

Application of Machine Learning Methods to Real-World Data

Machine Learning and Data Mining – 7021DATSCI

Group Name – Deep Strivers

Group Members – Thet Khine Lin, Samar Ben Zekri

Table of Contents

[Projct Plan 3](#_Toc194429211)

[Section – 1: Understanding the Given Datset 4](#_Toc194429212)

[Section – 2: Feature Engineering 4](#_Toc194429213)

[Section – 3: Exploratory Data Analysis (EDA) 6](#_Toc194429214)

[Section – 4: Model Development 7](#_Toc194429215)

[Model Development Workflow 7](#_Toc194429216)

[Gradient Boosting Machine (GBM) 9](#_Toc194429217)

[Random Forest Classifier (RF) 10](#_Toc194429218)

[K-Nearest Neighbors (KNN) 11](#_Toc194429219)

[Artificial Neural Network (ANN) 13](#_Toc194429220)

[Section – 5: Discussion of Results and Best Model Selection 15](#_Toc194429221)

[Future Improvement 16](#_Toc194429222)

Abstract

This project focuses on Human Activity Recognition (HAR) using machine learning techniques. The goal is to develop a model that can accurately classify different physical activities based on sensor data collected from personal digital devices. The process begins with exploring the raw signals and metadata, followed by feature engineering and exploratory data analysis to improve data quality. Several classification models are developed using provided train set and evaluated using both a provided test set and a Kaggle competition dataset. The models include K-Nearest Neighbors (KNN), Random Forest (RF), Gradient Boosting Machine (GBM), and an Artificial Neural Network (ANN). Hyperparameter tuning is applied to enhance model performance. Among all models, the ANN achieved the best overall results, showing strong generalization ability and consistent performance across different datasets.

# Projct Plan

**A diagram of a process

Description automatically generated**

Figure – 1: Overview of Project Workflow

This report is structured according to the workflow illustrated in the Figure-1. Each section in this report corresponds to a specific stage in the process.

# Section – 1: Understanding the Given Datset

The given dataset consists of two type, signals data and metadata data. The signals data contains raw time series data representing sensor readings, while metadata data provides a compressed summary of this data. Each user\_snippet in the metadata corresponds to a segment of signal data, typically around 100 rows, though the exact number may vary across snippets. The features in metadata data are statistical summaries such as mean, standard deviation, and other aggregated metrics derived from the corresponding segment of signal data. This compression helps to reduce noise, which might be caused by sensor inaccuracies, sudden jerks and external influences, and improve model performance by highlighting high-level trends. Consequently, metadata is used for model training and evaluation instead of the raw signal data.

However, a limitation of this approach is that the compressed features in metadata do not capture the temporal changes within each user snippet. This loss of sequential information may hinder the model's ability to recognize certain activity patterns. Therefore, feature engineering from the original signals data is necessary to better capture distinctions between activity classes.

# Section – 2: Feature Engineering

To improve activity classification, a range of features were engineered from signals data and integrated into metadata. These features capture statistical, temporal, and frequency characteristics of the sensor data.

(Note: In the feature names below, \* represents each of the three axes: x, y, and z.)

**1. Correlation Features**

* **corr\_xy, corr\_xz, corr\_yz**
  + These measure the Pearson correlation between acceleration values along different axes. High correlation suggests coordinated movement across those axes, common in activities like walking or jogging.

**2. Jerk Features**

* **jerk\_\*\_mean, jerk\_\*\_std**
  + Jerk is computed as the first derivative of acceleration, representing how quickly the acceleration changes. Mean and standard deviation summarize its overall level and variability, helping to separate dynamic actions from stillness.

**3. Signal Magnitude Area (SMA)**

* **sma**
  + Calculated as the average of the sum of absolute values of x, y, and z accelerations over a window. It reflects total activity intensity, where higher values indicate stronger or more vigorous motion.

**4. Statistical Features**

* **\*\_MAD, \*\_iqr, \*\_range**
  + MAD (Median Absolute Deviation): Measures average deviation from the median.
  + IQR (Interquartile Range): Difference between the 75th and 25th percentiles.
  + Range: Max minus min value.  
    These capture data spread and variability, important for distinguishing steady from erratic motion.

**5. Rolling Statistics**

* **\*\_rolling\_mean, \*\_rolling\_std**
  + Computed over sliding windows to capture short-term changes. These are helpful in detecting sudden bursts of activity or consistent trends over time within a user snippet.

**6. Shape Descriptors**

* **\*\_skewness, \*\_kurtosis**
  + Skewness quantifies asymmetry in the data distribution.
  + Kurtosis indicates the sharpness or flatness of the distribution peak.  
    These help identify unusual or bursty motion that departs from typical patterns.

**7. Normalized Variation**

* **\*\_coeff\_var**
  + This is the standard deviation divided by the mean. It normalizes variability, allowing meaningful comparisons across axes with different magnitude scales.

**8. Frequency Domain Features**

* **\*\_energy, \*\_spectral\_energy**
  + Energy is calculated as the sum of squared signal values.
  + Spectral Energy applies the same idea after applying Fourier Transform.
  + These indicate how much "power" or movement is concentrated in different frequency bands.
* **\*\_spectral\_entropy**
  + Derived from the power spectral density (PSD), this measures the randomness or unpredictability in the signal. High entropy implies more chaotic or irregular movements.
* **\*\_fft\_peak**
  + The frequency with the highest amplitude in the Fast Fourier Transform (FFT) result. It captures the most common repeating motion in the signal, such as the rhythm of steps while walking or regular hand movements.

**9. Complexity and Transition Metrics**

* **\*\_hjorth\_mobility, \*\_hjorth\_complexity**
  + Mobility represents the standard deviation of the signal derivative divided by the original signal's standard deviation.
  + Complexity compares the second derivative’s mobility to the first.
  + These features quantify how quickly and irregularly the signal fluctuates.
* **\*\_zero\_crossing\_rate**
  + Counts how often the signal changes sign (positive to negative or vice versa). It's a proxy for frequency and is useful in detecting sharp transitions or tremors.

# Section – 3: Exploratory Data Analysis (EDA)

The purpose of Exploratory Data Analysis (EDA) in this project is to understand how the features relate to the target variable (activity) and to one another. This process informs both feature selection and model choice to ensure optimal performance.The EDA was carried out in three key steps:

1. **Normality Check**:

* I began by testing the distribution of each feature using the Anderson-Darling test. The results indicated that none of the features follow a normal distribution. Based on this finding, I selected the Kruskal-Wallis H Test—a non-parametric statistical test appropriate for non-normal data—to evaluate the relationship between each feature and the activity labels.

1. **Feature Significance Testing**:

* The Kruskal-Wallis test revealed that most features showed a significant difference across activity categories. However, three features, which are jerk\_x\_mean, jerk\_y\_mean, and jerk\_z\_mean, did not show statistical significance and were therefore removed from the dataset manually. This step helped reduce noise and improve the relevance of the input features.

1. **Multicollinearity Analysis**:

* Next, I assessed the degree of multicollinearity among features using Pearson correlation. The correlation matrix in Figure – 2 showed that many features are highly correlated with each other. Since high multicollinearity can negatively affect linear models, I decided to focus on non-linear machine learning models that are more tolerant of correlated inputs. As a result, I selected Gradient Boosting Machine (GBM), Random Forest (RF), K-Nearest Neighbors (KNN), and Artificial Neural Network (ANN) for model development.

These findings helped guide the model selection strategy and ensured that only the most relevant features were retained for training.

A screenshot of a computer generated image

Description automatically generated

Figure – 2: Feature Correlation Heatmap

# Section – 4: Model Development

For this project, four machine learning models were developed and evaluated: Gradient Boosting Machine (GBM), Random Forest Classifier (RF), K-Nearest Neighbors (KNN), and an Artificial Neural Network (ANN). While GBM, RF, and KNN used the same data preprocessing pipeline, the ANN model required a slightly different approach tailored to neural network training. Each model underwent hyperparameter tuning to optimize performance using appropriate search strategies and evaluation metrics.

## Model Development Workflow

A diagram of a process flow

Description automatically generated

Figure – 3: Detailed Model Development Workflow

The workflow shown in Figure – 3 is carried out through the following sequential steps:

1. The process begins with the original metadata, which is enhanced through feature engineering by extracting additional features from the raw signals data. The resulting dataset is saved as Updated\_Metadata.
2. Next, exploratory data analysis (EDA) is performed to assess data distribution, correlation between features and the target (activity), and multicollinearity. Based on this analysis, significant features are selected and insights into multicollinearity help guide the selection of suitable machine learning models. The refined dataset is saved as Updated\_Metadata\_2.
3. The models are trained using Updated\_Metadata\_2. During data preprocessing, the activity column is encoded using label encoding for the GBM, RF, and KNN models, whereas one-hot encoding is applied for the ANN model to suit its neural network architecture. Hyperparameter tuning is applied to the GBM, RF, and KNN models using GridSearchCV, while manual tuning is used for the ANN model. All models are then tested using both the test dataset and the Kaggle dataset.
4. The trained models are evaluated based on Balanced Accuracy, Standard Accuracy, and Kaggle Accuracy to assess their performance.
5. Finally, the model that performs best across all evaluation criteria is selected as the best-performing model.

Note: Steps from Feature Engineering to Model Evaluation are treated as part of an iterative improvement loop, repeating as needed until the optimal model and hyperparameters are found.

## Gradient Boosting Machine (GBM)

**Hyperparameter Tuning**

To optimize the GBM model, several key hyperparameters were tuned using GridSearchCV with 5-fold cross-validation. The search was performed across the following grid:

A screenshot of a computer program

Description automatically generated

Figure – 4: Hyperparameter Tunning for GBM

* n\_estimators: Controls the number of boosting stages. Higher values improve model capacity but significantly increase training time.
* learning\_rate: Determines how much each tree contributes. Lower values lead to more gradual learning but often better generalization.
* max\_depth, min\_samples\_split, and min\_samples\_leaf: Help regulate tree complexity and reduce overfitting by limiting growth.
* max\_features: Restricts the number of features considered per split, enhancing tree diversity and reducing overfitting.

The best combination found was:

* 'learning\_rate': 0.2
* 'max\_depth': 5,
* 'max\_features': 'log2',
* 'min\_samples\_leaf': 4,
* 'min\_samples\_split': 5,
* 'n\_estimators': 100

**Training Experience**

Among all models, GBM had the longest training time. This is expected due to its sequential nature, where each tree depends on the residuals of the previous one. Additionally, GridSearchCV's exhaustive search across multiple parameter combinations, combined with 5-fold cross-validation, multiplied the training time significantly.

**Model Performance on Test Data & Interpretation**

**A screenshot of a graph

Description automatically generated**

Figure – 5: Classification Report of GBM on Test Data

The Figure – 5 shows that GBM achieved a balanced accuracy of 0.783 and an overall accuracy of 0.86. The classification report in Figure – 5 reveals strong performance in classes like Jogging and Walking, with F1-scores of 0.96 and 0.90, respectively. However, performance dropped for less represented or more complex activities like Downstairs (F1-score: 0.57) and Upstairs (F1-score: 0.63), where recall was particularly low. This indicates that while GBM is capable of modeling the general patterns in the data well, it may struggle with more underrepresented classes. Still, the model showed solid generalization and competitive performance across the board.

## Random Forest Classifier (RF)

**Hyperparameter Tuning**

Hyperparameter optimization for RF was performed using GridSearchCV with 5-fold cross-validation. The following parameter grid was explored:

A screenshot of a computer code

Description automatically generated

Figure – 6: Hyperparameter Tunning for Random Forest

* n\_estimators: Determines the number of trees in the forest. More trees generally improve performance but increase training time.
* max\_depth: Limits how deep each tree can grow, helping to control overfitting.
* min\_samples\_split and min\_samples\_leaf: Set constraints on how splits are made in the tree, improving generalization by avoiding overly specific branches.
* max\_features: Controls the subset of features considered when splitting, adding randomness for better robustness.
* bootstrap: Enables sampling with replacement, enhancing diversity among trees.

The best combination found was:

* 'bootstrap': True
* 'max\_depth': 20
* 'max\_features': 'log2'
* 'min\_samples\_leaf': 2
* 'min\_samples\_split': 10
* 'n\_estimators': 100

**Training Experience**

The RF model took less time to train compared to GBM, but longer than KNN. Unlike GBM, RF builds trees in parallel rather than sequentially, which significantly reduces training time. However, evaluating multiple combinations of deep trees with large forests still made the tuning process moderately time-consuming.

**Model Performance & Interpretation**

**A screenshot of a graph

Description automatically generated**

Figure – 7: Classification Report of Random Forest on Test Data

The Figure – 7 shows that RF achieved a balanced accuracy of 0.743 and an overall test accuracy of 0.85. Based on the classification report in Figure – 7, the model performed exceptionally well in recognizing Jogging and Walking, with F1-scores of 0.97 and 0.88, respectively. However, it struggled with Downstairs (F1-score: 0.41) and Upstairs (F1-score: 0.61), suggesting that these movements were harder for the model to differentiate. Notably, the model overpredicted common activities and underperformed on less frequent actions. While Random Forest provided strong general performance and benefited from its robustness to multicollinearity and overfitting, its reduced recall on certain classes indicates it might not be the best standalone choice for imbalanced activity classification.

## K-Nearest Neighbors (KNN)

**Hyperparameter Tuning**

Hyperparameters for KNN were optimized using GridSearchCV with 5-fold cross-validation. The search was conducted over the following parameter grid:

A screenshot of a computer code

Description automatically generated

Figure – 8: Hyperparameter Tunning for KNN

* n\_neighbors: Specifies the number of neighbors to consider. Fewer neighbors (e.g., 3) can capture local patterns better but may be more affected by noise, while larger values smooth the decision boundary.
* weights: Determines whether all neighbors contribute equally (uniform) or closer neighbors contribute more (distance).
* metric: Defines the distance function used to measure similarity between instances.

The best combination found was:

* 'metric': 'euclidean'
* 'n\_neighbors': 3
* 'weights': 'uniform'

**Training Experience**

KNN had the fastest training time among all the models. This is because KNN does not involve an explicit training phase, it simply stores the data. All computations happen at prediction time, making the "training" step extremely lightweight compared to ensemble models like GBM and RF.

**Model Performance & Interpretation**

**A screenshot of a graph

Description automatically generated**

Figure – 9: Classification Report of KNN on Test Data

The Figure – 9 shows that KNN model achieved a balanced accuracy of 0.754 and a test accuracy of 0.81. It showed strong performance on classes such as Jogging, Sitting, and Walking, with F1-scores above 0.85. However, it struggled to distinguish more subtle or overlapping activities like Downstairs and Upstairs, which had F1-scores of 0.48 and 0.50, respectively. These results highlight both the simplicity and limitations of KNN. While it captures clear class boundaries well, it can be sensitive to noisy or highly correlated features, and its performance tends to drop when class boundaries are not well-defined or the dataset has imbalanced class distributions.

## Artificial Neural Network (ANN)

**Model Architecture & Training**

To ensure consistency and reproducibility, random seeds were set for NumPy, Python, and TensorFlow. The final model architecture was designed as a deep feedforward neural network using Keras's Sequential API. The network consists of:

* Four hidden layers with 500, 300, 100, and 45 neurons respectively, each using the ReLU activation function to introduce non-linearity.
* Batch Normalization layers after each dense layer to stabilize and accelerate training by normalizing the inputs to each layer.
* Dropout layers with rates of 0.3 and 0.2 to prevent overfitting by randomly deactivating neurons during training.
* A softmax output layer with a number of neurons equal to the number of activity classes (num\_classes), allowing the model to output a probability distribution over classes.

Weight initialization strategies were chosen to support deeper networks:

* He Normal for hidden layers (ideal for ReLU).
* Glorot Uniform for the softmax output layer.

The model was compiled using:

* Adam optimizer with a learning rate of 0.001 for adaptive gradient updates.
* Categorical crossentropy loss, suitable for multi-class classification with one-hot encoded labels.
* Accuracy as the evaluation metric.

To prevent overfitting and save training time, EarlyStopping was used to halt training if the validation loss did not improve for 10 consecutive epochs. Training was performed for up to 100 epochs (but will stop earlier than that as EarlyStopping is used) with a batch size of 30 and 20% of the training data reserved for validation.

**Training Experience**

Compared to other models, training the ANN involved significantly more computational overhead due to its deeper architecture and the iterative nature of backpropagation. In terms of training time, it was longer than KNN but faster than Random Forest and GBM, striking a middle ground in computational cost.

The training process was also highly sensitive to initial parameters and random seeds. To ensure consistent and reliable results, multiple experiments were conducted with different seeds. Finding the best-performing configuration required extensive experimentation, as neural networks lack clear heuristics for determining optimal settings, which means the effectiveness of a configuration can only be assessed through actual training and validation.

Additionally, the tuning process revealed that simplifying the architecture (fewer hidden layers) and removing non-significant features led to a model that not only performed better but also generalized more effectively on unseen data. While the process was more time-consuming compared to classical ML models, early stopping and proper weight initialization helped keep the training fast, stable, and efficient overall.

**Model Performance & Interpretation**

**A screenshot of a graph

Description automatically generated**

Figure – 10: Classification Report of ANN on Test Data

The Figure – 10 shows that ANN achieved a balanced accuracy of 0.808 and an overall test accuracy of 0.85. The model demonstrated excellent generalization, especially for common activities like Jogging (F1-score: 0.96) and Walking (F1-score: 0.88). It also performed well on Sitting (F1-score: 0.80), despite its small class size. ANN outperformed other models in terms of consistency across classes. However, it still showed some challenges with classifying Upstairs and Downstairs, where overlapping patterns or underrepresentation might have led to moderate recall scores. Overall, the ANN proved to be the most effective model, balancing high accuracy with robust performance across nearly all classes.

# Section – 5: Discussion of Results and Best Model Selection

**A graph of different colored bars

Description automatically generated**

Figure – 11: Comparison of Model Performance Across Evaluation Metrics

Model performance was evaluated using Balanced Accuracy, Standard Accuracy on the Test Set, and Accuracy on the Kaggle Dataset (48%). The Figure – 11 shows that ANN model demonstrated the best overall performance, achieving a balanced accuracy of 80.8%, along with the highest test set accuracy (85.3%) and Kaggle leaderboard score (90.8%). This result reflects ANN’s strength in capturing complex, non-linear patterns in high-dimensional data. Although the GBM model had a slightly higher test accuracy (86.4%) than ANN, its balanced accuracy was lower at 78.3%, and it underperformed on the Kaggle set (87.4%), indicating biased to dominant classes. Random Forest also showed good generalization but fell slightly behind GBM. KNN, while more straightforward, achieved respectable results but lagged behind the other models on both the test set and the Kaggle dataset.

Importantly, Balanced Accuracy was included as an evaluation metric because the dataset is imbalanced—some activity classes are underrepresented. This metric provides a more reliable indication of how well each model performs across all classes, not just the majority ones, ensuring that performance isn’t biased toward dominant categories.

Considering all metrics and performance across datasets, ANN stands out as the most effective and best performing model among those trained in this project.

Note: Among all the submissions made on Kaggle, the two best-performing output files were generated by Artificial Neural Network (ANN) models using the same architecture and hyperparameters. The key difference is that the first model used EarlyStopping (resulting in the best overall performance and it is considered as the best model), while the second model did not use EarlyStopping but gave the highest accuracy score on the Kaggle dataset. The first model achieved a Kaggle accuracy of 0.90834, a test accuracy of 0.8532, and a test balanced accuracy of 0.8076, demonstrating strong and consistent performance. In contrast, the second model achieved a slightly higher Kaggle accuracy of 0.92815, but its test accuracy dropped to 0.8252, with a test balanced accuracy of 0.755, suggesting potential overfitting to the Kaggle dataset. Based on these results, the first ANN model is considered more reliable and better generalized across different datasets.

# Future Improvement

In the current work, there are several opportunities for improvement. First, the dataset is imbalanced, with activities like Walking (2,452 samples) and Jogging (1,951 samples) being far more represented than Sitting (321) and Standing (278). Acquiring more data for underrepresented classes would help improve model fairness and generalization. Although an LSTM model was implemented, it performed poorly due to the noisy nature of the raw signal data. If the signals are smoothed using filters like moving average, which replaces each data point with the average of its neighboring values or segmented more consistently, LSTM could be able to capture the differences between ectivities better. Dimensionality reduction using PCA was also explored in this project but did not improve model performance with the current dataset. However, with a more refined and denoised dataset, such techniques could become more effective in reducing noise and computation without sacrificing accuracy.

# Appendix

All the datasets, code files, and statistical analysis results used in this project are provided in a GitHub repository. The repository includes the cleaned and engineered data files used for model training and testing, all Jupyter notebooks developed for feature engineering, exploratory data analysis (EDA), model development, and model evaluation. It also contains the outputs and visualizations from the EDA process, such as normality test results, Kruskal-Wallis significance tests, and the correlation heatmap.

A link to the GitHub repository is as follow:

<https://github.com/thetkhinelin25/Machine-Learning-and-Data-Mining-Project.git>