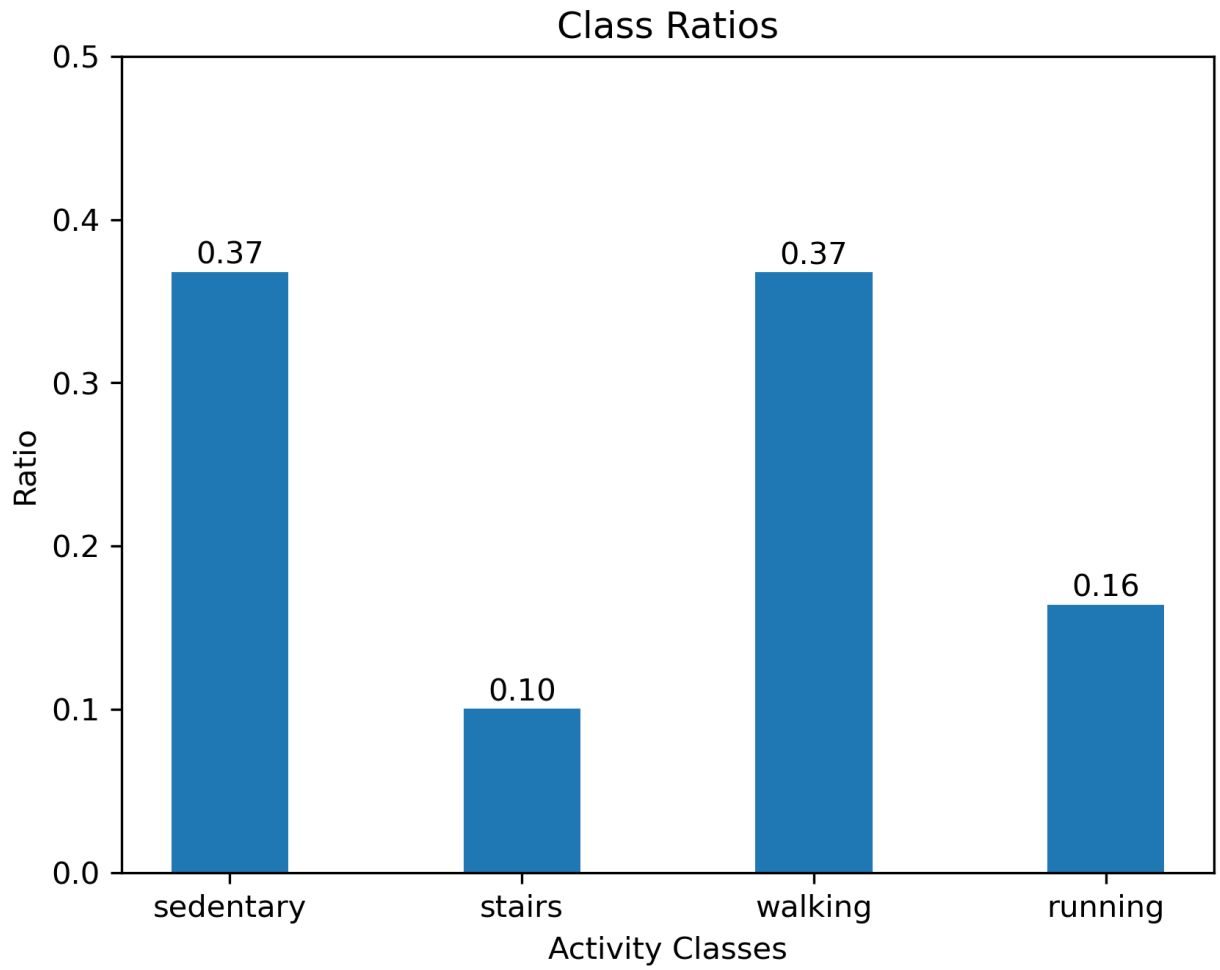

Fig 5. A bar chart of the class ratios based on L1 classification. Vertical axis represents the ratio of class points off the total. The horizontal axis represents class labels.

Next, consider the change in magnitude of acceleration by activity class intensity. Figure 6 illustrates that as activity intensity increases, so does the magnitude of acceleration.

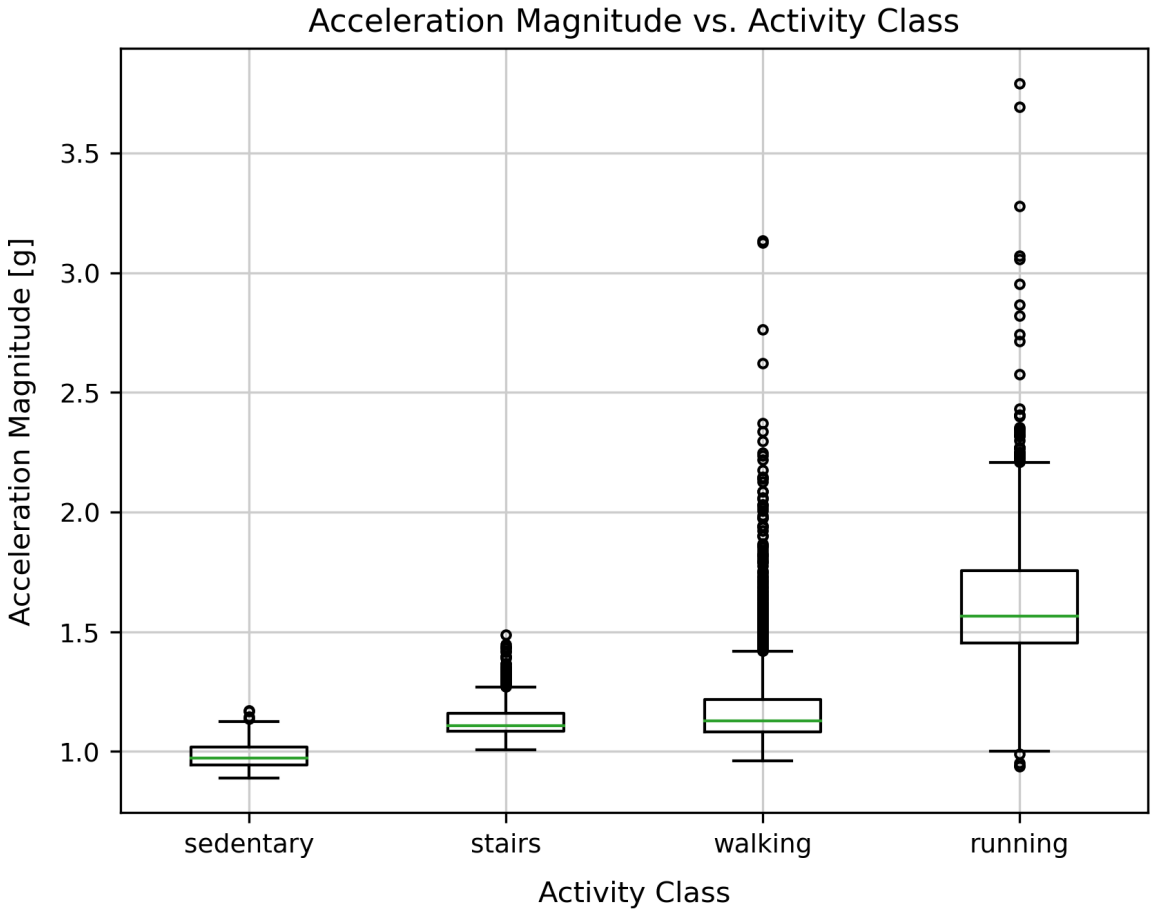

Fig 6. Box-plot comparing the acceleration magnitude of each activity- class on the horizontal axis and acceleration [g] on the vertical axis. As the activity intensity increases so does the magnitude.

Likewise from figure 7, we see that the dominant frequency (one that carries the most energy with respect to all others in the spectrum) increases with an increase in activity intensity.

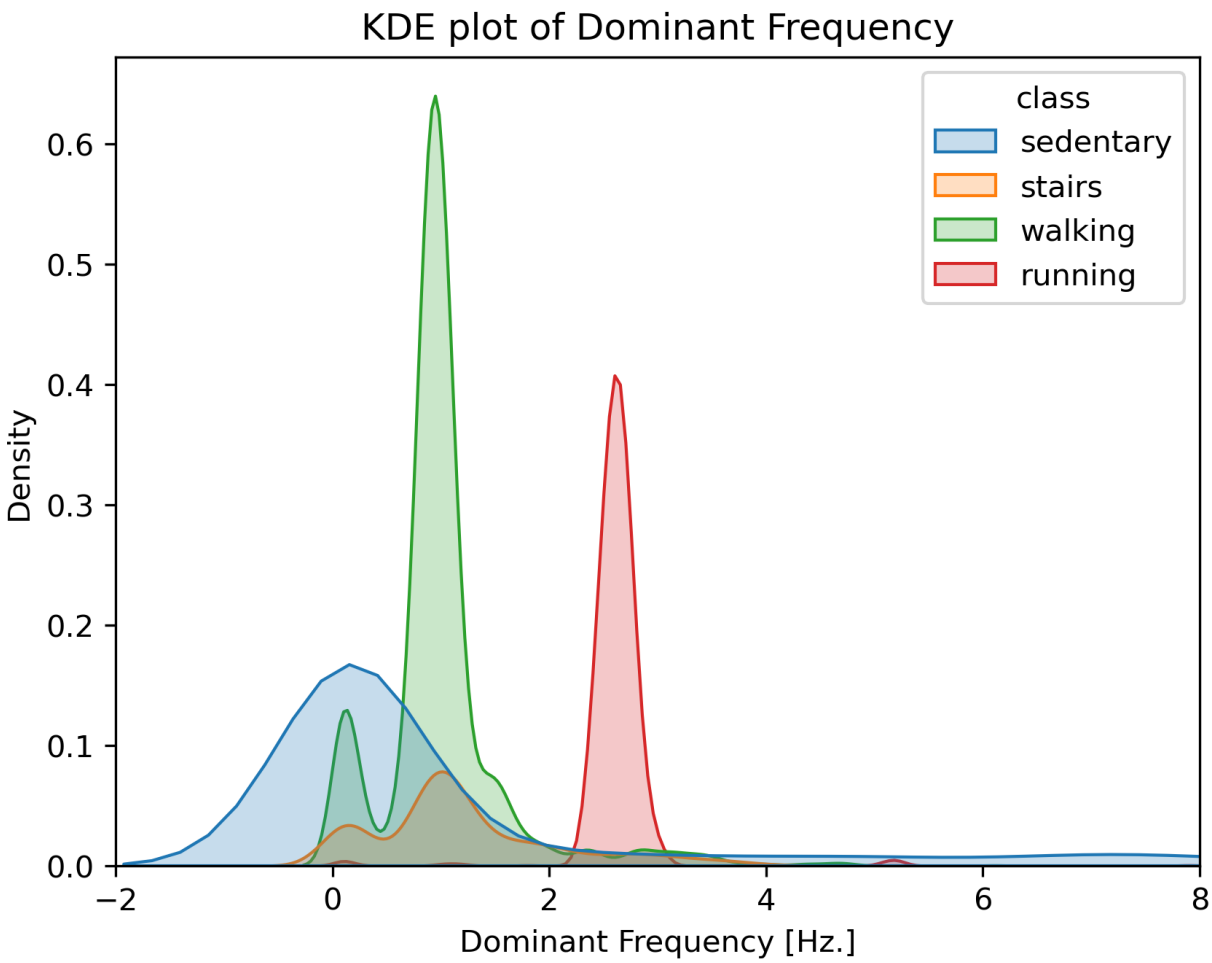

Fig 7. A kernel density estimate plot of the dominant frequency for each activity class. With dominant frequency on the horizontal axis and density on the vertical axis, each activity exhibits a different dominant frequency.

Lastly, taking a look at the relationship between log-energy and the number of steps taken, we see that as activity increases the number steps taken per window increases; however the spread about log-energy remains quite high.

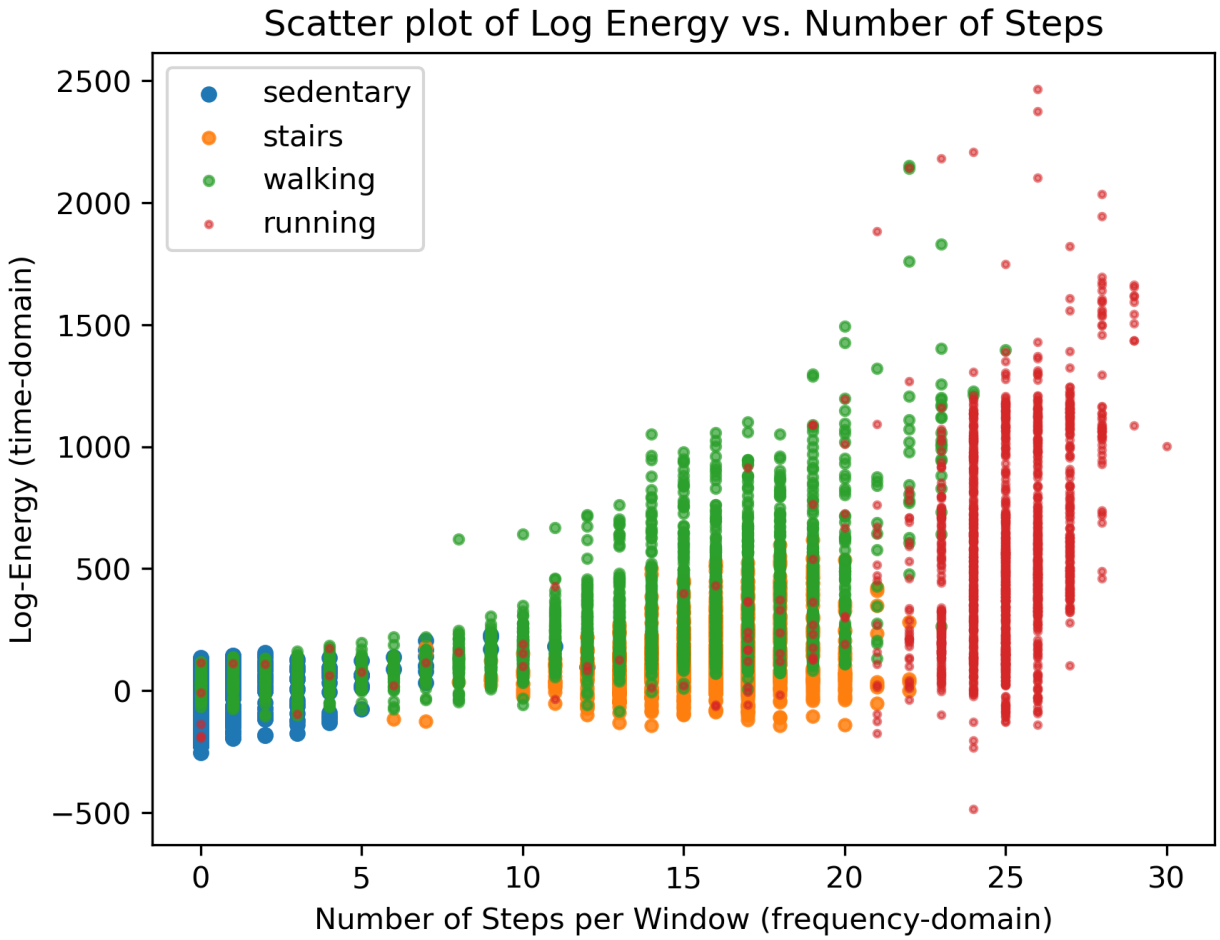

Fig 8. A scatter plot of the relationship between the number of steps taken and log-energy.

3. Methodology

The next step of model development deals with train-validation-test splits, feature preprocessing, and hyperparameter tuning. We train and evaluate the performance of four models– Logistic Regression (L2), Support Vector Classifier, Random Forest and XGBoost– based on F1 macro scores. Since this is a multi-class classifier with imbalanced classes, the F1 macro score works well for the following reasons:

1. Easy to interpret.
2. Resilient to majority skew since it weighs each class equally.
3. Precision and recall weighed equally since the cost of misclassification is the same for all classes.

3.1 Splitting

Since our goal is to predict the activity class for previously unseen users there are three factors affecting our dataset:

1. Group structure based on the user.
2. Unshuffled data as windows are built sequentially.
3. Class imbalance (figure 5).

Bearing the above, we build the test set using sklearn's *GroupShuffleSplit* method (with *n_splits*=1 and *test_size*=6) to achieve a 20% split of the 33 users.

3.2 Preprocessing (Feature Scaling)

All features are numerical and continuous- hence, sklearn's *StandardScaler* is the optimal choice. The *StandardScaler*, sets mean to 0 and the standard deviation to 1. This process has no impact on the shape of the data; unlike *OneHotEncoder*, *StandardScaler* does not add any columns to the feature matrix.

3.3 Hyperparameter Tuning and Cross Validation

The purpose of hyperparameter tuning is to identify the parameters that optimize the model's performance with respect to the evaluation metric. The general idea is to iterate through several parameter configurations, train the model on a portion of the training data (*CV train*), and evaluate performance against a validation set (*CV test*). Once the optimum parameter configuration is identified, the model is refit and evaluated on the test set. The validation sets are built using sklearn's *GroupKFold* method (with *n_splits=7*) to ensure that group structure is maintained as in section 3.1.

	Logistic Regression (L2)	Support Vector Classifier	Random Forest Classifier	XGBoost Classifier
Classification Strategy	One vs. Rest	One vs. One	Multiclass	Multiclass
Parameters Tuned	C- regularization inverse (log scale)	C- regularization inverse (log scale)	Max features (linear scale)	Learning rate (log scale)
		gamma- kernel coefficient (log scl)	Max depth (linear scale)	Max depth (linear scale)
Optimum Parameters	C = 100	C = 100; gamma = 0.01	Max features = 0.5; Max depth = 12	Learning rate = 0.5; Max depth = 8

Table 2. Models trained and hyperparameters tuned.

3.4 GridSearchCV and Model pipeline

The steps described in sections 3.2 and 3.3 are performed via sklearn's GridSearchCV. Once the best parameters are identified, the model configuration is saved, and the entire process (from 3.1 to 3.3) is repeated with a different random state. This is done in order to estimate the mean performance of the model and gauge the deviation from the mean across different datasets, increasing confidence in the results. A summary of the pipeline (for one random iteration) is presented in Figure 9.

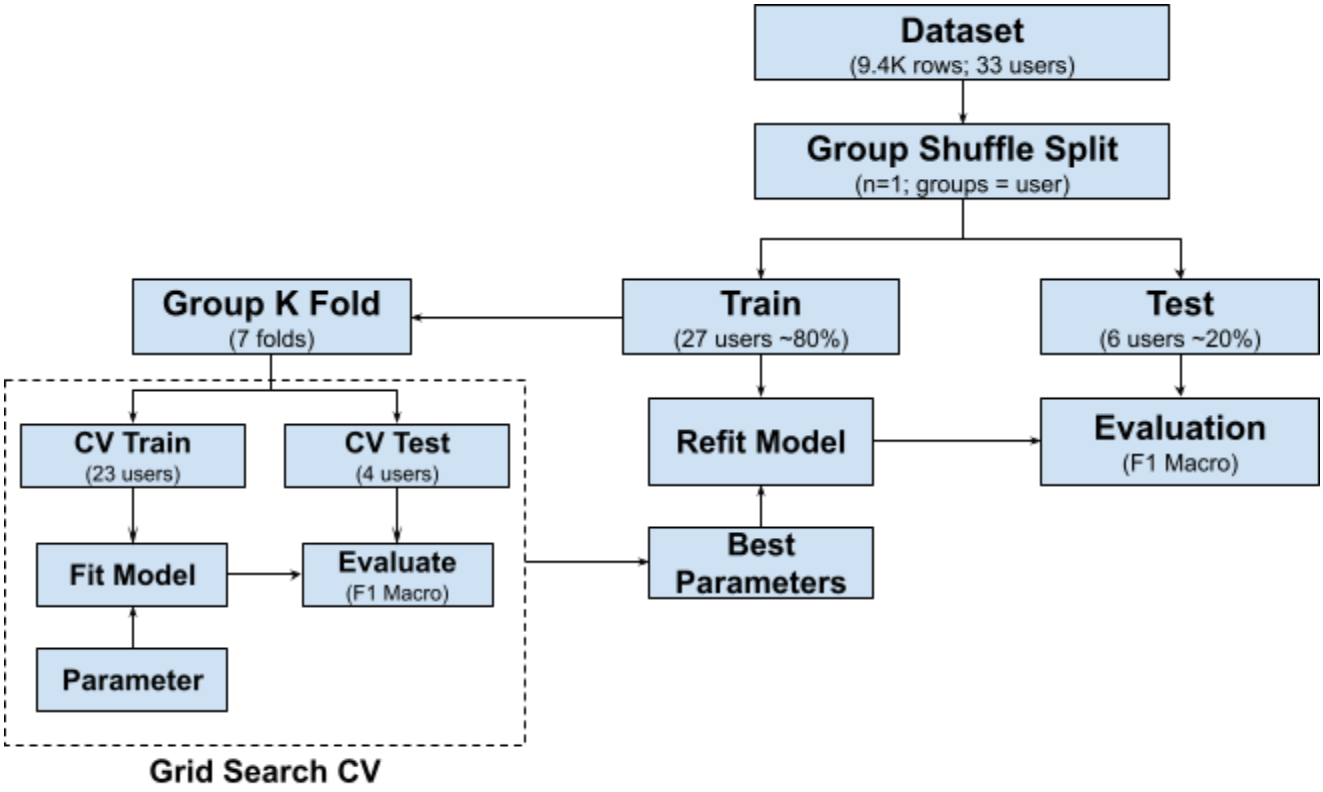

Fig 9. Flow chart of the model development and cross validation pipeline.

4. Results

The steps described in section 3 are repeated over 5 random states to gauge model performance with different train-validation-test set combinations, and account for variability in non-deterministic models. The resultant scores are summarized via the mean and standard deviation of each model's F1 macro scores. Also, associated with each random test set, a baseline model's (using sklearn's *DummyClassifier* method) F1 macro score is recorded. This process is done for two feature sets (time-frequency features and DCT) to identify the best classifier and the optimum feature set.

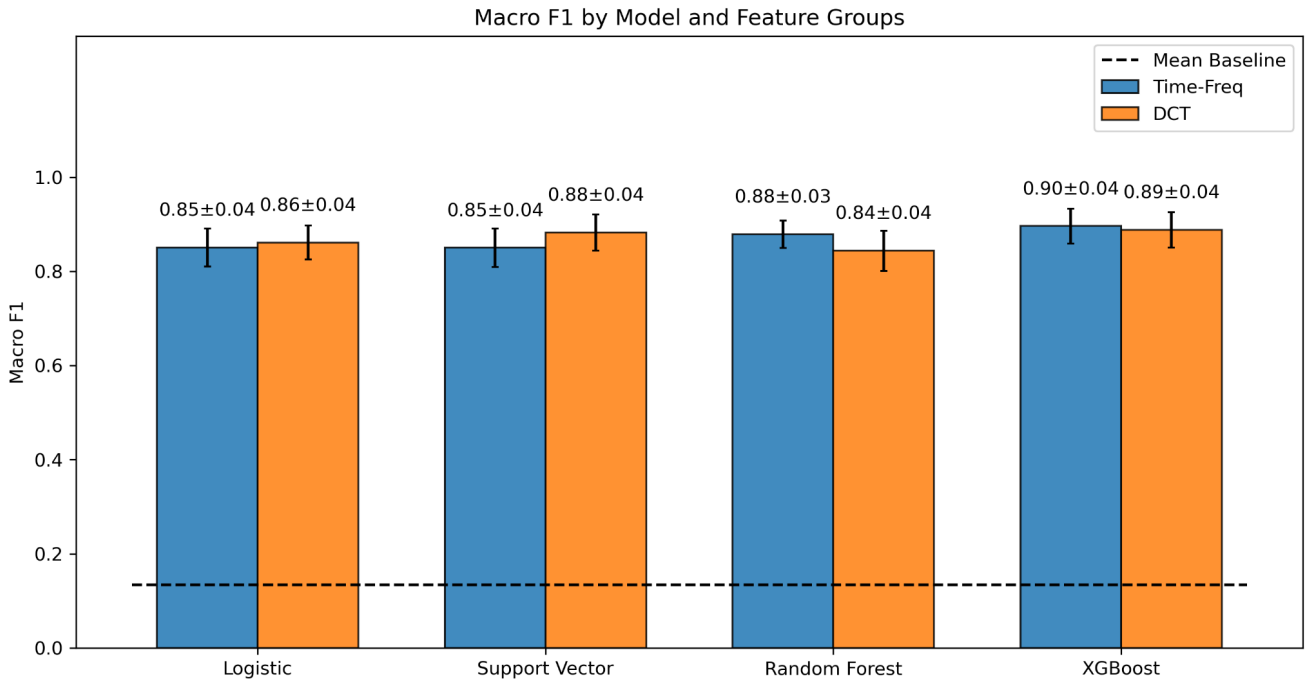

Fig 10. Comparison of macro F1 scores split by feature groups and compared to mean baseline macro F1 scores. The X-axis holds model type, Y-axis shows macro F1, and bars represent different feature sets.

From figure 10, the mean macro F1 of each of the models is significantly better than the mean baseline of 0.135. Furthermore, the XGBoost classifier with time-frequency features outperforms the other models with a mean F1 of 0.9– more than 10 standard deviations over the baseline F1. The best XGBoost model's confusion matrix is presented in figure 11.

Fig 11. XGBoost Classifier Confusion Matrix (normalized by 'true' labels; row-wise)

Figure 11 shows that while the macro F1 is high, the classifier shows low recall for the minority class stairs– with 28% of stairs activity and 14% of running instances being misclassified.

4.1 Global Feature Importance

Global Feature Importance metrics help gauge the contribution of a feature to the predictability of the model. 3 types of importance metrics for the best XGBoost model are:

1. Permutation feature importance: estimates predictive influence based on the drop in model performance due to random perturbation of a feature.
2. XGBoost Total Gain: estimates predictive influence based on the improvement in accuracy brought on by the addition of a feature.
3. Shap values: estimates predictive influence based on Shapley values from game theory.

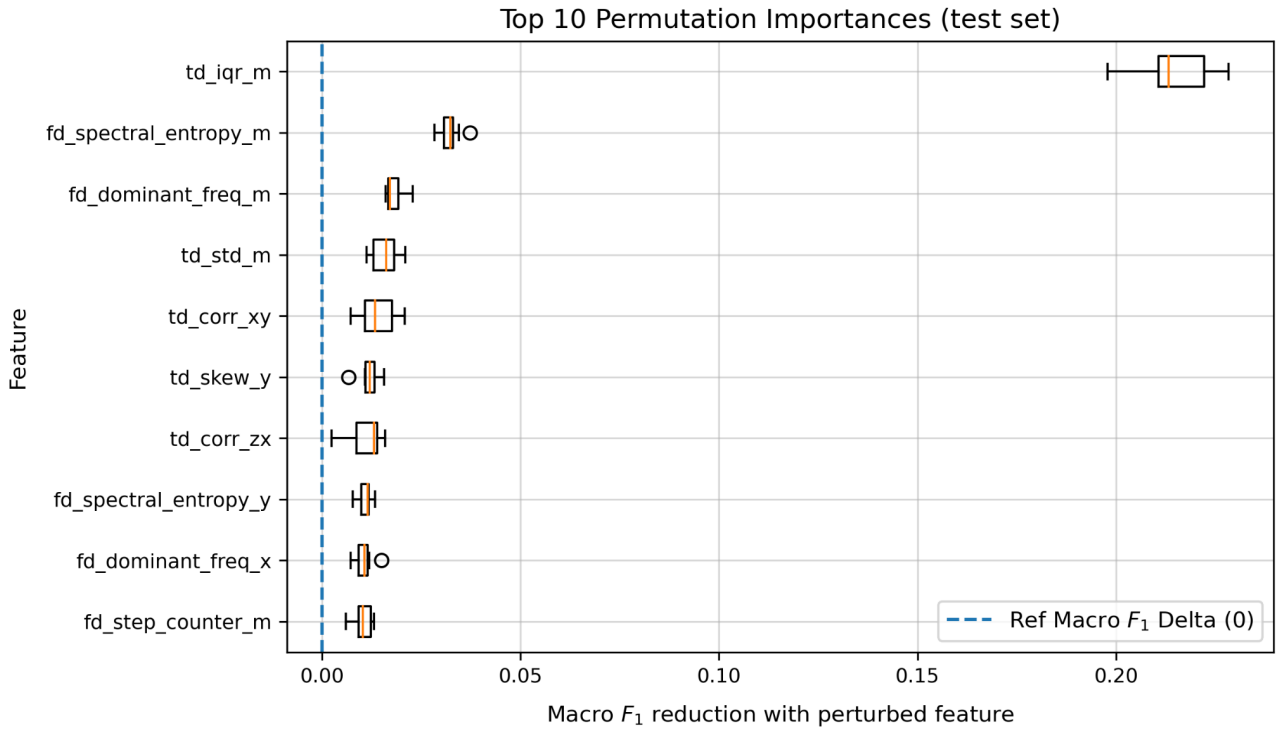

Fig 12. Top 10 most important features as per permutation feature importance.

From figure 12, we find that *td_iqr_m* (interquartile range of magnitude), *fd_spectral_entropy* (measure of forecastability), and *fd_dominant_frequency* (frequency that carries the most energy) are the top 3 features as per permutation importance.

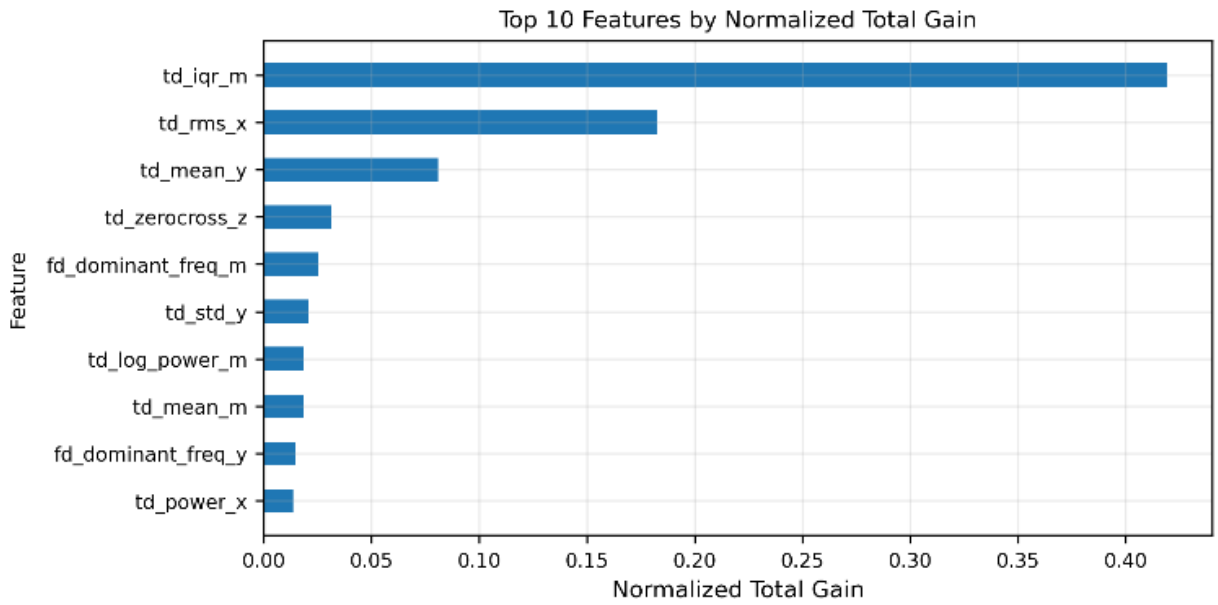

Fig 13. Top 10 most important features as per XGBoost relative importance measure- *total gain*.

From figure 13, we find that *td_iqr_m, td_rms_x* (RMS of acceleration along X-axis), and *td_mean_y* (mean acceleration along Y-axis) are the most important for prediction as per XGBoost's relative importance measure- *total gain.*

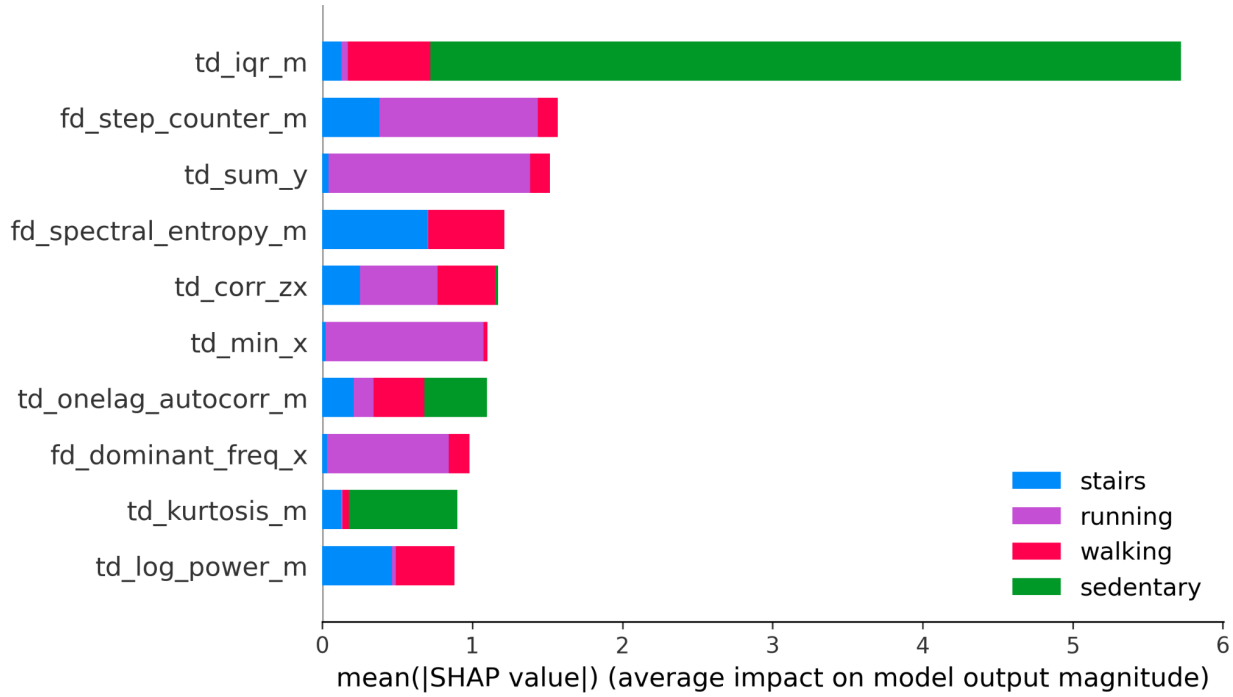

Fig 14. Top 10 most important features as per mean Shap value.

Lastly, from figure 14, we find that *td_iqr_m, fd_step_counter_m* (number of steps in a 10 sec time window), and *td_sum_y* (acceleration along Y-axis) are the most important features as per mean Shap values.

Key observations:

1. While there is some agreement on top 10 features across the different methods, the ranks vary strongly.
2. Global shap values demonstrate that the influence of features vary strongly by class.
3. Interquartile range of the magnitude is ranked at the top across all 3 methods.

4.2 Local Feature Importance

Local Feature Importance metrics help gauge the contribution of a feature to the predictability of a certain point. Note that, shap values for multi-class XGBoost models report log odds as opposed to predicted probabilities [6,7]; while this reduces interpretability, in general, higher log odds correspond to greater probabilities.

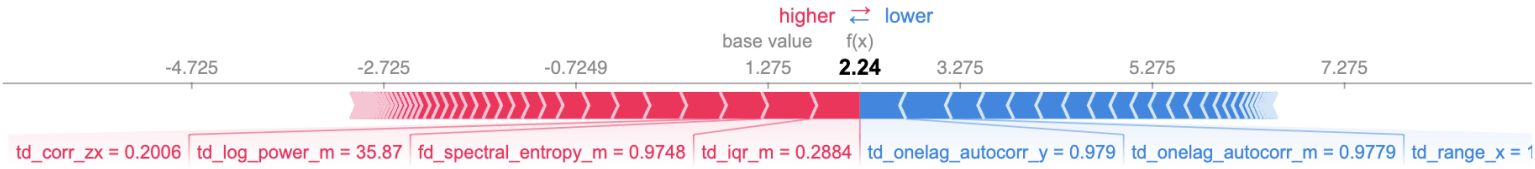

Fig 15. Local feature importance shap values for point 1, class *walking*. Each feature's importance + expected value sums to the log(odds) of 2.24 for classifying this instance as *walking.*

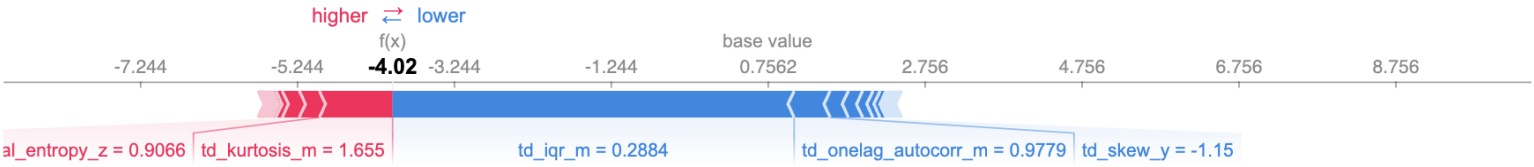

Fig 16. Local feature importance shap values for point 1, class *sedentary*. Each feature's importance + expected value sums to the log(odds) of -4.02 for classifying this instance as *sedentary.*

Figures 15 and 16 demonstrate that feature importance sums to the log odds of classifying that instance as a particular activity. In the above example, the probability can be calculated by taking the inverse logit of the class log odds; here, the predicted probability is 0.882 *walking*, 0.114 *running*, 0.003 *stairs* and 0.002 *sedentary*.

Outcomes

Having spent a couple months diving deeper into this problem, I find that the project could be improved by taking into account the following:

- Additional data through heart rate, skin conductivity, or multiple accelerometers would likely help increase recall for minority classes- *stairs* and *running*.
- Tune window length (2, 5, 10 secs) to create a balance between accuracy and latency.
- Gauge impact of β in F_β - the lower recall of minority classes made me question if weighing recall higher would be a more appropriate evaluation metric.
- Estimate feature importance (permutation/global shap) after dropping correlated variables.
- Principal component analysis to reduce dimensionality of the feature matrix.
- Train models for L2 classification.

References

[1] kNN Sampling for Personalized Human Activity Recognition
Sadiq Sani, Nirmalie Wiratunga, Stewart Massie, and Kay Cooper
[2] UCI Machine Learning Repository: selfBACK data set
[3] Physical Activity Recognition from Accelerometer Data Using a Multi-Scale Ensemble Method
Yonglei Zheng, Weng-Keen Wong, Xinze Guan, Stewart Trost
[4] Activity Recognition from acceleration data based on Discrete Cosine Transform and SVM
Zhenyu He, Lianwen Jin
[5] Placing the global burden of lower back pain in context
[6] Interpretation of multi-class classification with shap
[7] Shap values in XGBoost outside [0,1]
[8] GitHub repository - https://github.com/raichandanisagar/human-activity-recognition