Course Work

Application of Machine Learning Models to Real Word Data

Disclosure

In this project, some chunks of code and error correction are accomplished with the aid of artificial intelligence. Artificial intelligence (AI) techniques have been utilized to optimize the coding process and effectively address faults. Furthermore, this project used public code repositories, especially well-known platforms such as GitHub and W3School, medium, in order to seek support and access resources from the wider programming community.

Module Code: 021DATSC

Level: 7

Credit Rating: 20

Programme: MSc Data Science

Team Name: Sandbox
Student Name: Zainab Mahmood, SaiKirti Pisupati

Table of content

Table of Contents

1 Methodological Approach

1.1 Approaches to Solve the Problem

1.1.1 Understand the Problem Domain

Exploratory data analysis was used in the first step to understand the datasets, which included a timeseries and related metadata. This involved determining the data distribution and identifying missing values using basic statistical techniques box plot and pair scatter plots. Furthermore, correlation plots were produced in order to better understand the links between the variables and how they influenced the target feature, which is activity. In order to assess linear correlations between variables, a correlation matrix was also created. Afterwards, unsupervised learning methods including PCA, T-SNE, and UMPA were used to analyse the data more thoroughly and find clusters. The figure below shows the flow of exploratory data analysis that was adopted.

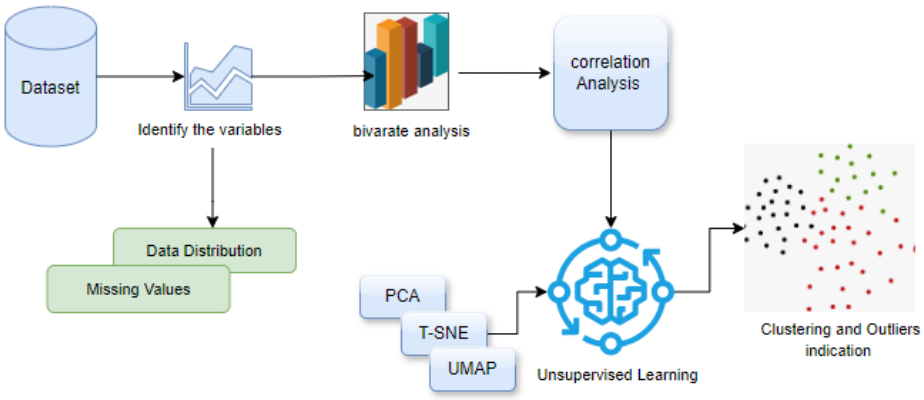


Figure 1: The initial dataset was examined in four stages, moving from basic analysis to unsupervised machine learning algorithms, in order to determine the data distribution, potential clusters, and feature correlation.

1.1.2 Machine Learning Model Training

The eleven different machine learning models were applied to solve the human activity multi-class classification problem. The models were fully trained in a series of three phases. In the first phase, seven models such as artificial neural networks, logistic regression, support vector machines, random forests, gradient boosting, K-nearest neighbours, decision tree, and grid search algorithms were used to analyse and understand where the model performed better with different parameters. A confusion matrix and model summary were also used. The models were tested on two different, unseen datasets to verify the model accuracy. The flow of first phase model application is defined in figure 2.

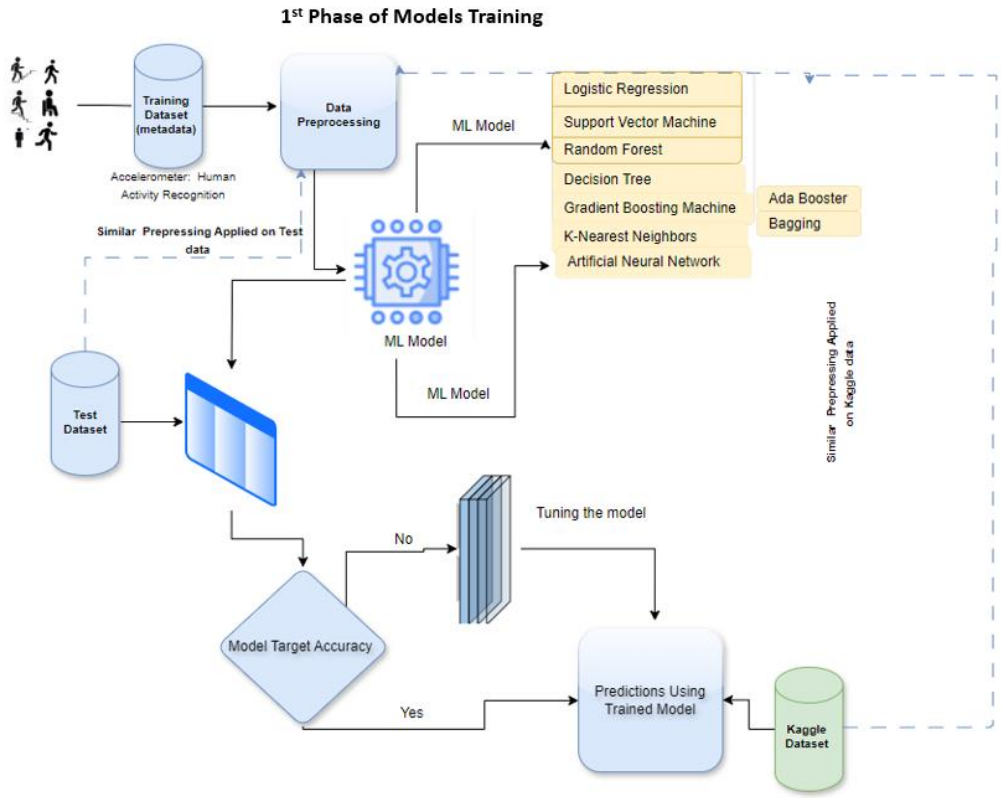


Figure 2: Supervised and ensemble learning models were trained to accurately classifying six different human activities—stair climbing, stair descending, walking, jogging, and standing. Trained models' accuracy was tested om unseen dataset.

The timeseries dataset from the accelerometer sensor of personal digital devices was used in the second phase of the model training process. Convolutional neural networks (CNNs), LSTM from recurrent neural networks, were the models used on the time series dataset, the figure 3 illustrate the flow of training model. Several parameters were applied to the multi-layer neural network in order to fine-tune it. We tested the model using two different timeseries that had similar features.

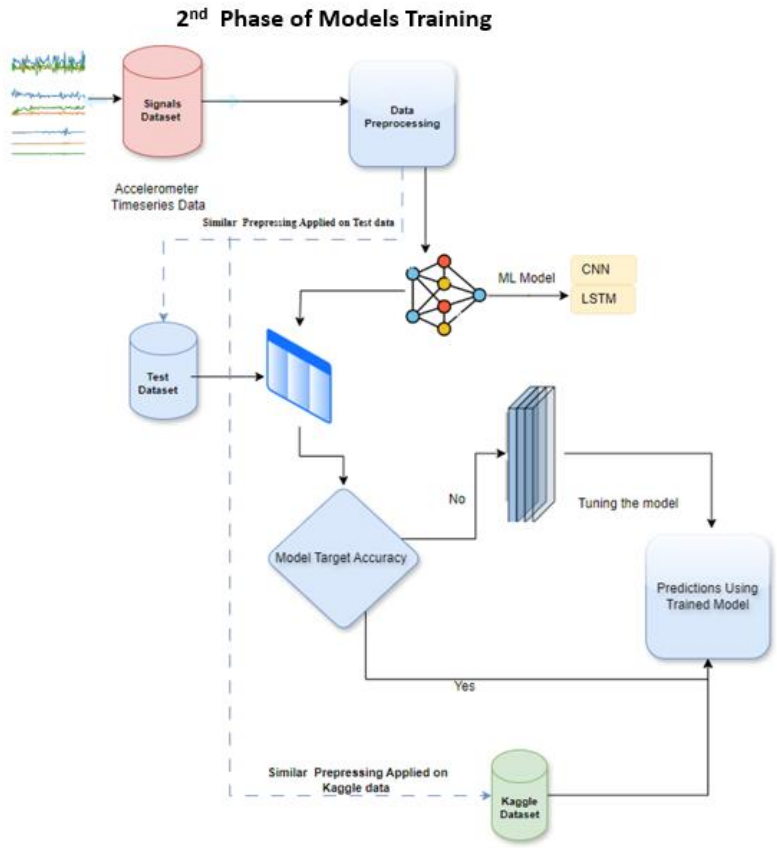


Figure 3: Timeseries from accelerometer of personal digital devices which divided in 10-second snippet was used to train the neural and deep learning ML models to classify the six human activities and applied on unseen data (test and Kaggle).

When the model's performance did not improve to the required extent in the third phase of training, 29 new features were extracted in three iterations. Timeseries datasets were used to extract the additional features, which were then combined with the original metadata file. Similar feature extraction functions were used for the "signal-test" and "signal-kaggle" files since the machine learning algorithm requires that the model be tested using the same features that it was trained on. In the first cycle of this phase GBM, CNN, and RF the models were trained on updated metadata. In the following cycle, logistic, RF, CNN, RF, and GBM were applied to 19 filtered features from the correlation matrix with the expectation that by eliminating all corelated variables, the model's accuracy would increase. To improve the model's ability to generalise on unobserved data, the forward and backward feature selection techniques were applied during the last cycle of model training. The figure 4 illustrate the flow of each cycle adopted to train the model in last phase of model training.

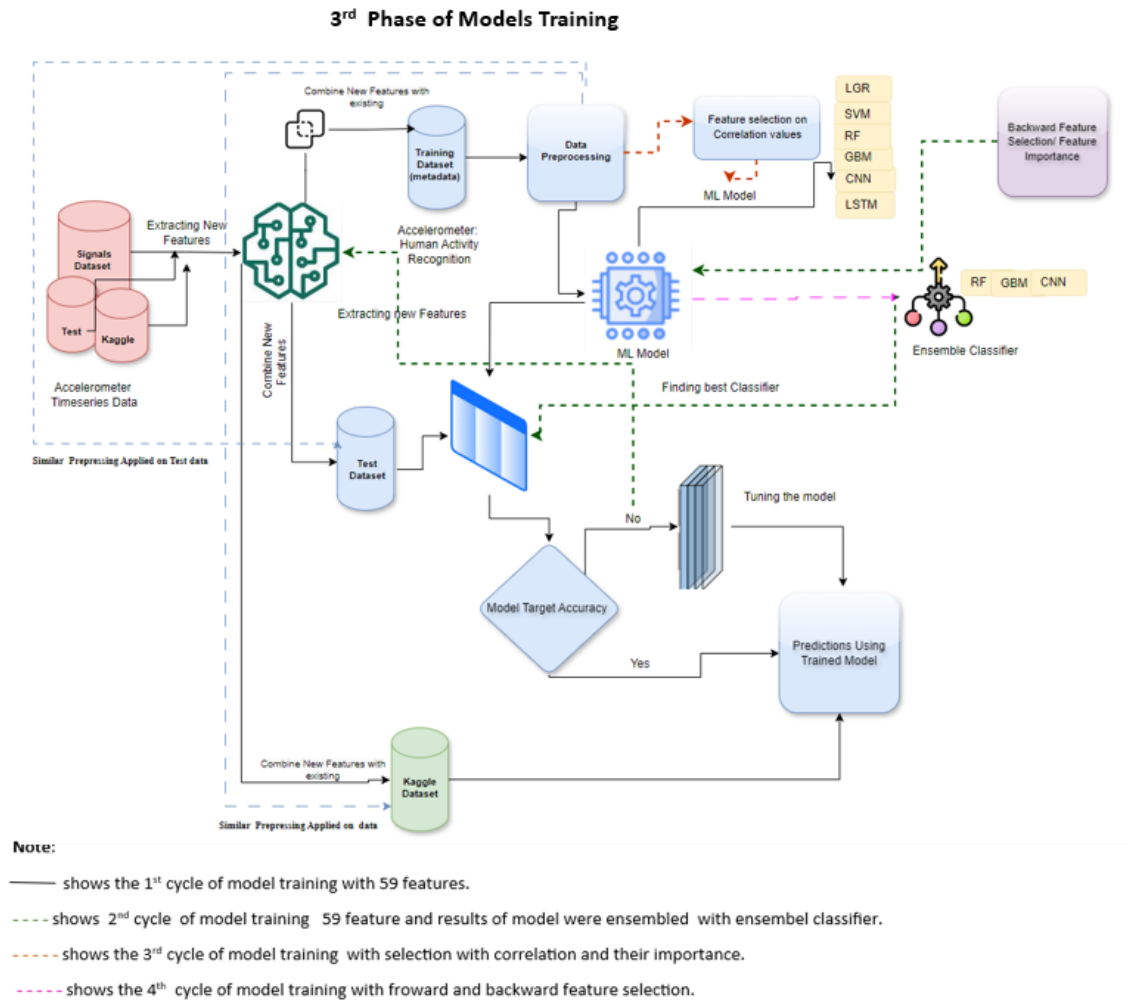


Figure 4: Extracted 29 new features from signals, GBM, RF and two deep learning (CNN, and LSTM) ML models were trained to achieve model high accuracy classifying the six human activities on unseen data (test data and Kaggle). Ensemble classifier used to reduce the misclassification rate of weak classifier by aggregating multiple classifiers.

2 Exploratory Data Analysis

2.1 Data Preprocessing

2.1.1 Data Imbalance

The training and test datasets were unbalanced, as Figure 5 below illustrates. Given that it is a multiclass classification problem, the six classes in the activity were plotted against the dataset. With 2452 counts, walking is the most frequent class, followed by jogging with 1951 counts. The number of values for the upstairs (702), downstairs (606), sitting (321), and standing (278) classes in the training dataset is relatively low. The test data set showed a similar trend with a different value count, with walking (768) and jogging (689) being the first and second dominant classes, respectively. The remaining classes have relatively low value counts in the data set, with the lowest value count being 22.

To deal the data imbalance, under sampling, smooth and parameter weight parameter were used during the model training. Some models like random forest, gradient boosting decision tree and CNN are not sensitive

to imbalanced data set. In first phase model training imbalanced datasets converted into balanced datasets. In last phase of model training 'Precision/'Recall' performance measure was used for imbalanced dataset to analyse the performance.

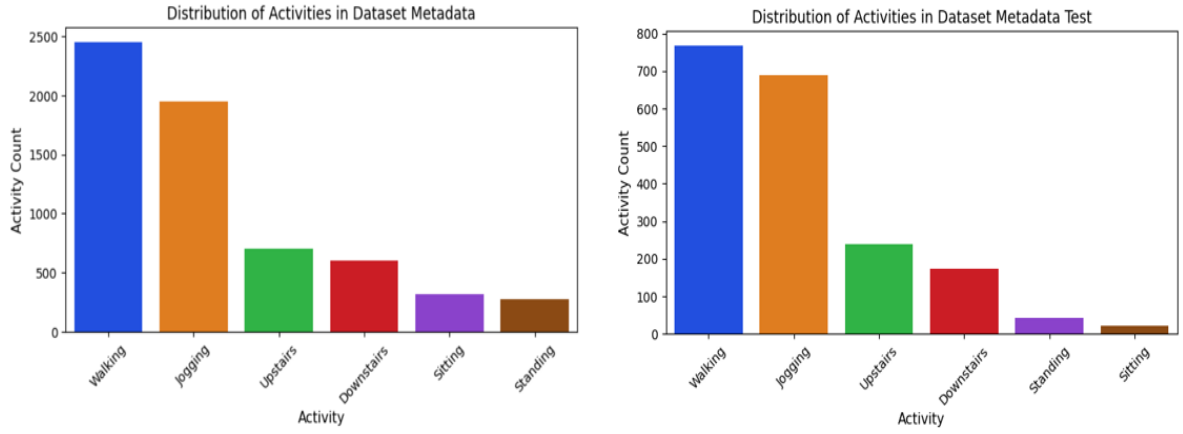


Figure 5: Walking and jogging class data are more commonly preset in datasets, indicating an imbalance in the distribution of data across training and test datasets. The data ratio between standing, seated and other classes is noticeably low, which may result in inaccurate model predictions.

2.1.2 Data Normalization

The "sklearn preprocessing" library was used to normalise the dataset. To standardise the dataset, the library calculated mean and standard deviation. Each feature's mean is subtracted, and the results are divided by the standard deviation. Preprocessing ensures that features are on the same scale, which may enhance algorithm performance in models such as random forest, logistic regression, GBM, CNN, ANN, and LSTM.

2.1.3 Feature Extractions with PCA

PC2 captures 16.88% of the variance, while PC1 captures 36.79%. This indicates that more than 53% of the variance in the data is captured by the first two main components combined. This implies that the two major components shown in figure 7 may adequately describe the data.

Most likely, the data points are spread out all around the origin, extending along PCs 1 and 2. Since jogging and walking are distinct along PC1, it is possible that PC1 caught a component of exercise intensity, with jogging exhibiting and walking a higher intensity. Since PC2 separates activities, such as standing and sitting on the upper and lower stairs, it suggests that PC2 could record components of vertical movement.

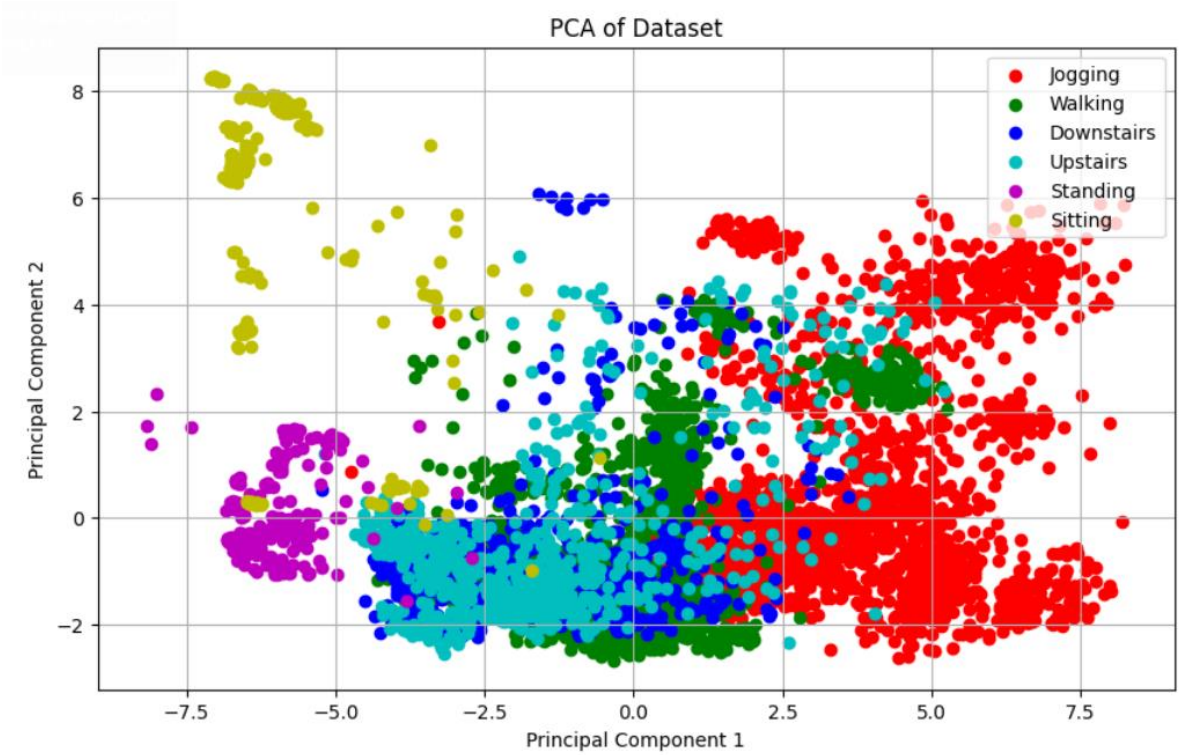


Figure 6: The first two PCs capture 53% variance of data and well-represented by these two principal components. PC1 captured aspect of activity intensity, with jogging and walking. PC2 captured vertical movement standing, upstairs downstairs and sitting.

2.1.4 Box Plots Analysis of Acceleration and Variance Metrics by Activity

Only four boxes are displayed in this box plot, which was made for every variable that is attached to the report's end. These data points are drawn outside of the whiskers, indicating that there are many outliers in the variables related to the y and z axes. A large range of values is displayed by the median value of all activities across all variables. Analysing the values from the data set shows varying values of "y-axis__root_mean_square," "y-axis_maximum," "y_axis_mean," and "y-axis absolute maximum," where some values are noticeably higher than others. Accelerometer data outliers can be challenging, particularly since extreme values may represent genuine movements or anomalies in the data collection process. Unusual activity, abrupt motions, or even sensor disruptions could be represented by outliers. The PCA algorithm was used to calculate the outliers and discussed in following part of report.

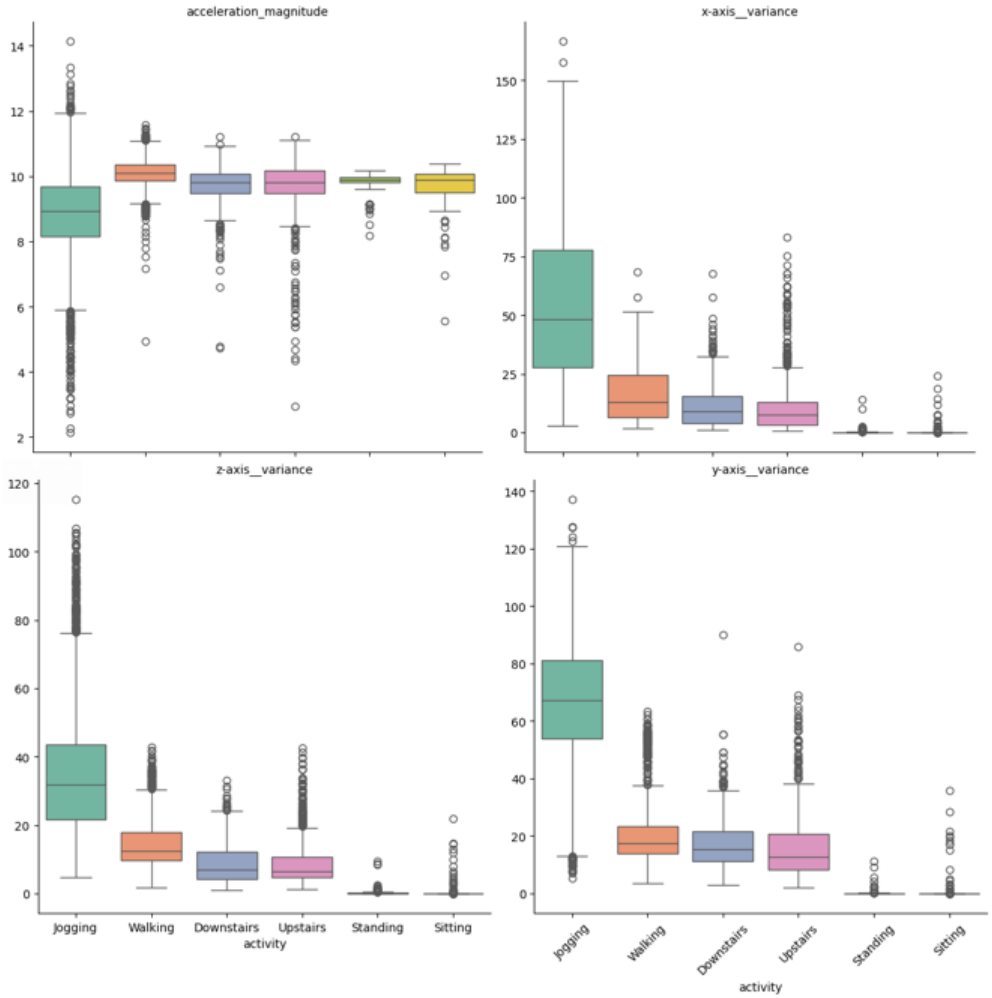


Figure 7:Box plots of variance and acceleration magnitude illustrate various datapoints outside the quantile Q1 and Q3 range which implies as a potential outlier in dataset. The large of datapoints outside whiskers may represent genuine movements with extreme values.

2.1.5 Outlier dictation with PCA

The Mahalanobis distance for PCs 1 and 2 is the robust method for calculating the outliers. To analyse the computed outliers, the threshold values of 4, 3.5, and 3 were used. There are only three outliers at the threshold value of 4, compared to a few at value 3.5 and several significant outliers at value 3. The identified outliers on the left side of the plots indicate the sitting class and grouped towards the PC2 between values 8 and 7, according to a comparison of Figure 7 with datapoints classified by PC1 and PC1 which demonstrate that the data points that have been detected are not true outliers. Comparably, in the PC1 vs. PC2 plot, the points found on the right side of the plot are classified as walking since they are closely associated with the walking class. It may therefore be the case that they are not true outliers, but rather a means of identifying either these or other real or distinct types of movement.

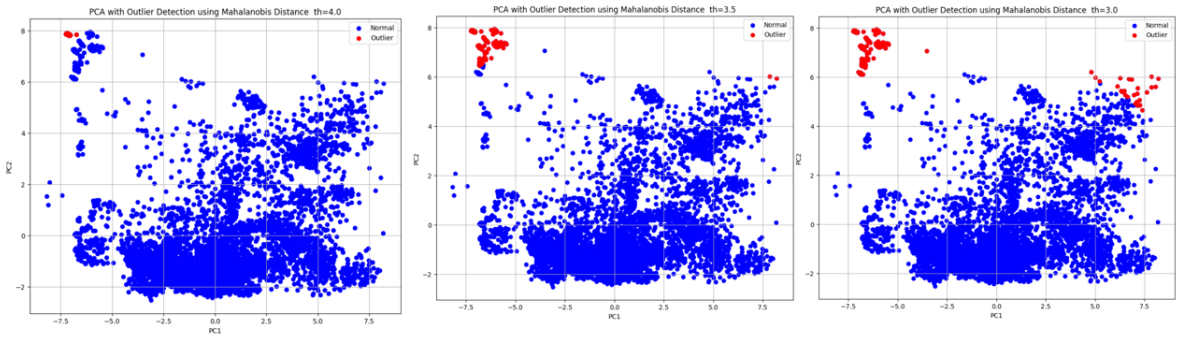


Figure 8: Identifying the outliers using the mahalanobis distance for PC1 and PC2 for threshold 3,3.5 and 4. A stricter threshold (4) identifies very few outliers, while a more lenient threshold (3) identifies many outliers.

2.1.6 Features Extraction from Timeseries Dataset

The variables below were extracted from a timeseries dataset and their contribution to the classification of human activity was taken into consideration while making the extraction choice. These features have been applied to a number of researches [1,2] that measure how well models perform in classifying human activities.

Feature	**How it has impact on activity**
IQR (Calculated for x, y, z)	It can aid in identifying between activities that have varying levels of movement intensity. Jogging, for instance, may have a higher IQR than sitting because of the greater variability in acceleration values.
Skewness (Calculated for x, y, z)	Activities with more balanced movement patterns, like walking may have a lower skew that is closer to zero, but activities with more frequent quick movements, like jogging upstairs, may have a positive skew.
Kurtosis (Calculated for x, y, z)	Activities with sharp peaks in acceleration (jumping) have higher kurtosis compared to activities with smoother distributions
Total Power (Calculated for x, y, z)	Higher movement like jogging, are probably going to have a greater total power than lower intensity like sitting.
Resultant Magnitude (Calculated for x, y, z)	In the absence of direction information, it can be helpful for capturing the general intensity of the movement.
Spectral Centroid	Activities that primarily include low-frequency motions, like walking, may have a lower spectral centroid than exercises that primarily involve higher-frequency movements, like jogging.
Energy (Calculated for x, y, z)	Measure how total energy is distributed for walking, jogging.
Energy (PSD mean)	Provides a general measure of how energy is distributed over the frequency range.
Spectral Entropy (Calculated for x, y, z)	Activities with more predictable, repetitive movements (e.g., walking) might have lower spectral entropy compared to activities with more irregular movements (e.g., upstairs).
Dominant Frequency	It can be especially useful for differentiating between tasks that have different rhythmic rhythms. walking typically exhibits a dominant frequency around 2 Hz (corresponding to two steps per second), while jogging might have a higher dominant frequency closer to 3 Hz or even higher depending on the jogging speed.
Resultant Direction	This can help differentiate activities that require for particular movement orientations. Activities that involve vertical movements, like going upstairs and downwards, could provide different resulting orientations, but walking or jogging might produce a direction that is more forward-looking.

2.2 Correlation of Activity with Other Variables

The relationship between activity classes and other variables is shown in the correlation grid plot in figure 6. There is a negative correlation between jogging, walking, downstairs, and upstairs and the z-axis absolute

maximum, root mean square, and standard deviation. This indicates that when the data points are more dispersed (greater standard deviation), the intensity of activities such as walking, running, and living upstairs and downstairs is lower (i.e., there are fewer extreme values on the z-axis). Walking and jogging have a significant negative relationship with the y-axis variance and y-axis standard deviation; in contrast, all other activities have a slight negative correlation. Jogging and walking appear to be more responsive to changes in the y-axis than do activities downstairs and upstairs. This may suggest that, in contrast to activities like going up stairs, these activities take place in surroundings with less vertical movement.

While walking has a negative correlation with the x-axis mean, median, and total values, jogging has a tiny positive correlation. The marginally positive correlation values indicate that jogging is more common in locations where the x-axis average is higher. This may have to do with things like location or distance. The previously mentioned pattern is seen in the positive to negative correlation between the upstairs and downstairs, respectively. The nature of these activities may account for the downstairs' positive correlation and the upstairs' negative correlation with the x-axis. Moving downstairs could involve a greater displacement in the horizontal direction (positive x-axis) than moving upstairs, which is more vertical.

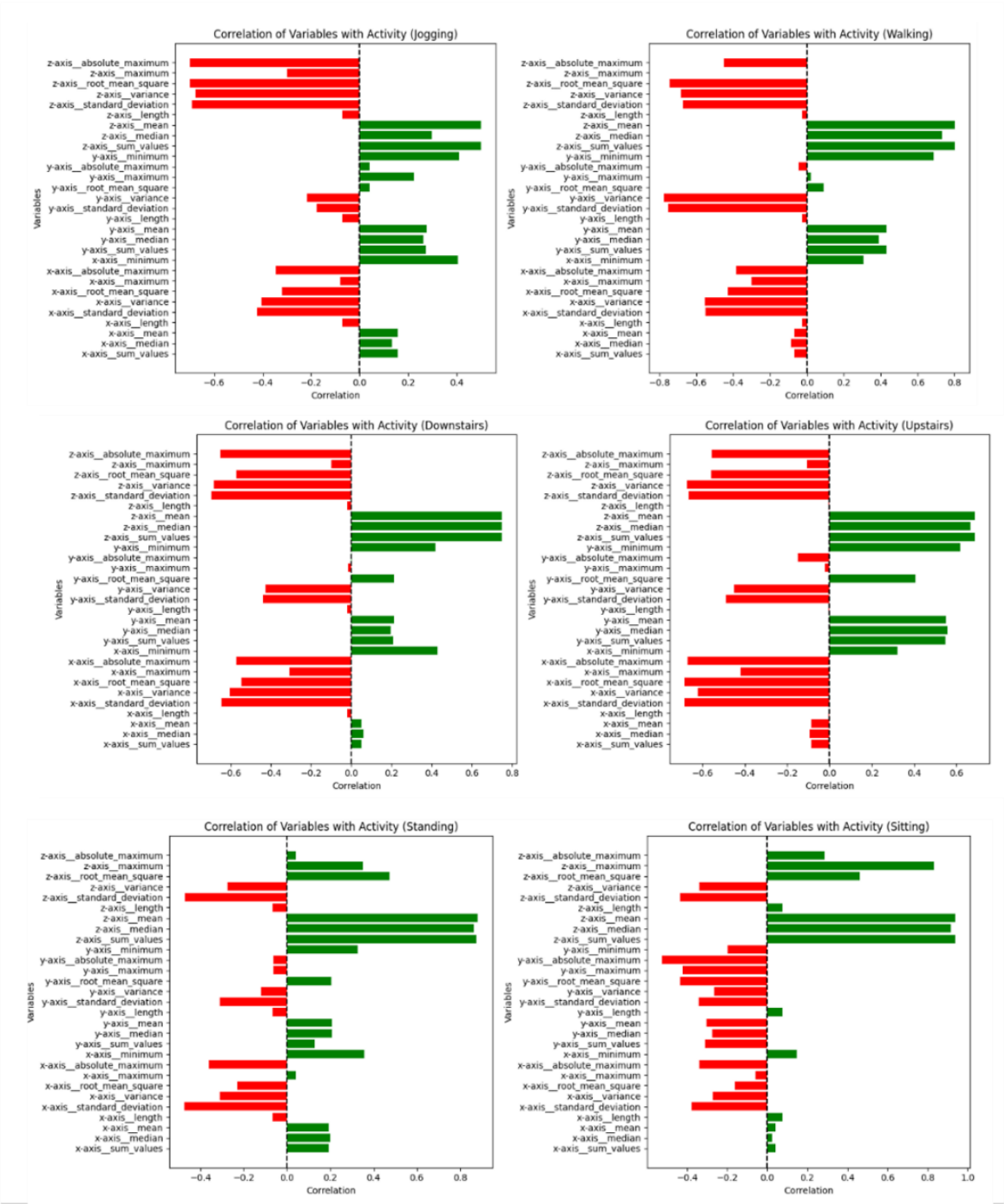


Figure 9: Grid plot illustrates how activity classes correlate with various movement features. Activities with high vertical movement (downstairs, upstairs) show a negative correlation with z-axis statistics (max, RMS, standard deviation). Walking and jogging exhibit negative correlation with y-axis variance and standard deviation indicating sensitivity to variations in vertical movement compared to downstairs.

2.3 Activity, the Predictive Variable's Behaviour

2.3.1 Distribution of Activities by User

Predictive variable activity was plotted against the user to determine the most popular activity performed by users in order to further analyse the data. It has been noted that the most frequent activity among users is waking, with jogging coming in second, and sitting being the least common activity among all users. One

further illustration of the unbalanced data is provided by this plot. Data was collected from the users' individual digital devices and data collected show that when users perform physical activities like walking or jogging, data collection happened. This suggests that the gadgets could be fitness trackers or made to capture the exercise data. It may also possible that while users were sitting, they were less likely to wear the devices or keep them active.

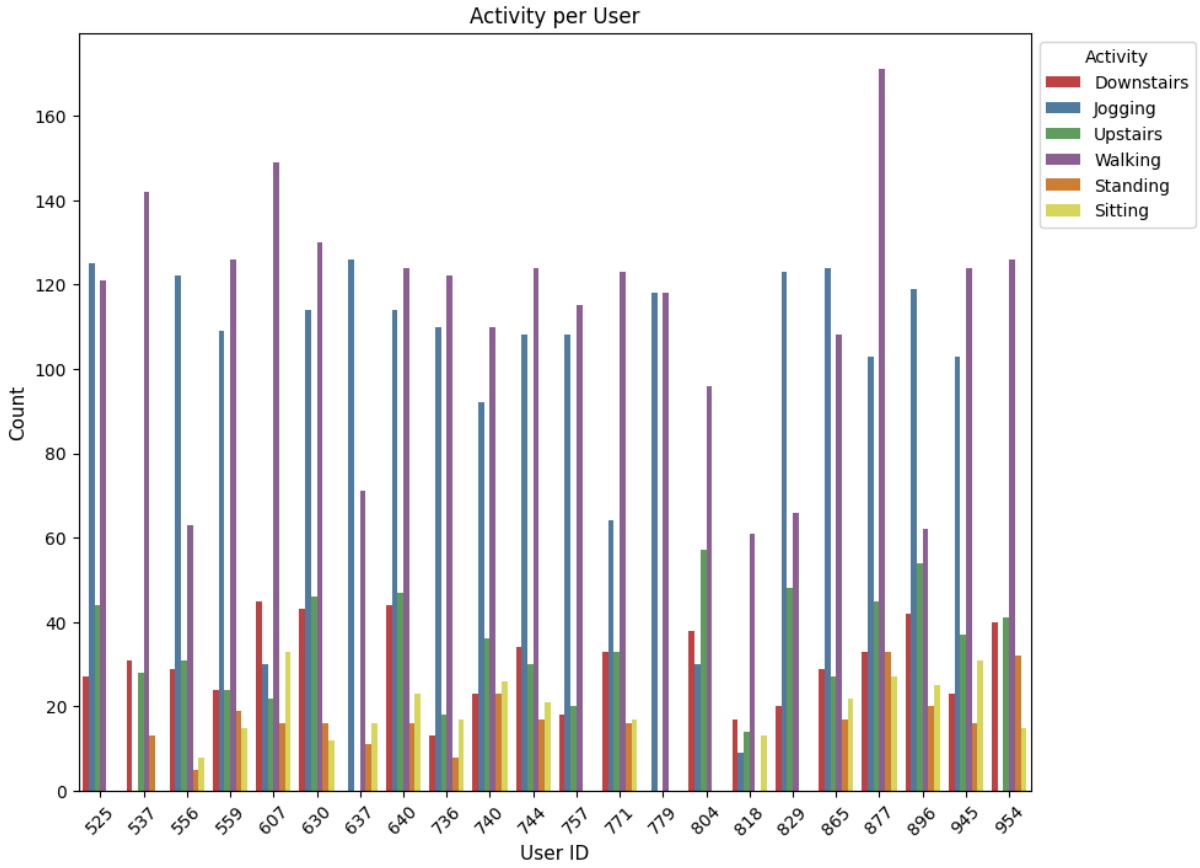


Figure 10: Activity class 'walking 'was the most popular activity, followed by jogging. The least frequent user actions include standing and sitting, which suggests that the data was collected when users were engaged in intense activities.

2.3.2 Change of Magnitude on Activities

Interesting observations about the relationship between acceleration magnitude and human activity density are made from this plot, which also offers important information about how users move. The narrow and high peak for standing at 9–10 magnitude and 2.5 density points to quick, powerful accelerations. This might be the result of a sudden change in weight distribution while standing still or the first movement made after getting up from a sitting position. One possible explanation for the two sitting peaks at density 1 and spread between 8 and 11 magnitudes could be a slight tilt in the body or fidget while seated.

Jogging's long tail on the x-axis (1–15 magnitude) with low density (0.5) shows a large range of acceleration values occurring infrequently. The reasons for this could be uneven ground, changes in the intensity of jogging, or leg swing. Walking's magnitude distribution (8–11) and density (up to 11) are higher than jogging's,

indicating a more targeted range of acceleration and a larger frequency of identical movements throughout each step.

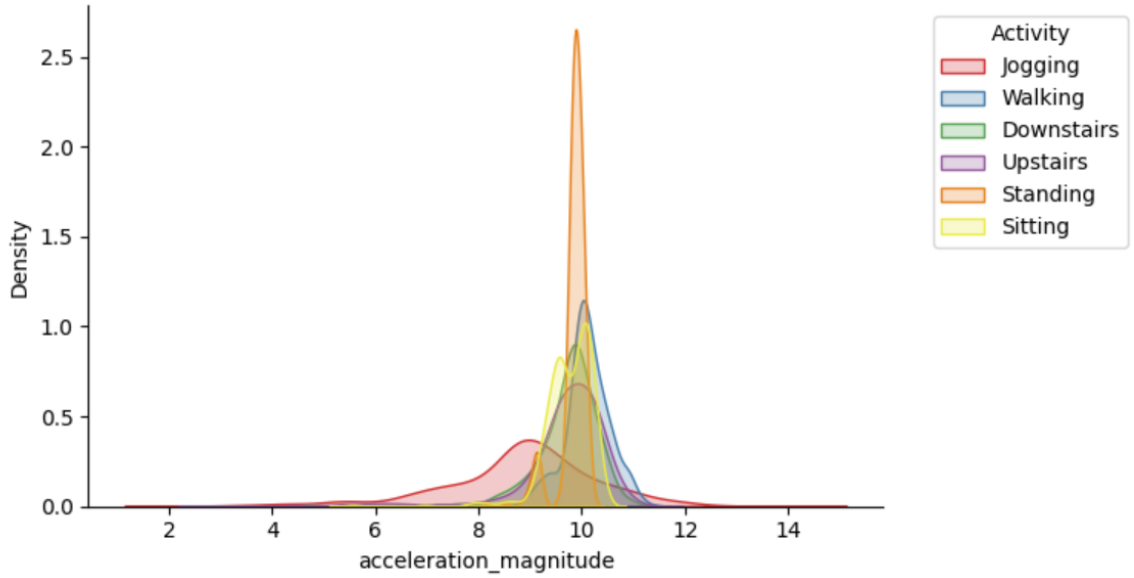


Figure 11: Peaks in sitting and standing behaviours show different patterns of acceleration, and jogging shows fluctuations over a broad range. Walking reveals a targeted acceleration range at a higher frequency, providing insight into the movement patterns of users during the various activities.

Figure 12 shows more detail at the peaks and changes in the density of dynamic activities like walking and jogging upstairs and downstairs and static activities like sitting and standing. The downstairs has a higher peak than the upstairs. Due to the user's repeated vertical and horizontal accelerations from bouncing, the jogging has a lengthy tail of high magnitude on all axes with consistent density of data points.

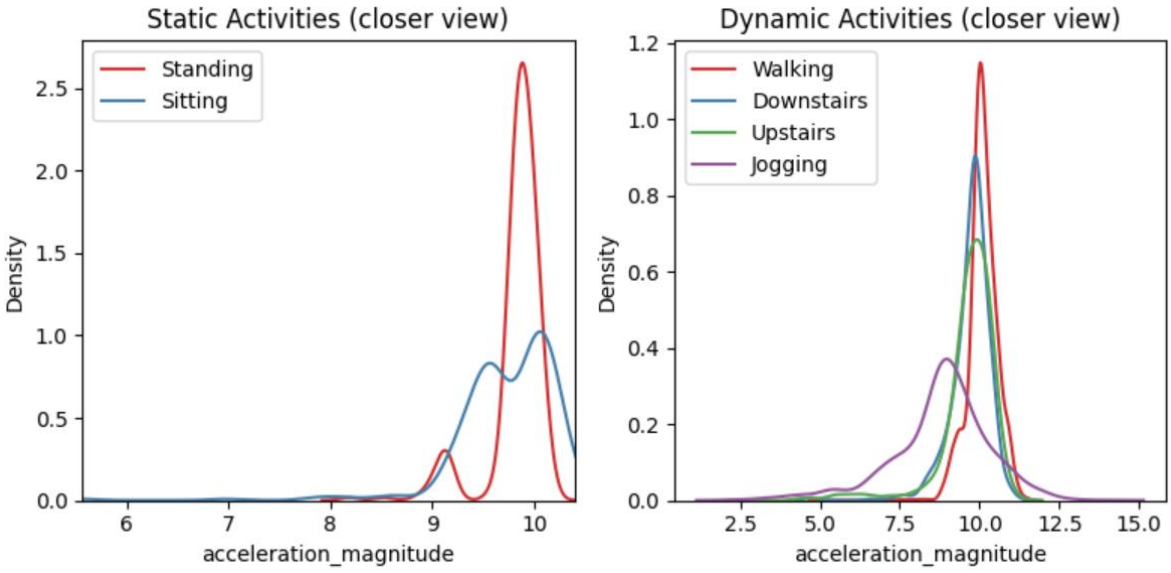


Figure 12: Static activity like standing and sitting have brief peaks which represent sudden change in position while balancing the position or weight. Walking illustrated by smooth peak which implies that walking activity is repetitive movements. Downstairs have higher peaks than upstairs with high density.

2.4 Data Clustering Using T-SNE Model

In order to analyse the class grouping, PCA was used first. This revealed clearly defined clusters with some overlap. Using different perplexity levels, the T-SNE model was used to validate these clusters. Some grouping was seen at a perplexity of 2, but there was also a lot of scattered data, which was especially apparent in activities like walking. Jogging showed more distinct clusters, suggesting that the t-SNE captured some underlying similarities in the data, especially for jogging. separating between activities such as walking and standing was difficult, even with mild confusion. Data points decreased when the complexity was increased to 5, however meaningful clustering patterns were still difficult to find. This pattern continued at perplexities of 10 and 15, indicating that actions such as standing, walking, and going upstairs and downstairs have characteristics in common that make it more difficult t distinguish between them in low-dimensional T-SNE. The scattered distribution of data points and the absence of clear clustering leads to refined the perplexity parameter and check cluster with perplexity value, 1, 8, 12 however jogging and walking cluster become more clear with perplexity 1 with many other overlapping regions.

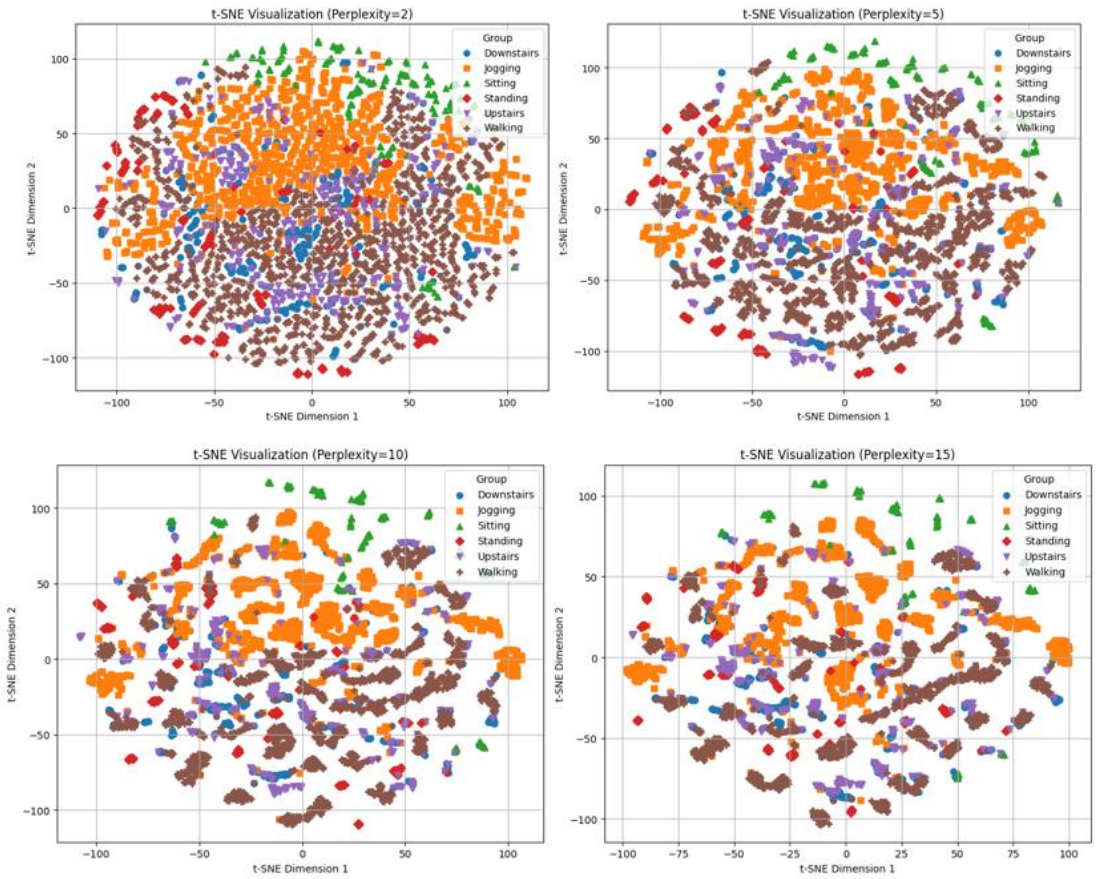


Figure 13: t-SNE captured some underlying similarities in the data, especially for jogging and walking, other datapoints are scattered around make it difficult to cluster the data into different classes.

2.5 Clustering Using UMAP Model

After T-SNE failed to validate the clustering observed in PCA, the UMAP model was used to validate the patterns of clustering in the dataset. The UMAP method identified clusters in the data by using parameters

like n_neighbors, min_dist, and metric='euclidean'. Surprisingly, the UMAP clustering supported the PCA results, especially when it came to differentiating between sitting and standing behaviours. Additionally, UMAP distinguished between walking and jogging groups, suggesting that these two activities have similar movement patterns. Walking, upstairs, and downstairs are likely to show overlapping features, like leg movement, which could make it more difficult to distinguish between them in the low-dimensional UMAP embedding. This shows that more customised features could be created in the future to enhance clustering efficiency and more accurately capture each aspect of these activities.

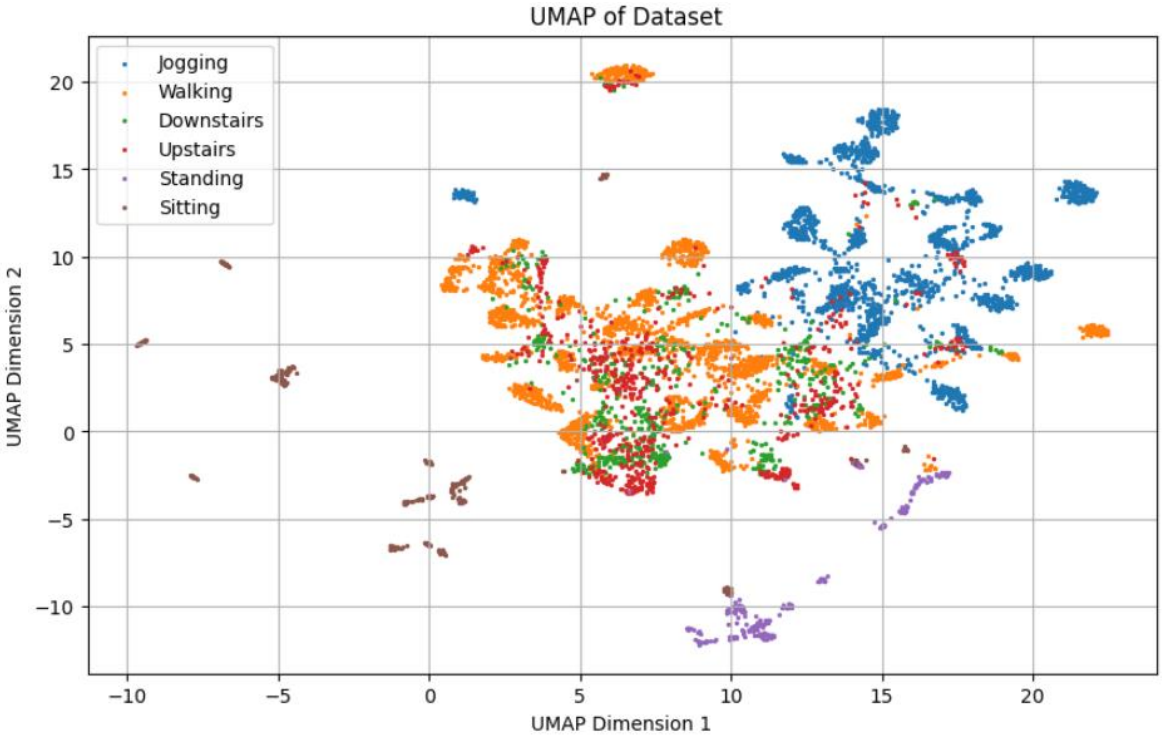


Figure 14: UMAP confirms the reported PCA clustering pattern with n_neighbors=40, min_dist=0.5, and metric='euclidean'. UMAP grouped walking, jogging, standing, and sitting together; however, the walking class overlapped with the activities of the downstairs and upstairs. Walking and jogging are the most frequent performed activities.

2.6 Signals for Activities

The x-axis accelerometer output shows some low peaks interspersed with high peaks when walking. Each walking steps forward acceleration during the heel strike and backward acceleration are represented by these peaks. Walking's y-axis signal is primarily downward and moves towards zero, with irregular large peaks. Higher peaks typically correspond to faster motion or more energetic steps. Walking steps per minute are correlated with peak frequency. While the z-axis displays the same peak pattern with time and the x-axis has more powerful peaks, jogging exhibits higher peaks along all axes. As the force of movement changes with each step above, stairs and downstairs have higher peaks; similarly, downstairs have several uneven peaks. It is clear that when one is seated, there is no movement and no energy expended, which means the axis will stay stable. Less peaks signify a shift in posture and help to balance the weight.

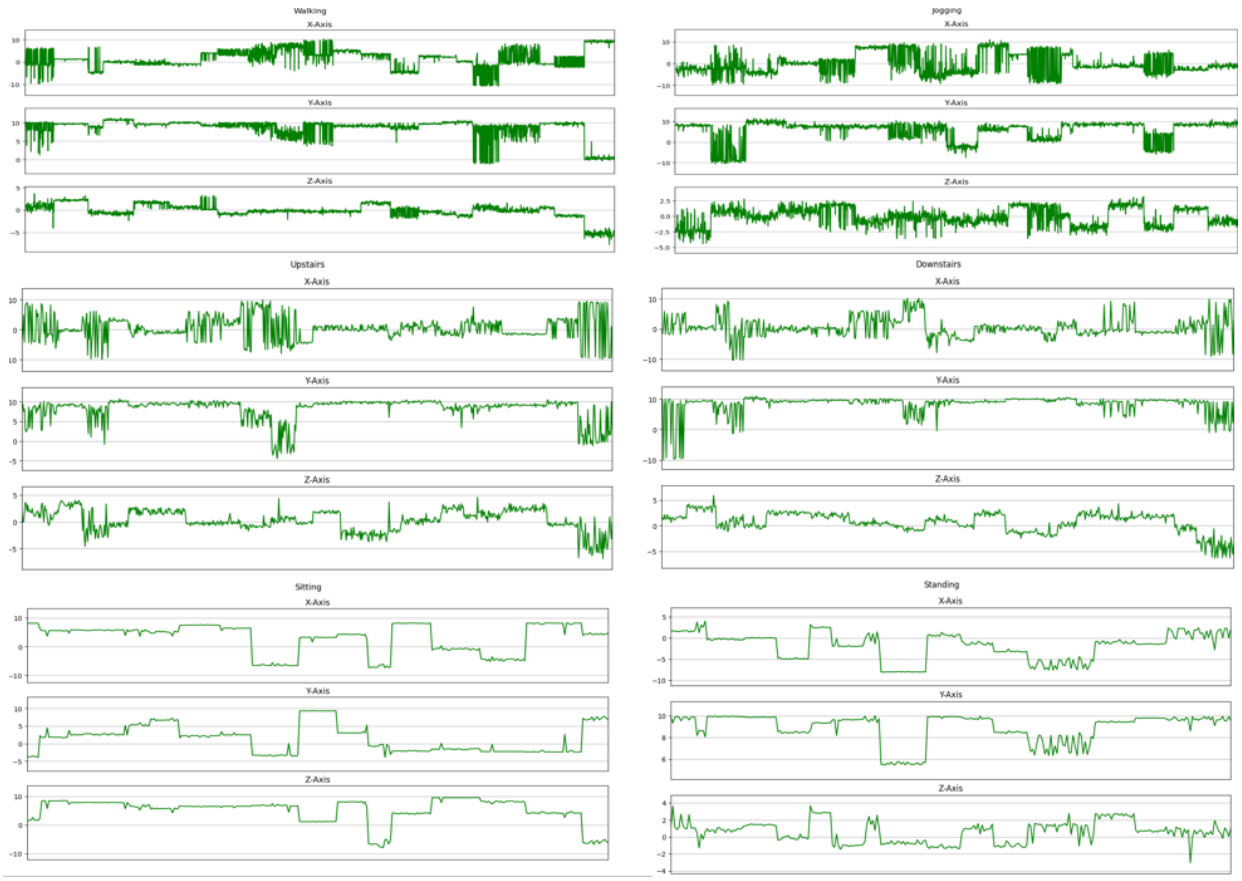


Figure 15: Walking shows repeating peaks with different heights based on pace on all three axes (X, Y, and Z). In comparison to walking, jogging exhibits greater and more frequent peaks on all axes, indicating faster and stronger hits. Values on X, Y, and Z remain rather steady while sitting with the potential for small peaks related to moving. With every step, upstairs and downstairs display high peaks on Z with downstairs having bigger impact peaks. Standing causes brief high intensity peaks on X, Y from posture changes with Z

3 Machine Learning Modelling

3.1 Phase -1: ML Model Training and Testing

3.1.1 Model Training Using Logistic Regression

Logistic Regression Model Training Parameters and Performance			
Training data used	80%	Validation data used	20%
Accuracy on Validation	0.7987	Accuracy on test data	0.7352
Hyperparameter Tuning: C: [2.5], 'penalty': ['l2'], class_weight='balanced'			cv=5
Accuracy on validation data	0.8438	Accuracy on test data	0.7942
Hyperparameter Tuning: C:[4], 'penalty': ['l2']			cv=5
Accuracy on validation data	0.8613	Accuracy on test data	0.7864
Hyperparameter Tuning: C: [2.5], 'penalty': ['l2'], class_weight='balanced'			cv=10
Accuracy on validation data	0.8423	Accuracy on test data	0.7957

3.1.2 Description of Models Training

Logistic regression was the first model used for current classification problem. A few parameters were changed, including the regularisation strength and type (l1 or l2). To resolve the imbalance in the training and test data, the class weight parameter was set to balanced. Additionally, used the 'ovr' multiclass argument to handle multiclass classification. Used confusion matrices for both test and validation data to compare the models' performances and evaluated them using a 5-fold cross-validation approach. Despite thorough parameter tuning, the best model performance was 84% on the training set and 79% on the unseen data indicating reasonable generalization to new data.

It is essential to evaluate precision and F1 scores for each class separately when assessing model performance. For example, we obtained the best accuracy (0.94) and F1-score (0.96) when identifying jogging activities, indicating exceptional performance with few false positives and negatives. On the other hand, for standing activities, the F1-score (0.75) indicates possible misclassifications with other activities even when precision (1) is ideal. Similarly, for sitting activities, there is potential for improvement in differentiating sitting from other activities, as indicated by the moderate performance the accuracy (0.56) and F1-score (0.69). Compared to jogging and walking, activities like downstairs and upstairs show poorer precision (0.31, 0.77) and F1-scores (0.33, 0.62), suggesting difficulties in correctly identifying these activities.

Although 84 downstairs activities (TP) were accurately classify by the model, it frequently misclassified these as upstairs activities (24), and walking activities (40). The model performed exceptionally well at classifying jogging activities (TP=387), with the majority of misclassifications occurring upstairs. It misclassified certain walking activity as downstairs (22), upstairs (29) and downstairs (29. All things considered, the model shows good accuracy when it comes to jogging; nevertheless, it still has to work on distinguishing between activities such as walking, sitting, and downstairs and upstairs. Since the model hasn't achieved the desired results, Support Vector Machines (SVMs), which might be more suitable for this dataset, have been explored as well as the creation of new features from the current data to more successfully distinguish between sitting and other actions was in consideration in next phase of model training.

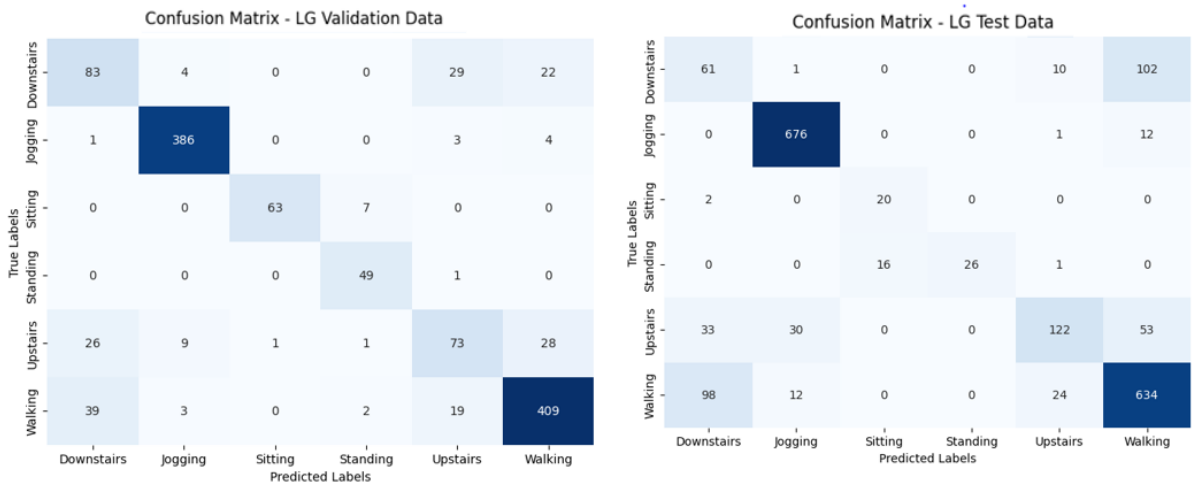


Figure 16: Model performance on validation and test data was compared using the confusion matrix. The model did well on training data, with an accuracy of 84%, and it was able to generalise to an unknown test with an accuracy of 80%.

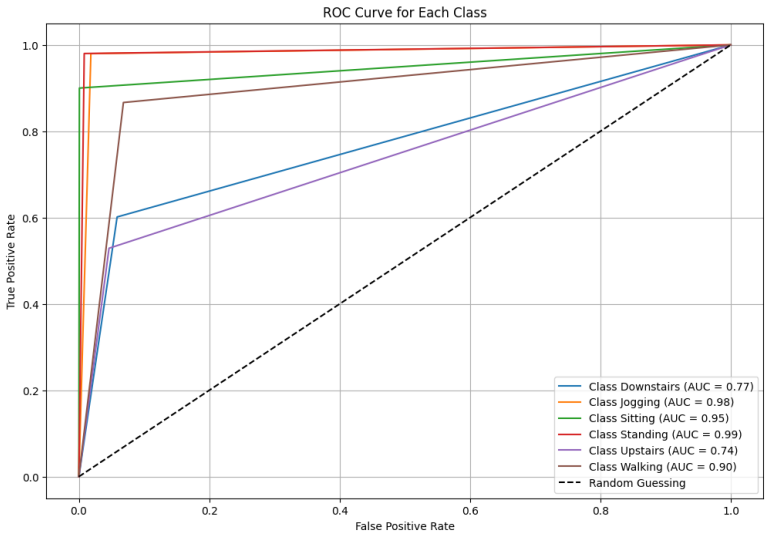


Figure 17: True positive rate of classifying each human activity against the false positive rate which illustrate the sitting and jogging have highest true positive rate.

3.1.3 Support Vector Machine

SVM Model Training Parameters and Performance			
Training data used	80%	Validation data used	20%
Parameter Used: kernel='rbf', gamma='scale', C=7			C=cv=5
Accuracy on Validation	0.8304	Accuracy on test data	0.7740
Hyperparameter Tuning: Parameter Used: kernel='rbf', gamma='scale', C=14			
Accuracy on validation data	0.8296	Accuracy on test data	0.7740
Hyperparameter Tuning: Parameter Used: kernel='rbf', gamma='scale', C=5			
Accuracy on validation data	0.8019	Accuracy on test data	0.7750
Hyperparameter Tuning: Parameter Used: kernel='rbf', gamma='scale', C=7			c
Accuracy on validation data	0.8090	Accuracy on test data	0.7786

3.1.4 Description of Models Training

The support vector machine was used to train the model, and a wide range of parameters were applied. Some of the parameters are defined in the summary table above. The best model achieved accuracy of 82% on training data, while the model achieved 78% accuracy on unseen data. The difference between the training and test data was only 5%, indicating that the model performed quite well in generalisation. To prevent overfitting of each model, fold 5 cross validation was applied. The first model, represented in the figure below, had a mean cross validation accuracy of 0.8079 and did well on unknown data; the second model, represented by a mean cross validation accuracy of 0.8304, did not perform likewise.

Although using a weight parameter to balance the data in the logistics regression class increased model performance, the data in the SVM model remained imbalanced in the results shown in the summary table. Nevertheless, the model's accuracy on test data did not change when class_weight='balanced' was applied to models 1 and 2, even though their performance on training data decreased by 1 and 3 points, respectively. The model's confusion matrix also showed that the class that jogging was most accurately identified, followed by the classes that sat and walked.

Despite having less values in the data set than other activity classes, the model sitting classification report was the most accurate with F1 score 1. Jogging's F1 score was 0.97, indicating that the model has correctly classified it. The movement upstairs and downstairs remained misclassified in this model with F1 scores dropping to 0.61 and 0.09, respectively as they were in the logistic regression model, which had rather high score in the downstairs example. The model's classification performance for walking has not improved. The model's performance is still 78%, which is less to the 80% of logistic regression.

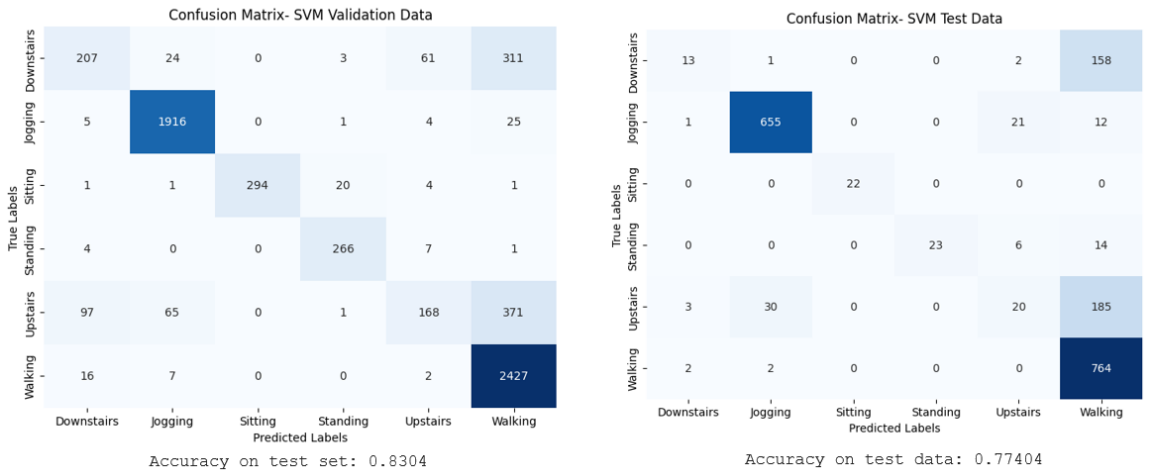


Figure 18: Model SVM performance on validation and test data was compared using the confusion matrix. The model did well on training data, with an accuracy of 83%, and it was able to generalise to an unknown test with an accuracy of 77%.

Classification Report of SVM

	precision	recall	f1-score	support
Downstairs	0.62	0.05	0.09	174
Jogging	0.95	0.98	0.97	689
Sitting	1.00	1	1	22
Standing	1.00	0.44	0.61	43
Upstairs	0.70	0.07	0.12	238
Walking	0.67	1	0.8	768
accuracy	0.78	1934		
macro	avg	0.82	0.59	0.6
weighted	avg	0.78	0.78	0.71

Figure 19: SVM Model performance present strong classification accuracy for jogging (F1 score: 97), Sitting (F1 score:1) and walking (F1 score: 80), while exhibiting limitations in classifying downstairs and upstairs activities. Overall model accuracy stands at 78.

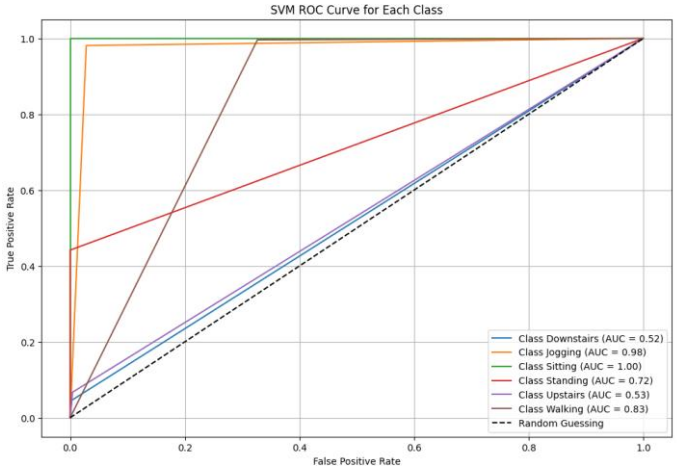


Figure 20: True positive rate of classifying each human activity against the false positive rate of SVM model illustrate the sitting and jogging have highest true positive rate. Upstairs (0.53) and downstairs (0.52) movement have lowest true positive rate.

3.1.5 Model Training Using Random Forest

Random Forest Model Training Parameters and Performance			
Training data used	80%	Validation data used	20%
Parameter Used: class_weight='balanced'			cv_scores:0.9332
Accuracy on Validation	0.9516	Accuracy on test data	0.8190
Hyperparameter Tuning: {'criterion': 'gini','max_depth': 20,'min_samples_leaf': 1, min_samples_split': 5}			cv_scores:0.9328
Accuracy on validation data	0.9429	Accuracy on test data	0.8190
Hyperparameter Tuning: {'criterion': 'gini','max_depth': 20,'min_samples_leaf': 1, min_samples_split': 5, n_estimators=200}			cv_scores: 0.9320
Accuracy on validation data	0.94136	Accuracy on test data	0.8231
Random Forest with Bagging Classifier			
Accuracy on validation data	0.82471	Accuracy on test data	0.82471
Random Forest with Ada Boost Classifier			
Accuracy on validation data	0.8195	Accuracy on test data	0.8195

Description of Models Training

Since random forests are among the best models for multi-class classification and handling imbalanced data, it has been thoroughly explored and used to a range of parameters and approaches with these models. The summary table above gives a brief overview of the models and the parameters that were employed. It was noticed during model training that the initial trained models did not perform well on test data, suggesting that these models would not be effective in generalisation. When bagging classifier and Ada boost classifier were used in conjunction with random forest in the ensembled approach. The model accuracy using bagging classifier on training data was 0.824, and the model performed the same on test data with accuracy of 0.82, indicating that the model performance was excellent while using these ensembles.

Similarly, the ada boosting model yielded the same results on test as it did on training data, with an accuracy of 0.8195. The optimal parameters were found using the grid search approach for bagging classifier. The model was then trained and evaluated, and a similar pattern with an accuracy of 0.823 was found on the training and test data. When compared to support vector machines and logistic regression, the random forest model performed admirably, yet it fell short of expected predictions.

Cross-validation was used during model training to make sure the model wouldn't get overfit.

Surprisingly, the model accuracy score during cross-validation fluctuates just 1 point in 5 folds when eliminating ensemble classifiers. However, when these models are applied to test data, the model accuracy decreases from 0.93 of training to 0.81 on test data, even though the standard deviation was only 0.006. The accuracy in cross validation of models 1 (0.9250), 2 (0.9332), and the bagging classifier was(0.9332) .

The basic random forest confusion matrix illustrates (figure 24, 25) how model performance declined from training to testing; while walking, sitting, and jogging were more accurately classified than other classes,

Surprisingly, the model performed slightly better on unknown data when compared to the training model for the upstairs and downstairs, with 34 true positives to 38 true positives and 103 true positives to 108 true positives, respectively. The model's output indicates that in order to improve the classification, an additional feature that clearly separates downstairs, upstairs activity from walking and jogging must be extracted. The bagging classifier is more accurate at classifying the activities, according to the confusion matrix of the Adaboost and bagging classifiers, even though the real positive value for the downstairs class decreased by 5. The ROC curves in the figure 26 shows the true positive and false positive rates for each class, demonstrating that while all models performed well in predicting sitting, jogging, and walking, their performance on stairs, particularly downstairs, was poor.

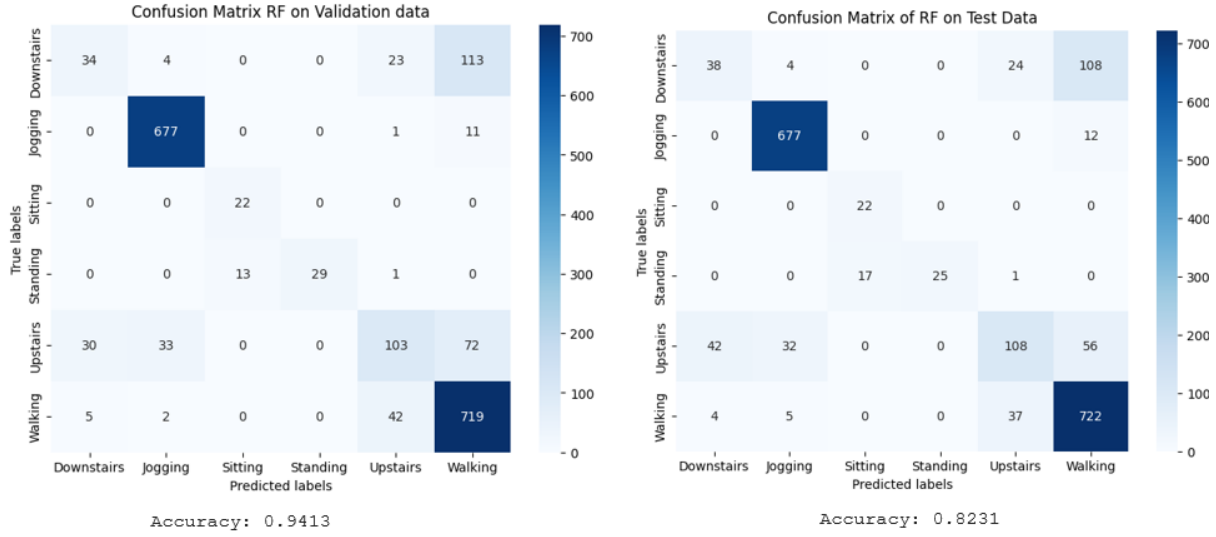


Figure 21: Random Forest performance on validation and test data was compared using the confusion matrix. The model did well on training data, with an accuracy of 94%, and accuracy on test data dropped to 82%. The difference in performance implies that was not good generalise to unknown data.

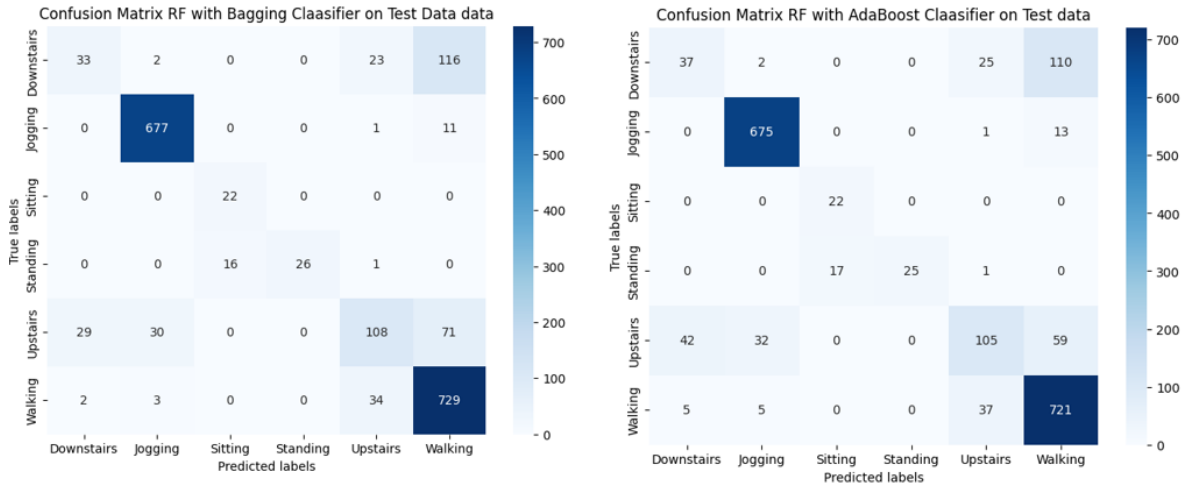


Figure 22: Ada and Bagging classifiers both did better on test data as did to training. Walking, jogging, and sitting are all accurately classified however upstairs, and downstairs continue to be substantially misclassified.

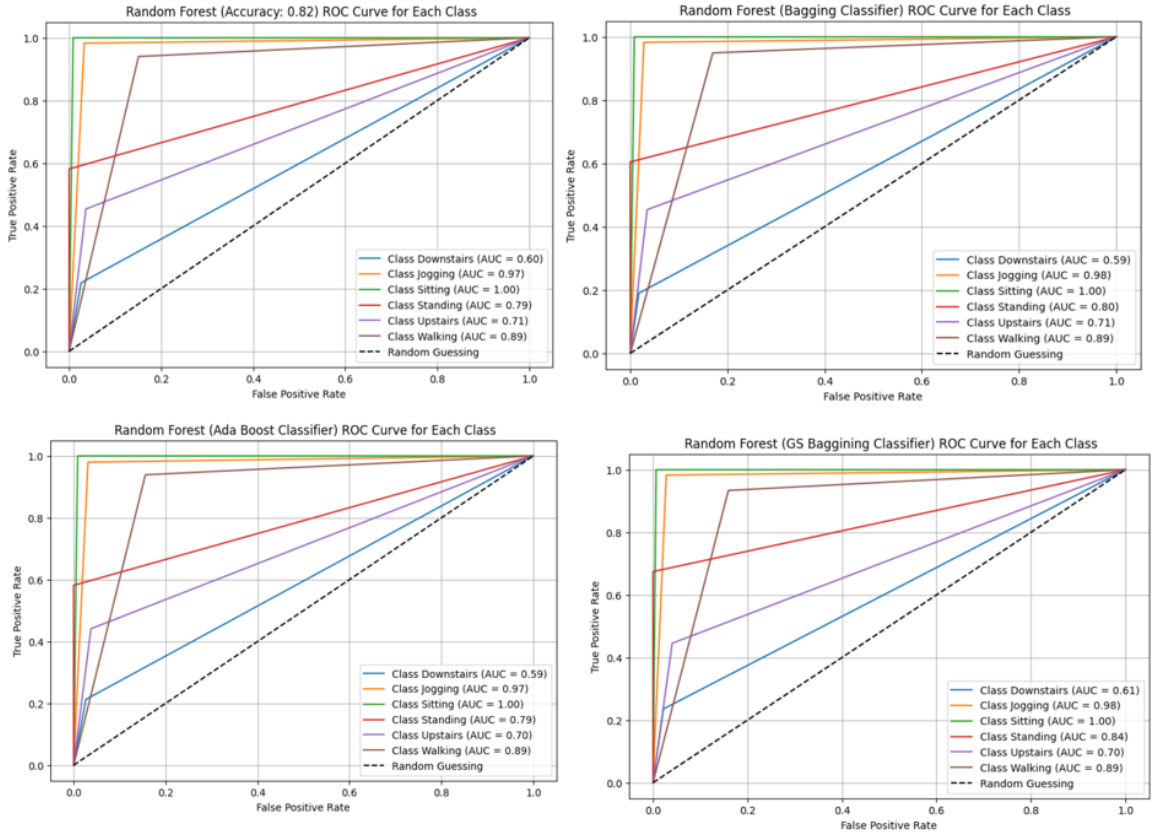


Figure 23: True positive rate vs false positive rate slight improved from random forest model to bagging, Ada boost and finetuned bagging classifier, with little change in accuracy the overall model's performance was 82%.

3.1.6 Model Training Using Gradient Boosting Machine

3.1.7 Description of Models Training

GBM Model Training Parameters and Performance			
Training data used	80%	Validation data used	20%
Parameter Used: learning_rate=0.2, kfold=5, kfold-10			cv_scores:0.8851 cv_scores:0.9002
Accuracy on Validation	0.9516	Accuracy on test data	0.9405
Hyperparameter Tuning: {learning_rate': 0.8, 'max_depth': 5, 'n_estimators': 100}			cv_scores: 0.9351
Accuracy on validation data	0.9405	Accuracy on test data	0.9366
Hyperparameter Tuning: {n_estimators=100, learning_rate=0.6, random_state=42, max_depth=5, criterion='squared_error'}			cv_scores: 0.92
Accuracy on validation data	0.9524	Accuracy on test data	0.9524
Hyperparameter Tuning: {n_estimators=100, learning_rate=0.6, random_state=42, max_depth=5, min_samples_split=2 criterion='squared_error'}			cv_scores: 0.8962
Accuracy on validation data	0.9453	Accuracy on test data	0.9453

GBM was explored using a number of parameters, only a few of which are listed in the above table. The learning rates that attempted were {0.15, 0.1, 0.2, 0.3, 0.35, 4,5, 6, 8}, with the number of estimators. The

model did well on both training and test data, and overfitting was prevented by using cross validation. Although a steady change in accuracy was seen for each fold during cross validation, it was noted during model training that the cross-validation accuracy score differed significantly from the training accuracy score. For example, the model's training accuracy was 0.9405 when it was trained in the first mode. However, after applying cross validation, the model's mean accuracy dropped to 0.8851, but each fold had a stable score [0.86956522 0.89328063 0.88888889 0.88888889 0.88492063]. To reduce the large discrepancy between training and cross fold accuracy the number of folds were increase from 5 to 10 and it was observed that the mean accuracy for cross validation increased to 0.9002 and standard deviation for cross validation remained close to 0.0.

Three models were chosen to demonstrate the GBM findings, and it is clear from the confusion matrix that the model did a good classification of seated, jogging, and walking activities. The performance of the GBM model improved to 2% from random forest. Still, GBM failed in accurately classifying the movement downstairs and upstairs. The accuracy is shown in the classification report model together with recall and F1 score. For example, walking had an F1 score of 86 but a precision of 0.82, while downstairs had a precision of 0.38. Similarly, the F1 score was 0.85 but the seated precision score was 0.73. The model's total accuracy was 82%, nevertheless, which was considered to be an improvement over the previously trained model. The ROC curves for the three models shown in the figure confirmed the results above, and it is evident that the majority of false positives were identified as occurring downstairs. The true positive rate is higher for class sitting and jogging.

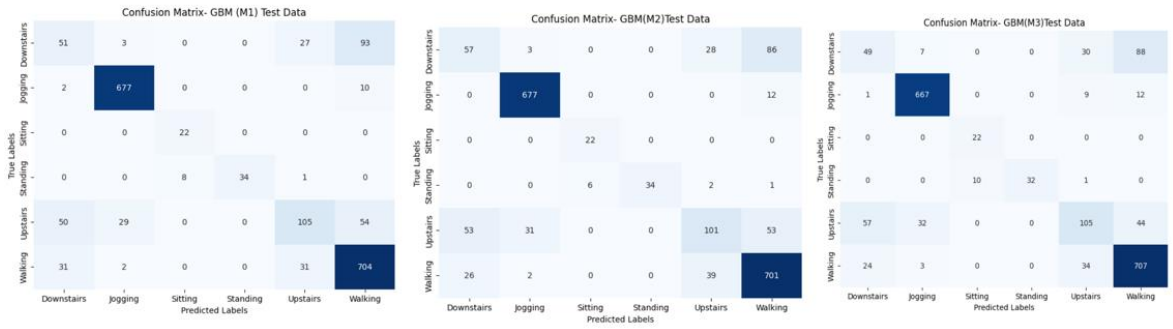


Figure 24: The accuracy of the gradient boosting classifier on unknown data was 82%, and when the model was startled with varying parameters, these three models showed a small improvement. While sitting (22), jogging(677), and walking(704) are all correctly categorised, upstairs(105) downstairs(51) and other activities are still significantly misclassified

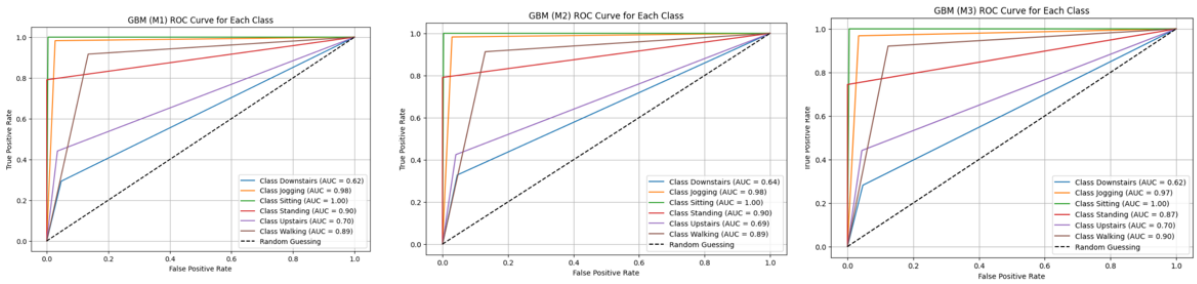


Figure 25: In GMB models the movements upstairs and downstairs are incorrectly labelled as walking and jogging, the ROC curves that show the models of true classification of labels.

3.1.8 Model Training Using Decision Tree

Decision Tree Model Training Parameters and Performance			
Training data used	80%	Validation data used	20%
Parameter Used: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': , 'min_samples_split': 2, splitter='best'}			cv_scores: 0.8753
Accuracy on Validation	0.8827	Accuracy on test data	0.8117
Hyperparameter Tuning: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 4}			cv_scores: 0.8642
Accuracy on validation data	0.8803	Accuracy on test data	0.8097
Hyperparameter Tuning : {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': , 'min_samples_split': 2, splitter='best'}			cv_scores: 0.9356
Accuracy on validation data	0.88272	Accuracy on test data	0.7911

Description of Model Training

After failing to attain the target accuracy on an unseen dataset from GBM a decision tree was used to train a model with a wide range of parameters for maximum depth, minimum sample leaf, and minimum samples split with criteria, Entropy and Gini to fine-tune the model. To address overfitting in models, cross validation is applied to each model. The maximum accuracy obtained as a result of hyper tuning was 0.8117 on the test data set, which is near to the accuracy of the GBM model. During the decision tree model's, it was found that the upstairs and downstairs classifications improved—something that had not happened in any prior model—and that there was just a slight decline in walking and jogging accuracy, which would be regarded as trade-off between classifying the activities.

The confusion matrix shows that while walking, jogging, and sitting are more properly identified, the downstairs true case was improved to 74 true labels. Similar improvements were made to the upstairs movement, with 113 examples turning out to be true predictions. The accuracy of sitting decreased slightly in model three, which was never seen in any of the previously trained models. Additionally, it was determined that gini was the ideal model criterion and that raising the max depth had no detrimental impact on the accuracy of the model. The model classification report's (figure 31) F1 score displays the trade-off between precision and recall. The F1 scores for the moves upstairs and downstairs were, respectively, 0.51 and 0.44. The ratio of true and false positive assumptions was displayed via the ROC curve. When compared to other models, model 2's standing accuracy decreased, but model 3's sitting accuracy slightly decreased, and its downstairs prediction of actual cases increased. This implies that decision tree was good in classifying the dynamic movement compared to static movement of sitting and standing, however this model did not help in attaining the desired accuracy.

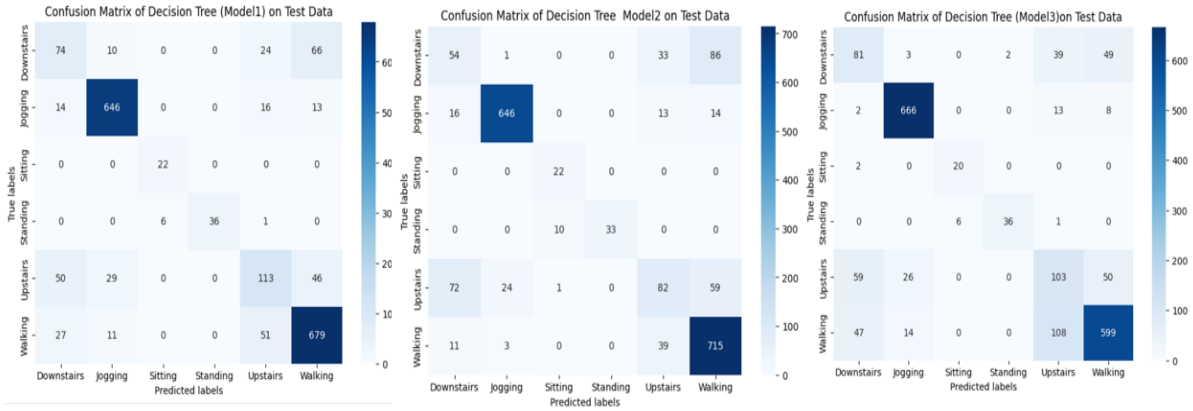


Figure 26: Confusion matrices provide as an example of correctly and incorrectly identified human activity labels. The decision tree model significantly outperformed, classifying the up and down stairs movement compared to logistic regression, random forest and support vector machine. However, the model overall performance is just 81% of classifying six human activities.

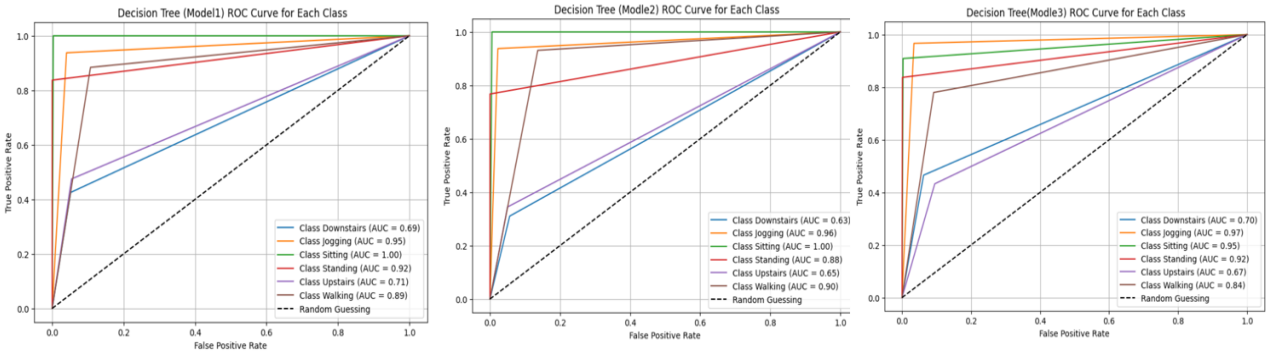


Figure 27: In models 1 and 2, sitting was most accurately classified, followed by walking and jogging. In model 3, jogging was classified as the most accurate activity, followed by walking. The downstairs classification was somewhat enhanced even though it was not as good as it needed to be.

3.1.9 Model Training Using K-Nearest Neighbors (KNN)

KNN Model Training Parameters and Performance			
Training data used	80%	Validation data used	20%
Parameter Used: {n_neighbors=5, leaf_size-30}			cv_scores: 0.8753
Accuracy on Validation	0.943740	Accuracy on test data	0.80713
Hyperparameter Tuning: {'leaf_size': 40, 'n_neighbors': 4}			cv_scores: 0.9294
Accuracy on validation data	0.94136	Accuracy on test data	0.81178
Hyperparameter Tuning: '{leaf_size': 40, 'n_neighbors': 3}			cv_scores: 0.93224
Accuracy on validation data	0.94057	Accuracy on test data	0.81954

3.1.10 Description of Model Training

Following decision tree training, the KNN model was employed because of tree structure algorithm demonstrated good performance on moving stairs, which were largely misclassified in other training algorithms. The model's overall performance was comparable to that of the GBM model, with an accuracy rate of 82%. As the number of neighbours increased, the model's performance declined below 0.80. As a result, an

effort was made to attain accuracy with few neighbours. When compared to the other models, Model 3 performed the best, and it was able to classify sitting, walking, and jogging with a high degree of accuracy. The models' confusion matrix shows the true and false classification. This does not perform as well on stairs as anticipate based on the algorithm's tree structure. The figure 34 displays the ROC curve, which provides a more improved visualisation of the model's performance across several classes.

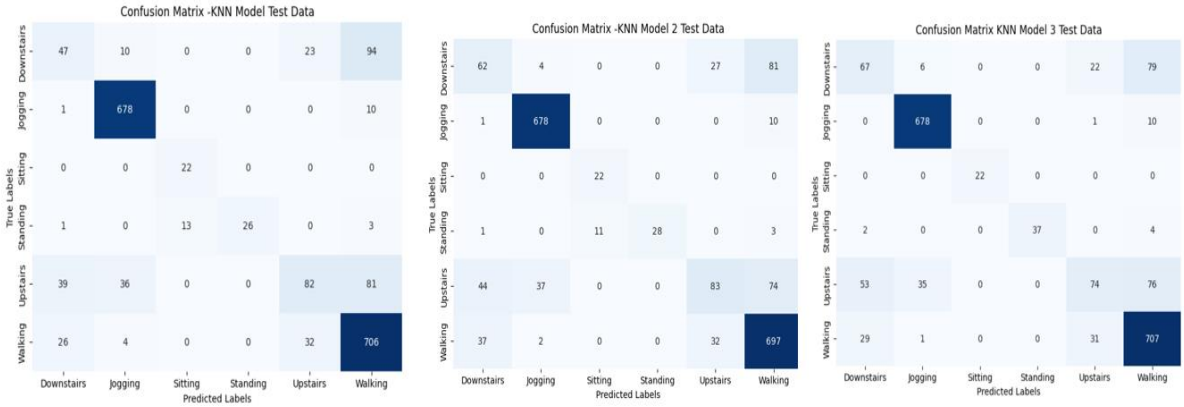


Figure 28:The KNN achieved an overall accuracy of 82%, although it was unable to classify the movement of stairs more precisely than what was predicted based on the decision tree results. Walking and sitting, however, continued to be the most properly identified activities.

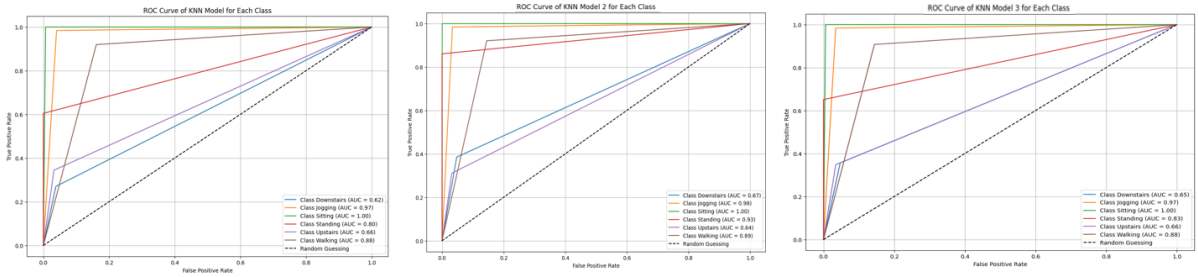


Figure 29: The true positive classified classes can be seen in the ROC curve, and it is evident that in model two standing accuracy improved and true classified labels are 83. While stairs movement overlapped in model 3.

3.2 Phase -2: ML Model Training Using Timeseries Dataset

As stated in section one about the features, additional features were derived from time series data because, during the first phase of model training, the required accuracy was not achieved. Kurtosis, IQR, skewness, and power of x, y, and z were employed in the KNN and CNN models in an effort to improve model performance on training and test data. The time series data was entirely used for model training in the LSTM and Prophet models.

3.2.1 Model Training Using Artificial Neural Network

On updated features, the artificial neural network was trained. In these models, multiply layer, filters, pooling, and dense layer were applied to get better results. Three models were chosen to present results in the report as given in figure 35. It was noted, nevertheless, that the accuracy was not being improved by the complex model architecture. Model three, which has an accuracy of 0.87228 on test data (unseen data), performed best among the models whose architecture is shown in figure. On test data, it was clear that the neural network with more

features helped to increase accuracy from 82% to 88%. To prevent overfitting of the model, regularisation and early stopping of model performance was not improved after certain epochs.

Plots of the models' accuracy and loss for training and validation were created (figure 36). The plot of model 2 makes it clear that a complicated model architecture leads to a significant difference in training and validation accuracy, whereas a simple model results in a smaller difference between validation and training accuracy.

Regarding the classification of movement on stairs, these three performed well with little variation in accuracy. Model 2 identified the majority of actual downstairs movement, whereas Model 1 identified the majority of true upstairs classes. In identifying all tasks, the model 3 trade-off between these movements and its overall performance was high in classifying all activities.

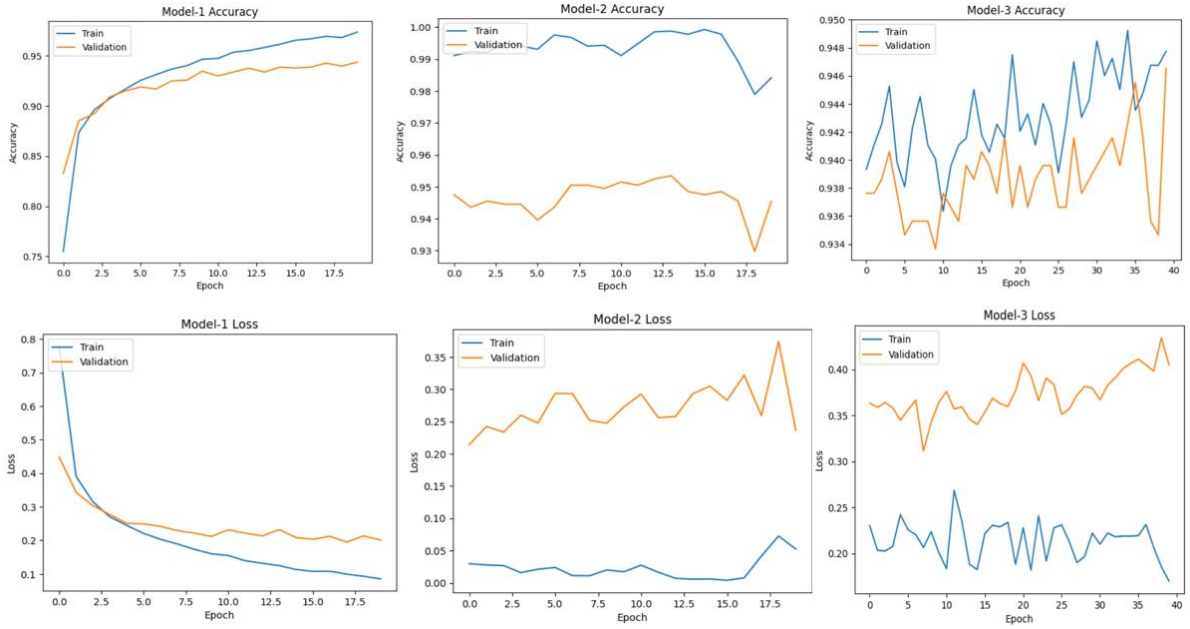


Figure 30: While model 2 with layers and a large number of filters leads to a considerable difference in accuracy and loss between training and test, model 3 with two layers and dropout did well in classification. Model 1 with two layers had little variation in accuracy and loss between validation and training. This implies the simple worked better in dataset.

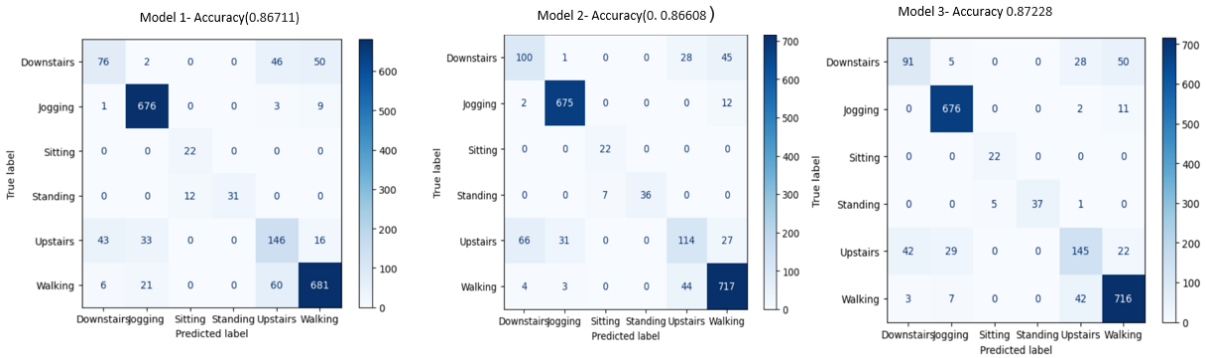


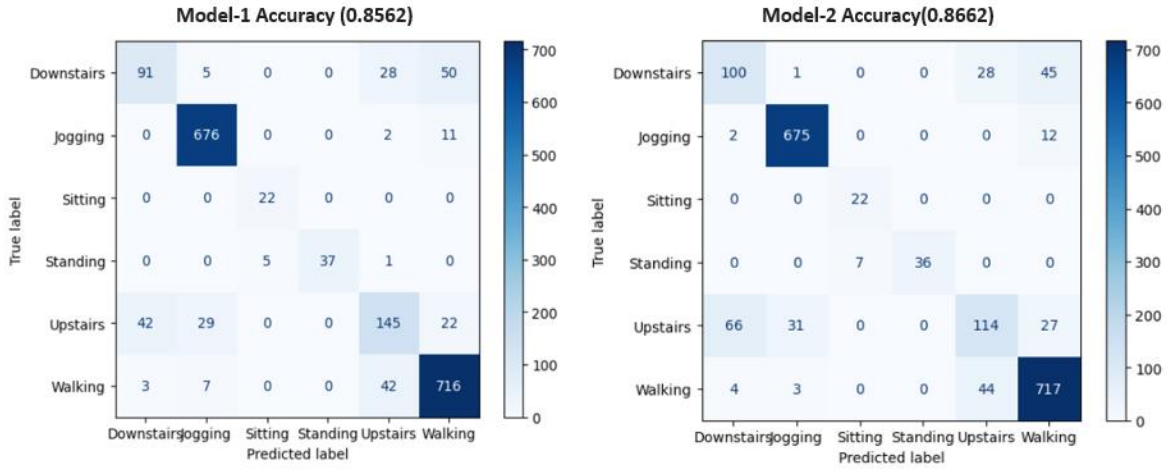
Figure 31: The KNN's confusion matrix displays the true and incorrect labels for the models. Even if true classification improved from the previously trained model, upstairs are, at most, misclassified in all models. Three KNN models have an accuracy of 88%.

3.2.2 Model Training Using Convolution Neural Network (CNN)

Working with a neural network was extended to the CNN model since the neural model outperformed all the other trained models. A large range of parameters are employed in the CNN model to fine-tune the model's intended accuracy. At each layer, the parameter is compressed using max pooling. Various pooling sizes (2,3), combination kernels, and batch sizes are employed. The best results were obtained by Model 2, which had an accuracy of 0.8562, a slight improvement above KNN. Tree layers with filters were employed; each layer's dropout ratio was 0.5 and pool size used was two. Different epoch batch sizes and dense layer neurons were attempted, but the best accuracy was not achieved. Three layers of neurons were also rather successfully executed by Model 1. Model 2's accuracy was 0.98 on test data and 0.92 on validation, both of which are pretty acceptable. However, the model's loss went from 0.01 on training to 0.46 on validation. 0.8660 was the model's accuracy on the test set.

In model 1, on the other hand, validation accuracy was 0.934 and loss grew from validation to reach 0.22 in 20 epochs, whereas training accuracy was 0.97 with a loss of 0.05. The model's accuracy on unobserved data was 0.8562. The significant difference between training and validation loss in Model 1 (0.05 vs 0.22) suggests potential overfitting. Which implies that model performs well on training data but struggles to generalize to unseen data. Many attempts were made to reduce the loss and increase accuracy, when loss was reduced than the accuracy of model dropped, It was left here and using regularization technique it was attempted to reduce in phase three of model training.

Figure 32: Model 1, 2 showed good performance on both test data around 0.98 and validation data approximately 0.93 while the loss value both models increase significantly from training to validation.



3.3 Phase -3: ML Model Training Using Updated Features.

3.3.1 Improved Logistic Regression Model

The accuracy of the logistic model increased from 80% with the initial metadata to 84% with the updated metadata when new features were extracted from time series dataset and these features importance were defined in section 1 of report. With these updated features, logistic regression was applied, and it was observed that the additional features made it possible to classify the upstairs movement with precision of 0.84, up from 0.74 previously, and the downstairs movement improved from 0.31 to 0.47. Even while the model's performance not improved as expected the reason would that the logistic regression has its limitation with class classification.

The F1 score, which is determined by balancing the cost of recall, is provided by the classification report of the model. It is clear that for high precision standing was 0.58 and recall 1, which is why its F1 score was 0.74. This confusion matrix, which shows the smaller number of true negative and false positive situations, is derived from validation data. Even though the accuracy of the stair's movement prediction has increased, the classification of running, standing, sitting, and walking is still most precise. It's interesting to note from the ROC curve that the maximum prediction accuracy was attained for standing, sitting, and running.

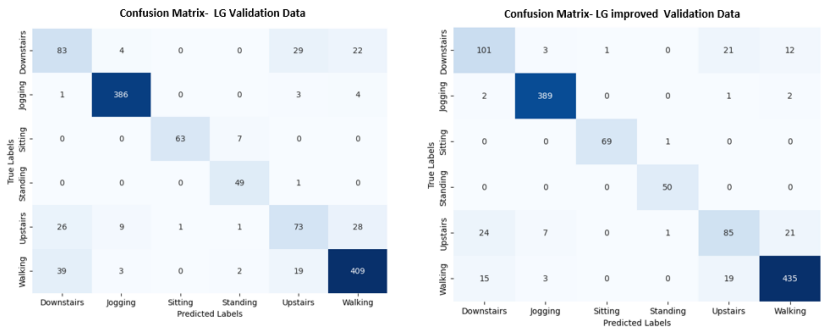


Figure 33: comparing the actual classified activities from the first data set with the new attributes that were retrieved to determine the true classified activities.

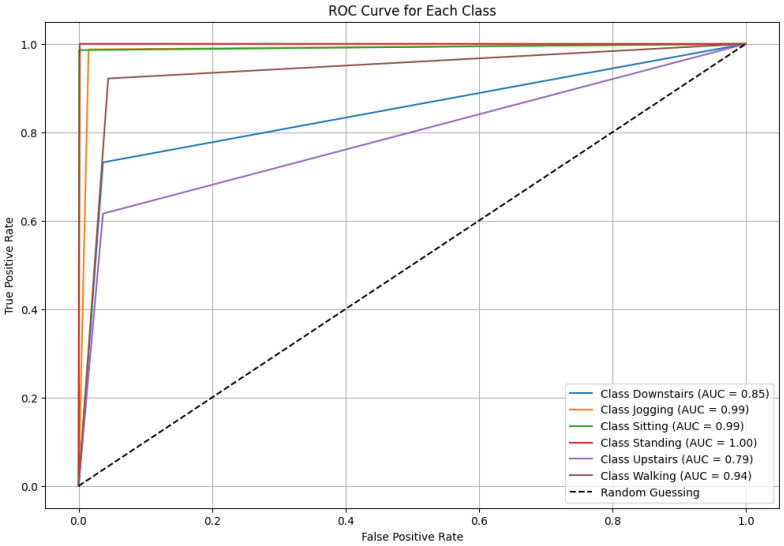


Figure 34: The modified logistic regression model's ROC curve indicates that each class's prediction accuracy increased with standing, sitting, and jogging being the most accurately classified activities.

Model Training with Feature Selection

Models were trained with all features in Phase 1 and the beginning of Phase 2, however the expected accuracy was not attained. In order to remove any noise that might be contributing to the failure to achieve the required accuracy, it was determined to focus on the key features that actually aid in classification. The approaches that were used are described in the model training section that follows.

3.3.2 SVM Feature Selection with Correlation

Correlation was calculated for data set in order to determine which features are most corelated. The variables that corelated with each other were identified, and the single variable from each pair of corelated variables was removed in order to obtain the uncorrelated variables. Support vector was used to train the model and provide predictions for the data set after the initial variable selection. Both test data files were pre-processed to match the variables of training and testing. However, this approach was remained ineffective, and the accuracy obtained was comparatively low compared to the 84% accuracy that was achieved with the enhanced logistic regression model. It wasn't proved a successful method to train, so the test and training results for the models are not discussed here.

3.3.3 Models (RF, CNN, GBM, LSTM) with Backward Feature Selection

Following an unsuccessful attempt at feature selection using correlation, the backward feature selection method was employed to choose the fewest features possible that would aid in model training by lowering noise (unnecessary variables). In this approach, the 19 variables that are meant to contribute to the 87% accuracy were chosen. A list of the chosen features is provided below.

List of Selected Features with Backward Feature selection			
total power for x, y	Kurtosis for x, y, z	IQR for x,y,z	Skewness for x,y,z
'x axis	y axis	z axis	Resultant magnitude y,
Median, mean, variance,	Median, mean, variance,	Median, mean, variance,	resultant magnitude z,
standard deviation,	standard deviation,	standard deviation,	PSD mean,
root mean square	root mean square	root mean square	Energy x, energy z,
maximum, absolute maximum'	maximum, absolute maximum'	maximum, absolute maximum'	Resultant direction

Following the selection of features, three models were applied: random forest, CNN, and GBM. The accuracy obtained was approximately 85%, which was near to the intended but not quite there. Afterwards, the best predictions were combined using the ensemble classifier. 86% accuracy was obtained with the ensemble classifier, which was seen as a minor improvement. The accuracy attained in each model, as well as in the validation and test data sets following parameter exploration, is briefly summarised in the model training and parameter table that follows.

Models Training Parameters and Performance				
Training data used		80%	Validation data used	20%
RF	Accuracy on Validation data	0.9587	Accuracy on test data	0.8438
CNN	Accuracy on validation data	0.9398	Accuracy on test data	0.8505
GBM	Accuracy on validation data	0.93026	Accuracy on test data	0.85160
Ensemble Classifier	Accuracy on validation data	0.9516	Accuracy on test data	0.86504

The random forest confusion matrix shows that the walking, sitting, and jogging had the highest precision while the upstairs prediction improved to 122 valid labels—a performance not achieved by any trained model. The CNN model was the best model yet, because its overall predictions were the most accurate since it predicted a high number of true labels in all activity classes, with little decline in walking and standing compared to random forest. As can be seen from the confusion, the third model used, GBM, was just as accurate in predicting the actions as CNN was in classifying them. One interesting finding made throughout the CNN models' training process was that the models' validation loss value remained low, in contrast to the CNN models trained with all features, whose loss value increased with the number of epochs.

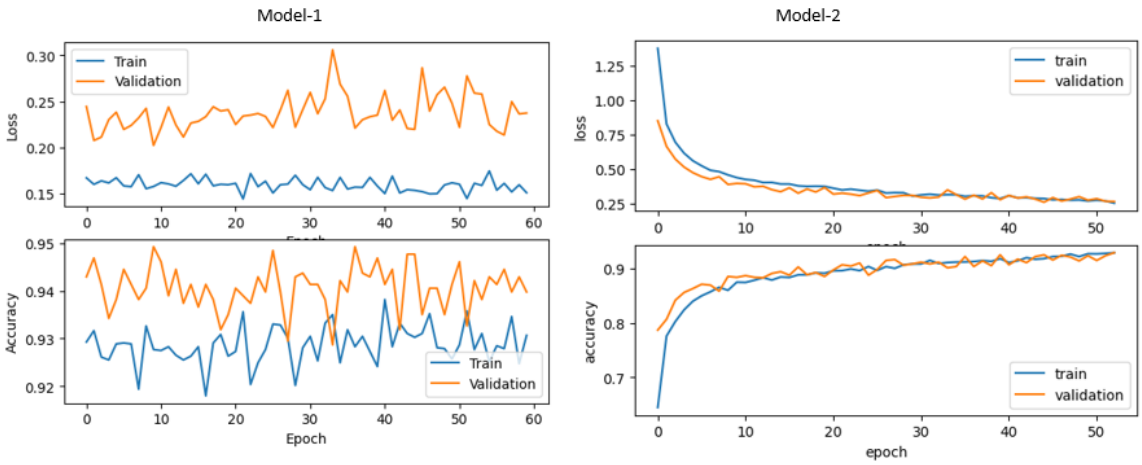


Figure 35: When comparing the accuracy of training and validation loss, it was found that when a noise was removed from the data set, the difference between training and validation loss and accuracy decreased significantly and the model's accuracy in predicting activities was increased to 86%.

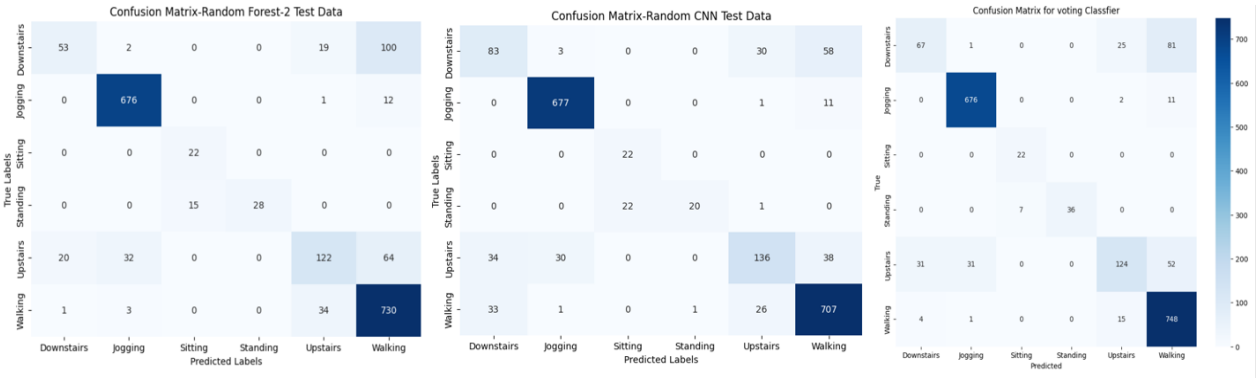


Figure 36: The confusion matrix of CNN and RF shows an increase in predicting true classes and when these models were aggregated to reduce the false predicted labels with ensemble classifier the accuracy of prediction increased to 86%.

3.3.4 LSTM Model Training with Selected Feature

The goal was to achieve 92% accuracy on test data after achieving 86% accuracy. The LSTM model was explored using a minimal model architecture consisting of 2 layers, each with 15 units, and a dropout set of 0.5. Although it was expected that the LSTM's improved machine learning capabilities would help it approach the target accuracy, the model's greatest accuracy of 83% was attained when it was trained using various layers and units.

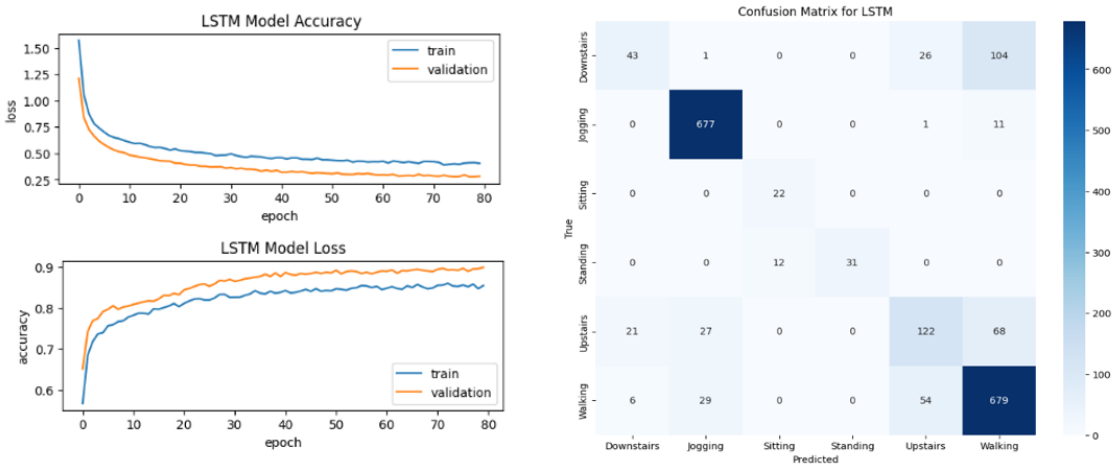


Figure 37: The best model of the LSTM only reached 83% accuracy, however, the loss and difference between training and validation remained small, implying that the model performed rather well on unseen dataset.

3.3.5 RNN Model Training with Selected Feature

In order to obtain the desired level of accuracy from the deep learning method, various model architectures were tested for the RNN model. The optimal model design comprised two layers with 60, 50 features and dropout rates (0.5). The accuracy obtained was 0.83092 which low from the accuracy obtained with CNN. It suggests that RNN is ineffective this dataset (selected features). The confusion matrix and roc curve clearly show that walking, which was the initial instance in this project's model training, was poorly classified. But even with a large number of epochs, the validation loss value remained small, which supported earlier findings about validation data loss.

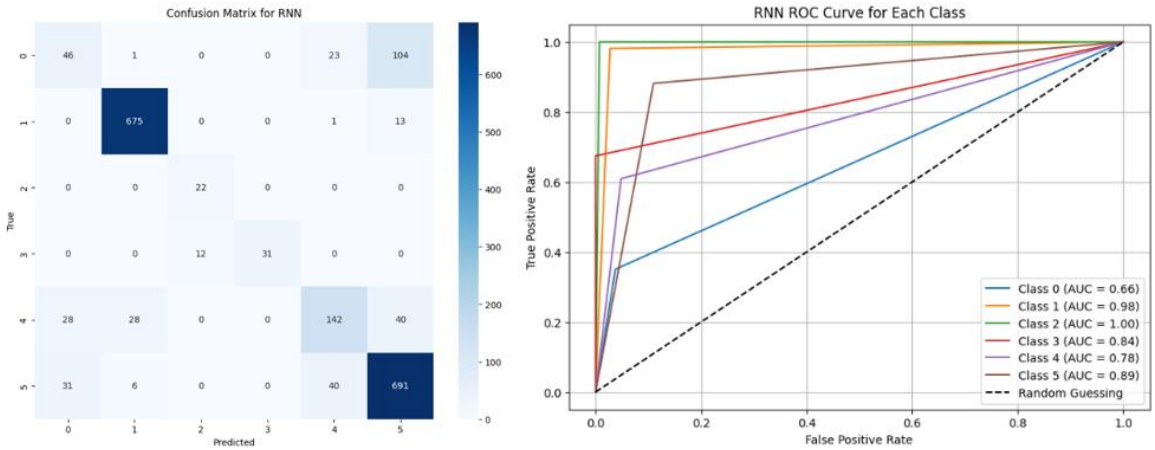


Figure 38: The RNN model's overall performance is shown by the confusion matrix and ROC curve. Its 83% accuracy rate indicates that many classes are misclassified, and the walking prediction true positive rating dropped to 0.7.

3.3.6 Gradient Boosting Model with Feature Interaction

The required accuracy hasn't been reached or increased significantly thus far. A new approach was applied, which included calculating the feature interaction and expanding the data set by adding test sets and metadata. After the data was combined and the feature interaction was calculated, 900 features were used

for model training. The gradient boosting model was trained and assessed using twenty percent of the data. The validation classification report demonstrates that the model worked effectively in classifying each variable, with 92% accuracy. However, when the model was tested with Kaggle (unseen data), its accuracy dropped to 83%.

Classification Report:

	precision	recall	f1-score	support
Downstairs	0.81	0.62	0.7	156
Jogging	0.95	0.98	0.97	528
Sitting	0.97	0.99	0.98	69
Standing	1.00	0.94	0.97	64
Upstairs	0.77	0.77	0.77	188
Walking	0.94	0.98	0.96	644
accuracy	0.92	1649		
macro	avg	0.91	0.88	0.89
weighted	avg	0.92	0.92	0.92

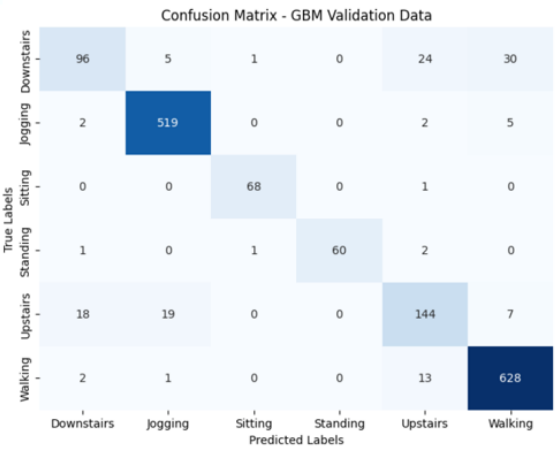


Figure 39: The model that trained using feature interaction and a large training size, achieved 92% accuracy suggesting that a large training area would aid in achieving the high classification accuracy.

3.3.7 Models Training with Feature Importance

The significance of features was determined using the feature importance parameter. The mean of the features' importance was then used to choose which features to train the model with, and 22 features in total choose. Because the random forest, GBM, and CNN models performed well on this dataset, these were the algorithms used again to train the new model. The model was trained using the GBM; its reported training accuracy was 94%, and its accuracy on one unseen dataset (the test) was 92%; however, its accuracy on another unseen dataset (Kaggle) decreased to 80%. The model was fine-tuned and cross-validated, and it was not proven effective on the Kaggle dataset.

Random forest was employed for training, and the results showed that they performed equally well in all unknown datasets, with accuracy rates of 86% and 85% on test data and Kaggle, respectively. CNN was then used to improve accuracy on unseen datasets, with the multilayer layers added to the model architecture, however it performed poorly on both unseen datasets. As a result, random forests with important features proved to best model to achieve the highest score on both unseen data sets.

3.3.8 Final Model Training with Feature Importance

Extra tree Classifier Training Parameters and Performance			
Training data used	80%	Validation data used	20%
Parameter Used: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}			cv_scores: 0.94255
Accuracy on Validation	0.961172	Accuracy on test data	0.8469
Hyperparameter Tuning: 'max_depth': 12, 'min_samples_split': 5, 'min_samples_leaf':2'n_estimators': 150}			cv_scores: 0.9194
Accuracy on validation data	0.90808	Accuracy on test data	0.84723
Hyperparameter Tuning: {'max_depth': 14, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}			
Accuracy on validation data	0.92789	Accuracy on test data	0.8546

The additional tree classifier was the last model that contributed to achieving high accuracy on unknown datasets (test and Kaggle). Prior to the model being trained, the feature's importance value was determined, and the variables were chosen based on their mean importance. The models were trained using several parameters in order to determine which would work best with the dataset. The table above summarises the model training and parameter used. The total accuracy that achieved was 0.85968. This model equally performed good on both unseen datasets The test data was used to generate a confusion matrix, which shows that standing and sitting were the most accurately classified activities, followed by walking and jogging, with very few miss classifications.

While the upstairs and downstairs stair movements were the most incorrectly classified classes in this dataset, the model's performance in classifying stairs movement suggests that additional features are needed to best explain why these movements are classified as walking jogging. The F1 score indicates the model prediction accuracy and helps to better comprehend the precision of each class when compared to recall. It also shows that the recall value for stairs movements was low, which is why the F1 score of these two classes decreased. The true positive vs. false positive rate of the model's performance is further visualised using the ROC curve.

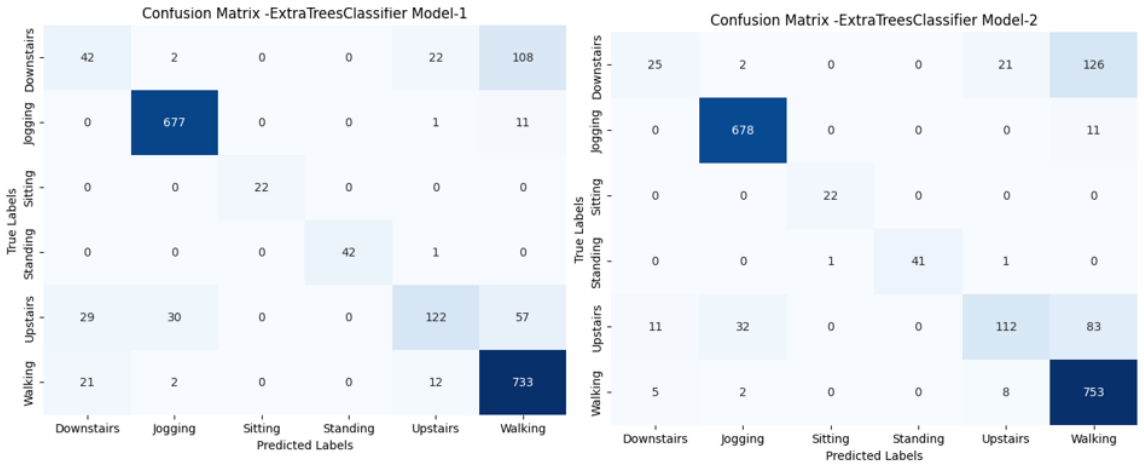
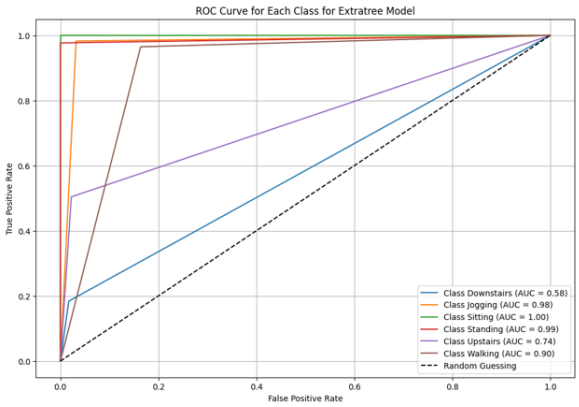


Figure 40: The models generalise well on the test data set and attain accuracy rates of 84% and 85%, respectively. With the exception of the downstairs and upstairs classes, all classes had high true classified labels with very few cases of misclassification.



Classification Report ExtraTreesClassifier- Model2

	precision	recall	f1-score	support
Downstairs	0.46	0.24	0.32	174
Jogging	0.95	0.98	0.97	689
Sitting	1.00	1	1	22
Standing	1.00	0.98	0.99	43
Upstairs	0.77	0.51	0.62	238
Walking	0.81	0.95	0.87	768
accuracy	0.85	1934		
macro	avg	0.83	0.78	0.79
weighted	avg	0.83	0.85	0.83

Figure 41: The downstairs and upstairs area under curves were relatively low, indicating that the model's ability to discriminate between these two classes was inadequate. The F1 Score for these classes offers further confirmation that, while the model did well in other classes, it did not perform well on stairs movements.

4 Discussion and Results

In order to train models that would perform equally well on any unknown dataset and best classify the human activities (walking, jogging, sitting, standing, and going upstairs and downstairs), fourteen machine learning algorithms were employed. The training of the models was carried out in three phases, using various techniques such as extracting additional features (59) from the initial set of features (30) and timeseries, applying feature interaction between features, balancing the data set with both smooth and under sampling, choosing features with correlation values, determining the feature importance, removing noise from the dataset, selecting features using a backward approach, merging datasets to increase training size, applying stratify in data splitting for training and testing, and employing the ensemble method to obtain the highest prediction.

From the initial phase of the model training process, it was found that the accuracy of the model increased when the machine learning model changed to a more robust model since GBM and RF achieved the best predictions from the original dataset. All of the models—RF, GBM, and CNN in particular—performed well with additional features added, but the desired accuracy was not attained. This suggests that adding more features alone would not be very helpful in achieving high accuracy. Nevertheless, a few other variables, such as PSD, energy, acceleration, resultant direction, magnitude, and upstairs and downstairs movements, aid in obtaining overall accuracy and precision.

Subsequently, various techniques were employed to eliminate irrelevant variables (noise) from the data set, leaving only significant features that could provide more of an impact to accuracy. The feature importance method worked very well for CNN and RF and extra-tree classifier, but it did not perform well for LSTM, RNN, or GBM. Shuffling features, expanding the training set, calculating the features interaction and balancing the

classes didn't work or help to achieve accuracy. In contrast to all complicated model architectures, it was observed during model training that the deep learning simple model architecture performed well on unknown data sets.

While using overlapping snippets to add features and train the model to work with time series data sets appeared to be a powerful and promising approach, however the time series data that was provided had inconsistent timestamps, making it difficult to achieve and it remained impossible for us to figure out how to deal with them to make prediction for Kaggle dataset. The best model extra-tree classifier contributed to the 0.85968 total accuracy. In terms of predicting the classes for jogging (F1-score: 0.97), walking (F1-score: 0.87), sitting (F1-score: 1), and standing (F1-score: 0.99), the model did well. In terms of predicting the classes for downstairs (F1-score: 0.32), and upstairs (F1-score: 0.62), the model's performance declined. This pattern with various models and features was observed, which suggests that it may have occurred for reasons that I am not aware of.

Inconsistent timestamps in the initial timeseries data, from which all features were extracted, could be the cause of the significant error in the model's capacity to learn the temporal patterns of movements upstairs and downstairs. Even though for many classifiers a parameter for balancing data was used, it's still important to analyse the specific balance between downstairs and upstairs data in comparison to other data to determine what might be causing the imbalance. Possible causes for imbalance would include situations where sitting users' devices are not activated, or the placement of the devices used to collect the data. We may not have fully captured all the details of upstairs/downstairs movements, despite the fact that additional features and feature importance assisted with some models.

The model's capacity to distinguish between downstairs and upwards movements has to be improved in future work. The model's inability to identify temporal trends was probably caused by the inconsistent timestamps in the original data. Future considerations might include interpolation, or windowing approaches to address this. To address the imbalance between the data from the upstairs and downstairs, techniques like oversampling would also be taken into consideration in relation to other activities. Lastly, employing advanced models like LSTMs and exploring feature specific to vertical movement can aid in capturing the specifics of these motions.

Reference

1. Bennasar, M.; Price, B.A, Gooch, D.; Bandara, A.K., Nuseibeh, B. Significant Features for Human Activity Recognition Using Tri-Axial Accelerometers. Sensors, 2022, 22, 7482. https://doi.org/10.3390/s22197482
2. Zhu, J., San-Segundo, R. & Pardo, J.M. Feature extraction for robust physical activity recognition. *Hum. Cent. Comput. Inf. Sci.* **7**, 16 (2017). https://doi.org/10.1186/s13673-017-0097-2

Appendix: Representative Sample Code

See the next page for code

Execute the code chunk in sequence, after few code, there is instruction to skip the following to code chunks to time delay in execution, after skiping those code chunks , resume the execution in sequence

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
url1="https://raw.githubusercontent.com/ZainabMCheema/Exploratory-Data-Analysis/main/Human%20Activity%20Classifiction%20Dataset/Updated_
url_2="https://raw.githubusercontent.com/ZainabMCheema/Exploratory-Data-Analysis/main/Human%20Activity%20Classifiction%20Dataset/Updated
url_3="https://raw.githubusercontent.com/ZainabMCheema/Exploratory-Data-Analysis/main/Human%20Activity%20Classifiction%20Dataset/Updated
df_meta=pd.read_csv(url1)
df_test=pd.read_csv(url_2)
df_kaggle=pd.read_csv(url_3)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
#file_path3='/content/drive/My Drive/Course Dataset/Updated_feature_meta.csv' #k
#file_path4='/content/drive/My Drive/Course Dataset/Updated_feature_test.csv'
#file_path5='/content/drive/My Drive/Course Dataset/Updated_feature_kaggle.csv'
```

```
#file_path3='/content/drive/MyDrive/Machine Learning- Session/Course Work Project/Updated_feature_meta.csv' #zainab's
#file_path4='/content/drive/MyDrive/Machine Learning- Session/Course Work Project/Updated_feature_test.csv' #zainab's
#file_path5='/content/drive/MyDrive/Machine Learning- Session/Course Work Project/Updated_feature_kaggle.csv' #zainab's

#df_meta=pd.read_csv(file_path3)
#df_test=pd.read_csv(file_path4)
#df_kaggle=pd.read_csv(file_path5)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df_meta.shape
```

```
(6310, 61)
```

```
# Reorder columns in df_test
df_test = df_test.reindex(columns=df_meta.columns)

# Reorder columns in df_kaggle
df_kaggle = df_kaggle.reindex(columns=df_meta.columns)
```

```
# Extract features and target
X = df_meta.drop(columns=['activity', 'user_snippet',  'entropy_x', 'entropy_y', 'entropy_z'])  # Features (excluding 'activity' and 'us
y = df_meta['activity']
X_test_df = df_test.drop(columns=['activity', 'user_snippet', 'entropy_x', 'entropy_y', 'entropy_z'])
y_test_df = df_test['activity']
X_kaggle = df_kaggle.drop(columns=['user_snippet', 'activity' , 'entropy_x', 'entropy_y', 'entropy_z'])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train.head()
```

```
#X_train.corr()
```

```
# Feature scaling using standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled=scaler.fit_transform(X_test)
X_test_df_scaled = scaler.transform(X_test_df)
X_kaggle_scaled = scaler.transform(X_kaggle)
```

```
X_test_df_scaled .shape
```

```
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report
```

```
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier( random_state=123, class_weight='balanced')
```

Double-click (or enter) to edit

```
# Train the Random Forest classifier
rf_classifier.fit(X_train_scaled , y_train)
```

```
RandomForestClassifier(class_weight='balanced', random_state=123)
```

```
# Initialize selected features to include all features
selected_features = list(range(X_train_scaled.shape[1]))
```

```
# Initial performance evaluation
initial_accuracy = accuracy_score(y_test, rf_classifier.predict(X_test_scaled))
print("Initial Accuracy:", initial_accuracy)
```

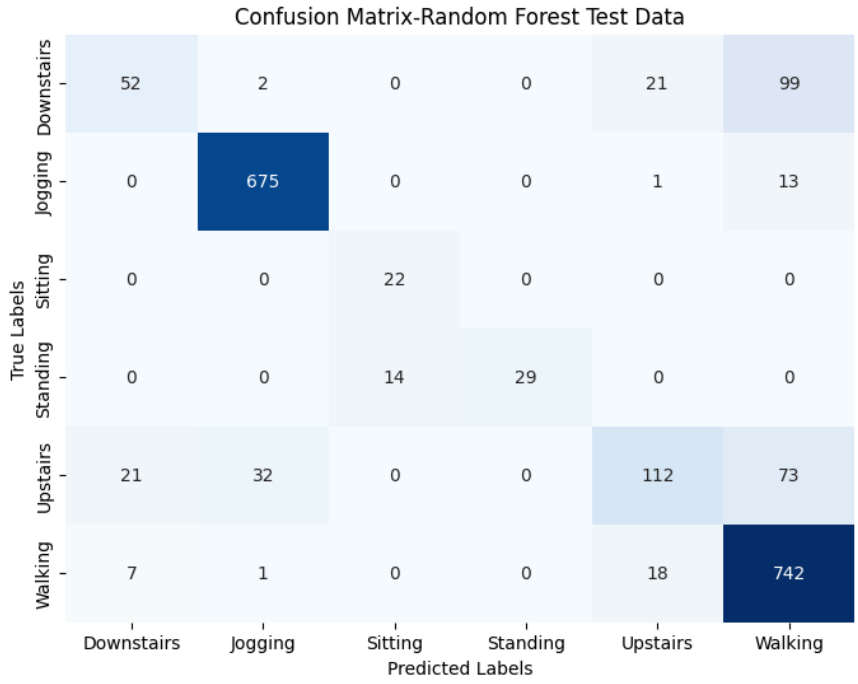
*Skip the Following two code chunks it will take long to complete its execution *

```
# Initialize selected_features with all features
selected_features = list(range(X_train_scaled.shape[1]))                              # it take long to run, you can intruput the execut

# Perform backward feature selection
while len(selected_features) > 1:
    # Track the best feature set and its performance
    best_feature_set = selected_features[:]
    best_accuracy = initial_accuracy

    # Iterate over each feature and evaluate its removal
    for feature_idx in selected_features:
        # Copy the feature set and remove the current feature
        reduced_feature_set = selected_features[:]
        reduced_feature_set.remove(feature_idx)

        # Train the Random Forest classifier with reduced feature set
        rf_classifier.fit(X_train_scaled[:, reduced_feature_set], y_train)

        # Evaluate performance on test set
        accuracy = accuracy_score(y_test, rf_classifier.predict(X_test_scaled[:, reduced_feature_set]))

        # Update the best feature set if the performance improves
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_feature_set = reduced_feature_set[:]

    # Update the selected features with the best feature set
    selected_features = best_feature_set[:]

    # Print the selected features and corresponding accuracy
    print("Selected Features:", selected_features)
    print("Test Accuracy:", best_accuracy)

# Final selected features
print("Final Selected Features:", selected_features)
```

```
selected_features = list(range(X_train_scaled.shape[1]))
feature_names = X_train.columns

# Extract names of selected features
selected_feature_names = [feature_names[i] for i in selected_features]

# Print the names of selected features
print("Selected Feature Names:", selected_feature_names)
```

Resum the Code execution from the following code chunk

```
# Extract selected features
selected_features = [0, 1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 19, 20, 21, 22, 23, 25, 27, 29, 30, 31, 33, 34, 35, 36, 37, 39, 40, 4
# Extract selected features from X_train_scaled, X_val_scaled, and X_test_scaled
X_train_selected = X_train_scaled[:, selected_features]
X_test_selected = X_test_scaled[:, selected_features]


# Train the Random Forest classifier with selected features
rf_classifier_selected = RandomForestClassifier(random_state=123, class_weight='balanced')
rf_classifier_selected.fit(X_train_selected, y_train)


# Evaluate performance on test set
test_accuracy = rf_classifier_selected.score(X_test_selected, y_test)  # validation
print("Test Accuracy:", test_accuracy)



X_test_df_selected = X_test_df_scaled[:, selected_features]    #update the test data based on selected fetures
X_test_df_selected.shape


y_test_df_pred = rf_classifier_selected.predict(X_test_df_selected)
# Evaluate performance on test set
test_accuracy = rf_classifier_selected.score(X_test_df_selected, y_test_df)  # validation
print("Test Accuracy:", test_accuracy)




from tabulate import tabulate
class_report_str = classification_report(y_test_df, y_test_df_pred)

# Convert the classification report string to a list of lines
class_report_lines = class_report_str.split('\n')

# Remove empty lines and headers
class_report_lines = [line.split() for line in class_report_lines if line.strip() and line.strip() != 'precision']

# Print the classification report table
headers = class_report_lines[0]
data = class_report_lines[1:]

print("Classification Report of Random Forest with Backward Feature Selection")
print(tabulate(data, headers=headers, tablefmt='grid'))



from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Get unique labels
labels = sorted(y_test_df.unique())

# Plot confusion matrix for test data
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test_df, y_test_df_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix-Random Forest Test Data")
plt.show()
```




```
X_kaggle_selected =X_kaggle_scaled[:, selected_features]
```

```
X_kaggle_selected.shape
```

```
#Make predictions on the Kaggle test data
predictions_kaggle = rf_classifier_selected.predict(X_kaggle_selected)
```

```
# Specify path within Google Drive
predictions_file_path = '/content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX3.csv'
#predictions_file_path = '/content/drive/My Drive/Course Dataset/SANDBOX3.csv'
# Save predictions for the Kaggle data to a CSV file
df_kaggle['prediction'] = predictions_kaggle
df_kaggle.to_csv(predictions_file_path, columns=['user_snippet', 'prediction'], index=False)

print("Predictions for Kaggle data saved to:", predictions_file_path)
```

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
rf_classifier = RandomForestClassifier(random_state=123)
param_grid = {
    'n_estimators': [ 40, 60, 70 ],
    'max_depth': [ 16],
    'min_samples_split': [2, 3],
    'min_samples_leaf': [3, 4]
}
```

```
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy')
```

```
grid_search.fit(X_train_selected, y_train)
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
best_params = grid_search.best_params_
best_score = grid_search.best_score_
best_params
```

```
{'max_depth': 18,
 'min_samples_leaf': 3,
 'min_samples_split': 2,
 'n_estimators': 60}
```

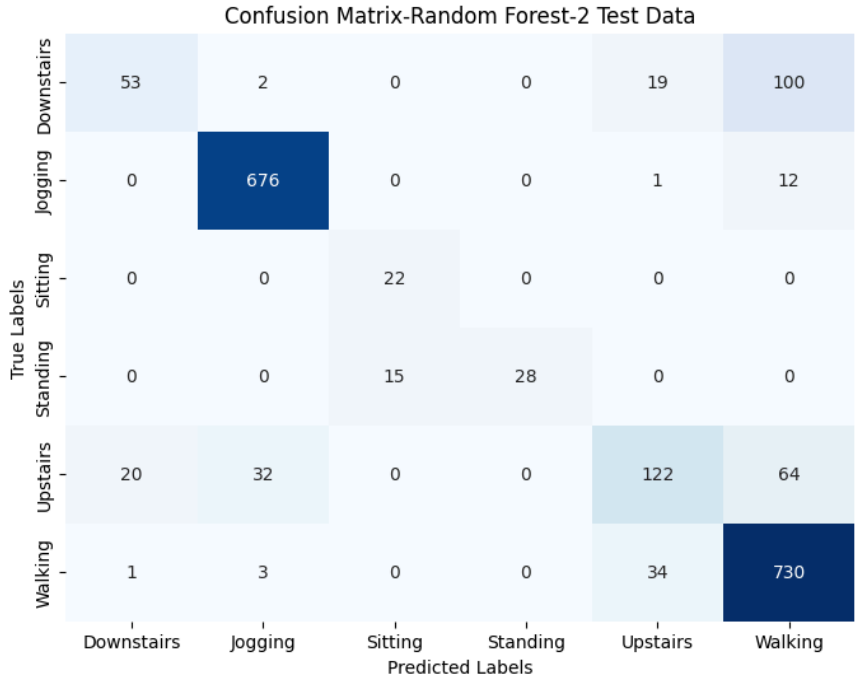
```
# Create a new RandomForestClassifier with the best parameters
best_rf_classifier = RandomForestClassifier(random_state=42, **best_params)
# Fit the model with the best parameters to the data
best_rf_classifier.fit(X_train_selected, y_train)
```

```
RandomForestClassifier(max_depth=18, min_samples_leaf=3, n_estimators=60,
                       random_state=42)
```

```
# Evaluate the model
accuracy = best_rf_classifier.score(X_test_selected, y_test)
print("Accuracy:", accuracy)
```

```
# Predictions on the unseen data
y_pred_test_bestrf = best_rf_classifier.predict(X_test_df_selected)

# Accuracy
accuracy = accuracy_score(y_test_df,y_pred_test_bestrf)
print("Accuracy:", accuracy)
```

```
# Get unique labels
labels = sorted(y_test_df.unique())

# Plot confusion matrix for test data
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test_df, y_pred_test_bestrf)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix-Random Forest-2 Test Data")
plt.show()
```



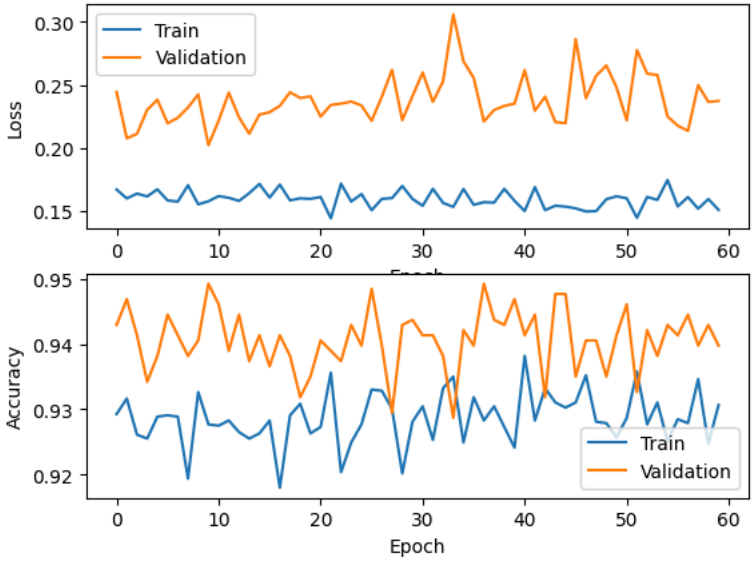

```
predictions_updated=best_rf_classifier.predict(X_kaggle_selected)
predictions_file_path = '/content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX4.csv'
# Save predictions for the Kaggle data to a CSV file
df_kaggle['prediction'] = predictions_updated
df_kaggle.to_csv(predictions_file_path, columns=['user_snippet', 'prediction'], index=False)

print("Predictions for Kaggle data saved to:", predictions_file_path)
```

```
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report
```

```
# Define input shape
input_shape = X_train_selected.shape[1:]

# Define number of classes X_train_selected, y_train
num_classes = 6
input_shape


# Define CNN Architecture
model = models.Sequential([
    layers.Reshape((X_train_selected.shape[1], 1), input_shape=(X_train_selected.shape[1],)),
    layers.Conv1D(64, kernel_size=3, activation='relu'),
    layers.MaxPooling1D(pool_size=2),
    layers.Flatten(),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])


from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Convert string labels to numerical labels
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
y_test_df_encoded = label_encoder.transform(y_test_df)


#  Train the Model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history_cnn=model.fit(X_train_selected, y_train_encoded, validation_data=(X_test_selected, y_test_encoded),  epochs=60, batch_size=32)
```

```
Epoch 57/60
158/158 [==============================] - 1s 5ms/step - loss: 0.1608 - accuracy: 0.9279 - val_loss: 0.2134 - val_accuracy: 0.944
Epoch 58/60
158/158 [==============================] - 1s 5ms/step - loss: 0.1515 - accuracy: 0.9346 - val_loss: 0.2498 - val_accuracy: 0.939
Epoch 59/60
158/158 [==============================] - 1s 5ms/step - loss: 0.1592 - accuracy: 0.9247 - val_loss: 0.2364 - val_accuracy: 0.942
Epoch 60/60
158/158 [==============================] - 1s 5ms/step - loss: 0.1506 - accuracy: 0.9307 - val_loss: 0.2372 - val_accuracy: 0.939
```

```
import matplotlib.pyplot as plt

def plot_history(history_cnn):
    plt.figure()
    plt.subplot(2, 1, 1)
    plt.plot(history_cnn.history['loss'])
    plt.plot(history_cnn.history['val_loss'])
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'])
    plt.subplot(2, 1, 2)
    plt.plot(history_cnn.history['accuracy'])
    plt.plot(history_cnn.history['val_accuracy'])
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'])
    plt.show()

plot_history(history_cnn)
```




```
from keras.utils import plot_model

# Plot the architecture
plot_model(model, to_file='model_architecture.png', show_shapes=True, show_layer_names=True,)


# Evaluate the model on the validation set
loss, accuracy = model.evaluate(X_test_df_selected, y_test_df_encoded)

# Print the evaluation results
print("Validation Loss:", loss)
print("Validation Accuracy:", accuracy)


labels = sorted(y_test_df.unique())
y_pred_cnn = np.argmax(model.predict(X_test_df_selected), axis=1)
# Plot confusion matrix for test data
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test_df_encoded, y_pred_cnn)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix-Random CNN Test Data")
plt.show()
```

```
predictions = model.predict(X_kaggle_selected)

# Print the predictions
print(predictions)


predicted_labels_encoded = np.argmax(predictions, axis=1)
predicted_labels = label_encoder.inverse_transform(predicted_labels_encoded)

# Print the predicted class labels
print(predicted_labels)
```

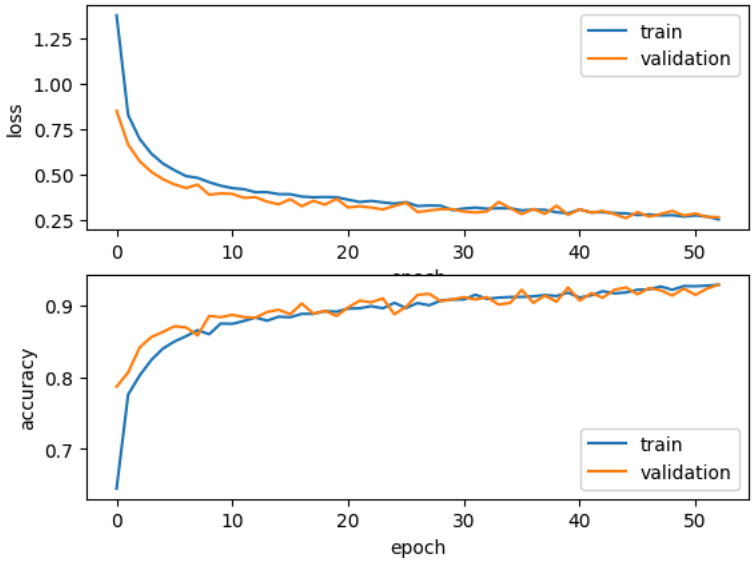
```
['Walking' 'Sitting' 'Upstairs' ... 'Jogging' 'Walking' 'Downstairs']
```

```
# Specify path within Google Drive
predictions_file_path = '/content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX7.csv' #zainab
#predictions_file_path = '/content/drive/My Drive/Course Dataset/predictions_sandbox.csv'
# Save predictions for the Kaggle data to a CSV file
df_kaggle['prediction'] = predicted_labels
df_kaggle.to_csv(predictions_file_path, columns=['user_snippet', 'prediction'], index=False)

print("Predictions for Kaggle data saved to:", predictions_file_path)


from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping

# Define CNN Architecture with L2 Regularization and Early Stopping
model_2 = models.Sequential([
    layers.Reshape((X_train_selected.shape[1], 1), input_shape=(X_test_selected.shape[1],)),
    layers.Conv1D(62, kernel_size=3, activation='relu'),
    layers.MaxPooling1D(pool_size=2),
    layers.Conv1D(32, kernel_size=3, activation='relu'),
    layers.MaxPooling1D(pool_size=2),
   # layers.Conv1D(32, kernel_size=3, activation='relu'),
   # layers.MaxPooling1D(pool_size=2),
   # layers.Conv1D(16, kernel_size=3, activation='relu'),
   # layers.MaxPooling1D(pool_size=2),
    layers.Flatten(),
    layers.Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(6, activation='softmax')
])


model_2.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])


from keras.utils import plot_model

# Plot the architecture
plot_model(model_2, to_file='model_architecture.png', show_shapes=True, show_layer_names=True,)


from tensorflow.keras.optimizers import Adam

# Define Early Stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)

# Train the model with Early Stopping
history = model_2.fit(X_train_selected, y_train_encoded, epochs=100, batch_size=32, validation_data=(X_test_selected, y_test_encoded), ca


model_2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape_4 (Reshape)         (None, 39, 1)             0

 conv1d_8 (Conv1D)           (None, 37, 62)            248

 max_pooling1d_8 (MaxPoolin  (None, 18, 62)            0
 g1D)

 conv1d_9 (Conv1D)           (None, 16, 32)            5984

 max_pooling1d_9 (MaxPoolin  (None, 8, 32)             0
 g1D)

 flatten_4 (Flatten)         (None, 256)               0
```

```
 dense_8 (Dense)             (None, 32)                8224

 dropout_4 (Dropout)         (None, 32)                0

 dense_9 (Dense)             (None, 6)                 198

=================================================================
Total params: 14654 (57.24 KB)
Trainable params: 14654 (57.24 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
def plot_history(history):
    plt.figure()
    plt.subplot(2,1,1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'])
    plt.subplot(2,1,2)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'])
    return;

plot_history(history)
```




```
loss, accuracy = model_2.evaluate(X_test_df_selected, y_test_df_encoded)

# Print the evaluation results
print("Validation Loss:", loss)
print("Validation Accuracy:", accuracy)


from sklearn.metrics import confusion_matrix
import numpy as np
y_pred_cnn2 = np.argmax(model_2.predict(X_test_df_selected), axis=1)
# Plot confusion matrix for test data
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test_df_encoded, y_pred_cnn)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix-Random CNN Test Data")
plt.show()


# Predict on the test data using the Keras model
y_pred_keras = np.argmax(model_2.predict(X_test_df_scaled), axis=1)
```

```
predictions2 = model_2.predict(X_kaggle_selected)
predicted_labels_encoded = np.argmax(predictions2, axis=1)
predicted_labels_2 = label_encoder.inverse_transform(predicted_labels_encoded)

# Print the predicted class labels
print(predicted_labels_2)


# Specify path within Google Drive
predictions_file_path = '/content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX8.csv' #zainab
# Save predictions for the Kaggle data to a CSV file
df_kaggle['prediction'] = predicted_labels_2
df_kaggle.to_csv(predictions_file_path, columns=['user_snippet', 'prediction'], index=False)

print("Predictions for Kaggle data saved to:", predictions_file_path)


from sklearn.ensemble import GradientBoostingClassifier
# Initialize the Gradient Boosting classifier
gbm_classifier = GradientBoostingClassifier()

# Train the classifier on the training data
gbm_classifier.fit(X_train_selected, y_train_encoded)




# Predict on the Validation data
y_pred_gbm = gbm_classifier.predict(X_test_selected)

# Evaluate the accuracy of the classifier
accuracy_gbm = accuracy_score(y_test_encoded, y_pred_gbm)
print("Accuracy (GBM):", accuracy_gbm)


# Predict on the test data
y_pred_gbm_test = gbm_classifier.predict(X_test_df_selected)

# Evaluate the accuracy of the classifier
accuracy_gbm = accuracy_score(y_test_df_encoded, y_pred_gbm_test)
print("Accuracy (GBM):", accuracy_gbm)


predictions_updated_gbm = gbm_classifier.predict(X_kaggle_selected)
predicted_labels_encoded = label_encoder.inverse_transform(predictions_updated_gbm)

predictions_file_path = '/content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX10.csv'
# Save predictions for the Kaggle data to a CSV file
df_kaggle['prediction'] = predicted_labels_encoded
df_kaggle.to_csv(predictions_file_path, columns=['user_snippet', 'prediction'], index=False)

print("Predictions for Kaggle data saved to:", predictions_file_path)


from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100],
    'learning_rate': [ 0.01, 0.8],
    'max_depth': [5, 8]
}


# Initialize the Gradient Boosting Classifier
gbm = GradientBoostingClassifier(random_state=42)
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=gbm, param_grid=param_grid, cv=5, scoring='accuracy')


# Perform grid search on the training data
grid_search.fit(X_train_selected, y_train_encoded)
```

```
# Get the best estimator from the grid search
best_gbm = grid_search.best_estimator_

# Print the best hyperparameters found
print("Best hyperparameters:", grid_search.best_params_)

# Evaluate the best model on the test data
y_pred_test_gbm = best_gbm.predict(X_test_selected)
accuracy_test_gbm = accuracy_score(y_test_encoded, y_pred_test_gbm)
print("Accuracy on test data (GBM):", accuracy_test_gbm)


# Predict on the test set
y_pred_va = best_gbm.predict(X_test_df_selected)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test_df_encoded, y_pred_va)
print("Accuracy:", accuracy)


from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score
gbm_pred = gbm_classifier.predict_proba(X_test_selected)
rf_pred = best_rf_classifier.predict_proba(X_test_selected)
cnn_pred = model.predict(X_test_selected)


# Combine predictions for soft voting
ensemble_pred = (gbm_pred + rf_pred + cnn_pred) / 3

# Get the predicted class with the highest probability
ensemble_pred_class = np.argmax(ensemble_pred, axis=1)


# Evaluate the accuracy of the ensemble classifier
accuracy_ensemble = accuracy_score(y_test_encoded, ensemble_pred_class)
print("Accuracy (Ensemble):", accuracy_ensemble)



gbm_pred_t = gbm_classifier.predict_proba(X_test_df_selected)
rf_pred_t = best_rf_classifier.predict_proba(X_test_df_selected)
cnn_pred_t = model.predict(X_test_df_selected)

# Combine predictions for soft voting
ensemble_pred_test = (gbm_pred_t + rf_pred_t + cnn_pred_t) / 3

# Get the predicted class with the highest probability
ensemble_pred_class = np.argmax(ensemble_pred_test, axis=1)
 #Evaluate the accuracy of the ensemble classifier
accuracy_ensemble = accuracy_score(y_test_df_encoded, ensemble_pred_class)
print("Accuracy (Ensemble):", accuracy_ensemble)


from sklearn.metrics import roc_curve, auc
labels = sorted(y_test_df.unique())
# Compute confusion matrix
cm = confusion_matrix(y_test_df_encoded, ensemble_pred_class)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for voting Classfier')
plt.show()


gbm_pred_kaggle = gbm_classifier.predict_proba(X_kaggle_selected)
rf_pred_kaggle = best_rf_classifier.predict_proba(X_kaggle_selected)
cnn_pred_kaggle = model.predict(X_kaggle_selected)

     79/79 [==============================] - 0s 2ms/step


# Combine predictions for soft voting
ensemble_pred_kaggle = (gbm_pred_kaggle + rf_pred_kaggle + cnn_pred_kaggle) / 3


# Get the predicted class with the highest probability
ensemble_pred_class_kaggle = np.argmax(ensemble_pred_kaggle, axis=1)

ensemble_pred_class_kaggle = label_encoder.inverse_transform(ensemble_pred_class_kaggle)
```

```
predictions_file_path = '/content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX11.csv'
# Save predictions for the Kaggle data to a CSV file
df_kaggle['prediction'] = ensemble_pred_class_kaggle
df_kaggle.to_csv(predictions_file_path, columns=['user_snippet', 'prediction'], index=False)

print("Predictions for Kaggle data saved to:", predictions_file_path)
```

```
Predictions for Kaggle data saved to: /content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX11.csv
```

```
import numpy as np
X_train_selected, y_train_encoded
# Reshape the training data
X_train_scaled_reshaped = np.reshape(X_train_selected, (X_train_selected.shape[0], 1, X_train_selected.shape[1]))

# Reshape the testing data
X_test_scaled_reshaped = np.reshape(X_test_selected, (X_test_selected.shape[0], 1, X_test_selected.shape[1]))
X_test_scaled_reshaped = np.reshape(X_test_selected, (X_test_selected.shape[0], 1, X_test_selected.shape[1]))
#y_test_reshaped = np.reshape(y_test_encoded, (y_test_encoded.shape[0], 1, y_test_encoded.shape[1]))

# Reshape the unseen data (X_test_df_scaled and X_kaggle_scaled)
X_test_df_scaled_reshaped = np.reshape(X_test_df_selected, (X_test_df_selected.shape[0], 1, X_test_df_selected.shape[1]))
#y_test_reshaped = np.reshape(y_test_df_encoded, (y_test_df_encoded.shape[0], 1, y_test_df_encoded.shape[1]))
X_kaggle_scaled_reshaped = np.reshape(X_kaggle_selected, (X_kaggle_selected.shape[0], 1, X_kaggle_selected.shape[1]))
```

```
X_train_scaled_reshaped.shape
```

```
X_test_scaled_reshaped.shpe
```

```
y_test_encoded.shape
```

```
y_test_encoded = y_test_encoded[:X_test_scaled_reshaped.shape[0]]
```

```
print("Shapes:")
print("X_train_scaled_reshaped:", X_train_scaled_reshaped.shape)
print("y_train_encoded:", y_train_encoded.shape)
print("X_test_scaled_reshaped:", X_test_scaled_reshaped.shape)
print("y_test_encoded:", y_test_encoded.shape)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Define the number of features
num_features = X_train_scaled_reshaped.shape[2]  # Number of features in your dataset
# Create the model
model = Sequential()
model.add(LSTM(units=15, input_shape=(X_train_scaled_reshaped.shape[1], num_features), return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(units=15))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Print model summary
model.summary()
```

```
history_lstm = model.fit(X_train_scaled_reshaped, y_train_encoded, epochs=80, batch_size=32, validation_data=(X_test_scaled_reshaped, y_t

# Evaluate the model on test data
loss, accuracy = model.evaluate(X_test_scaled_reshaped, y_test_encoded)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
# Plot training history

def plot_history(history_lstm):
    plt.figure()
    plt.subplot(2,1,1)
    plt.plot(history_lstm.history['loss'])
    plt.plot(history_lstm.history['val_loss'])
```

```
    plt.title(' LSTM Model Accuracy')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'])


    plt.subplot(2,1,2)
    plt.plot(history_lstm.history['accuracy'])
    plt.plot(history_lstm.history['val_accuracy'])

    plt.title('LSTM Model Loss')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'])

    return;

plot_history(history_lstm)


# Make predictions on X_test_df_scaled
y_test_df_pred= model.predict(X_test_df_scaled_reshaped)
# Calculate accuracy on X_test_df_scaled
y_test_df_pred_labels = np.argmax(y_test_df_pred, axis=1)

# Calculate accuracy on X_test_df_scaled
accuracy_test_df = accuracy_score(y_test_df_encoded, y_test_df_pred_labels)
print("Accuracy on X_test_df_scaled:", accuracy_test_df)
```

```
    61/61 [==============================] - 1s 3ms/step
    Accuracy on X_test_df_scaled: 0.8138572905894519
```

```
# Compute confusion matrix
# Convert probabilities to class labels
y_test_df_pred_labels = np.argmax(y_test_df_pred, axis=1)

# Compute confusion matrix
cm = confusion_matrix(y_test_df_encoded, y_test_df_pred_labels)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for LSTM')
plt.show()
```

```
 #Make predictions on X_test_df_scaled
Kaggle_df_pred_prob = model.predict(X_kaggle_scaled_reshaped)
Kaggle_df_pred = np.argmax(Kaggle_df_pred_prob, axis=1)
pred_class_kaggle = label_encoder.inverse_transform(Kaggle_df_pred)

predictions_file_path = '/content/drive/MyDrive/Machine Learning- Session/Course Work Project/SANDBOX22.csv'
# Save predictions for the Kaggle data to a CSV file
df_kaggle['prediction'] = pred_class_kaggle
df_kaggle.to_csv(predictions_file_path, columns=['user_snippet', 'prediction'], index=False)

print("Predictions for Kaggle data saved to:", predictions_file_path)
```

```
num_features
```

```
    59
```

```
from keras.layers import SimpleRNN

history_rnn = model_rnn.fit(X_train_scaled_reshaped, y_train_encoded, epochs=100, batch_size=32, validation_data=(X_test_scaled_reshaped

# Evaluate the model on test data
loss, accuracy = model.evaluate(X_test_scaled_reshaped, y_test_encoded)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)



# Make predictions on X_test_df_scaled
y_test_df_pred_r= model_rnn.predict(X_test_df_scaled_reshaped)
# Calculate accuracy on X_test_df_scaled
y_test_df_pred_labels_r = np.argmax(y_test_df_pred_r, axis=1)

# Calculate accuracy on X_test_df_scaled
accuracy_test_df = accuracy_score(y_test_df_encoded, y_test_df_pred_labels_r)
print("Accuracy on X_test_df_scaled:", accuracy_test_df)


def plot_history(history_rnn):
    plt.figure()
    plt.subplot(2,1,1)
    plt.plot(history_rnn.history['loss'])
    plt.plot(history_rnn.history['val_loss'])

    plt.title(' RNN Model Accuracy')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'])


    plt.subplot(2,1,2)
    plt.plot(history_rnn.history['accuracy'])
    plt.plot(history_rnn.history['val_accuracy'])

    plt.title('RNN Model Loss')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'])

    return;

plot_history(history_lstm)


# Convert probabilities to class labels
y_test_df_pred_labels = np.argmax(y_test_df_pred_r, axis=1)

# Compute confusion matrix
cm = confusion_matrix(y_test_df_encoded, y_test_df_pred_labels)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for RNN')
plt.show()



labels = np.unique(np.argmax(y_test_df_pred_r, axis=1))

plt.figure(figsize=(9, 6))
for label in labels:
    fpr, tpr, _ = roc_curve(y_test_df_encoded== label, y_test_df_pred_labels == label)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Class {label} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('RNN ROC Curve for Each Class')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```