Introduction

Human Activity Recognition (HAR) is a crucial computing branch ubiquitous in diverse applications, leveraging the widespread use of smartphones and devices equipped with sensors, accelerometers, and GPS. HAR mainly aims to enable the automatic recognition of human actions, gestures, or motions that are captured by sensors and processed using Machine-learning and signal processing techniques or other techniques depending on the application and data type being analyzed.

This report details modeling a supervised classification problem, using various machine-learning algorithms and configurations. The multiple outcomes resulted from the models were systematically evaluated and cross-compared to identify the most effective model for accurately detecting six different activities. Using the datasets provided, the conducted methodology includes stages of Exploratory data analysis, data pre-processing, modeling, evaluation and improvement, and finally a discussion of results and recommendations.

**Keywords:** HAR, ML, signal processing, Ensemble learning, CNN, Random Forest, XGBOOST, XGBM.

1. Methodology

# Problem Understanding

The objective of this coursework is to classify six different human activities consisting of walking, jogging, upstairs, downstairs, sitting, and standing by processing the provided datasets in their two forms; on one hand, we have the signals files which include the acceleration across three axes with a corresponding timestamp. On the other hand, the metadata files consist of pre-extracted statistical features from the signals data.

## Datasets Overview

* **Signals.csv** and **metadata.csv**: representing signals acceleration and pre-extracted statistical features dedicated to the model training.
* **signals\_test.csv, signals\_kaggle.csv, metadata\_test.csv, metadata\_kaggle.csv**: datasets designated for testing and validating predictions on the Kaggle platform.

### Approach Outline

1. **Exploratory Data Analysis (EDA):** This concerns the exploration of the data acquired to understand its structure and distribution using visualization techniques, descriptive statistics, correlation inspection, detecting inconsistencies in the data, and so forth.
2. **Pre-processing:** Handling missing values, and normalizing features if needed.
3. **Additional features extraction:** Concerns extracting additional features from the signal files to enrich the metadata files in capturing more information about the time-series data. the features were extracted across three phases including statistical, frequency, and correlation features.
4. **Machine learning modeling:** Train and test multiple ML models to determine the suitable algorithms for this HAR task, including hyperparameters tuning and evaluation of performance. The algorithms exploited are Random Forest, Ensemble method with voting classifier, CNN, and XGboost and each of these algorithms tested for three attempts corresponding to the three phases of the features extraction. At each attempt, additional features are provided then the accuracy performance is evaluated.
5. **Final model selection:** After cross-comparing results, the best-performing model based on the top accuracy score is selected.

The coding was done using Python including the following libraries:

*Numpy* and *Pandas* for data manipulation.

*Matplotlib* and *Seaborn* for visualization, and *Sklearn/TensorFlow* for the ML algorithms.

1. Exploratory Data Analysis

**Datasets Overview:**

The signals dataset contains the acceleration through the three axes (x, y, and z) captured at frequency sampling at a rate of one sample every 50 milliseconds (20 Hz), the axes are oriented into distinct directions with a value ranging between -20 and 20, a value of 10 = 1g = 9.81 m/s^2. So, it is a data of type signal time series. Each row is identified by a user\_snippet variable representing userID and snippetID concatenated into user\_snippet, which means that every user has many snippets.

A sample from the “signals.csv” dataset:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **user\_snippet** | **x-axis** | **y-axis** | **z-axis** | **Timestamp (ms)** |
| 525\_0 | 1.57 | -0.61 | -0.65 | 0.0 |
| 525\_0 | -1.12 | 1.84 | -1.46 | 50.0 |
| … | … | … | … | … |

The *signals* files are organized as follows:

*Signals.csv file*:Consists of 22 users where each has a different amount of snippets and each snippet is composed of many samples (sub-snippets).

*Signals\_test.csv file*:Consists of 6 users and each user has a different amount of snippets and each snippet is composed of many samples (sub-snippets).

*Signals\_kaggle.csv file*:Consists of 8 users, each one has a different amount of snippets and each snippet is composed of many samples (sub-snippets).

Note: For details about the number of snippets per distinct user, please check the code file (exploration part).

The *metadata* files are organized as follows:

*metadata.csv file*:Consists of pre-defined features identified by unique user\_snippet from the *signals.csv* file.

*metadata\_test.csv file*:Consists of pre-defined features identified by unique user\_snippet from the *signals\_test.csv* file.

*metadata\_kaggle.csv file*:Consists of pre-defined features identified by unique user\_snippet from the *signals\_kaggle.csv* file.

**Signals dataset**

**Descriptive statistics:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Signals.csv** | **x-axis** | **y-axis** | **z-axis** | **Timestamp** |
| **count** | 629866 | 629866 | 629866 | 629866 |
| **mean** | 0.766236 | 7.266391 | 0.293067 | 2473.450464 |
| **std** | 6.673741 | 6.924296 | 4.814493 | 1443.257538 |
| **min** | -19.61 | -19.61 | -19.80 | 0 |
| **25%** | -2.75 | 3.30 | -2.301839 | 1200 |
| **50%** | 0.46 | 8.12 | 0.040861 | 2450 |
| **75%** | 4.37 | 11.65 | 2.724070 | 3700 |
| **max** | 19.91 | 20.04 | 19.61 | 4950 |

From the above table, a higher mean characterizes the y-axis and approximate values of std between the x-axis and y-axis. A count of 629866 readings is captured by the accelerometer distributed across the users, also a snippet is reset to zero after each 4950 ms.

A snippet is composed of a bunch of frequent sub-snippets (rows), and this characterizes all snippets for all users. So, we have checked if the number of sub-snippets is consistent across all user\_snippets. As a result, found that there are significant gaps between some of them. The following table presents just a sample of the result:

|  |  |
| --- | --- |
| user\_snippet | Sub-snippets count |
| 804\_138 | 100 |
| 804\_12 | 20 |
| 771\_24 | 9 |
| 736\_284 | 3 |
| …. | …. |

The variation in the number of sub-snippets per user\_snippets is a subject of consideration as it may represent an imbalance between the classes that may bias the model towards classes with higher instances over those with fewer presence.

**Missing values:** no missing data in all datasets.

**Data visualization:** To have a look into patterns, trends, and variations that may provide indications for further additional feature extraction**,** the following plot visualizes a signal sample from each distinct activity.

A screenshot of a graph

Description automatically generated

Observations: Visually, some patterns can be identified that characterize the activities such as the low variability in the sitting and standing classes, also a higher fluctuation, and significant peaks for the y-axis in the activities downstairs and jogging. Also, it appears that some axes are correlated with others, for instance in the jogging plot, the y-axis seems to be negatively correlated with the x-axis. This will be inspected further in the exploration.

After combining the signal data of different user snippets grouped by activity classes, the following figure illustrates a boxplot for each activity signal combination in order to compare the groups, visualize the spread of the data, and detect potential outliers:

A group of graphs with different colored squares

Description automatically generated with medium confidence

Observations: The two classes upstairs and walking appear to have a quite similarity in terms of axes data distribution with a different interquartile value, while jogging and downstairs are characterized by a higher y-axis value which confirms the observations made previously on the signal plots. Thus, sitting and standing appear to have a contrary distribution of axes’ values compared to each other, for instance in sitting, the y-axis has lower interquartile values whereas in standing is the inverse. Alternatively, plenty of potential outliers are pointed which may be due to measurement inaccuracies or other forms of outliers.

**Correlation inspection:** The following heatmap figure inspects the correlation coefficients between the three axes but grouped by unique activity classes to inspect to what extent the axes are correlated to each other in different activities grouped together:

A screenshot of a graph

Description automatically generated

Observations: the significant positive coefficients are in standing activity between the y and x axes.

Alternatively, the following heatmaps concern a single sample from each activity class:

A screenshot of a graph

Description automatically generated

Observations: different sets of correlation coefficients characterize each activity as illustrated in the figure above.

**Distribution visualization:** in order to check the shape of the distribution for each activity class which is important to visualize what characterizes each distribution, the following figure present a Kernel Density Plots organized by the six activity classes:

A group of graphs showing different colored lines

Description automatically generated

. The distributions show indeed different shapes across the different activities, where a notable skewness and different kurtoses characterize some activities like walking on the three axes. Others are multimodal, so it would be relevant to consider these observations further in the feature engineering phase.

**Metadata datasets**

The *metadata.csv* dataset includes these features already extracted for each axis: *sum\_values, median, mean, length, standard\_deviation, variance, root\_mean\_square, maximum, absolute\_maximum, minimum.*

In order to check how many instances of each unique activity, check the subsequent bar plot:

A graph of activity classes

Description automatically generated

. Walking and jogging have the most instances in the metadata training set while sitting and standing have the fewer presence.

**Variables importance:** There are multiple statistical features in the metadata.csv file, where an inspection of the important features is crucial in identifying which variables contribute effectively and most significantly to the predictive power of the model, also this inspection enables more control over the features selection which leads to enhance model performance.

The process of feature importance we have used relies on training a random forest classifier that learns the relationships between features and targets the activity variable. After training, the contribution of each feature in the prediction was extracted in terms of a relative score that represents the impact of that variable in reducing the model’s error when predicting.

So, the next plot highlights the important features in our metadata set:

A graph showing a number of different colored bars

Description automatically generated with medium confidence

Observations: The length features are completely useless, while std, variance, min, max, and absolute min/max features show the most pertinent contribution to the predictions.

1. **Pre-processing**

**Normalization:** a common scale for the features is required for the classification. For this, the *StandardScaler* imported from the *sklearn.preprocessing* library is used to transform the features distribution to have a mean equal to zero and a standard deviation of 1.

**Label encoder:** the target variable concerned in this project is of categorical type, which requires encoding it into a numerical value. This is done using LabelEncoder from the sklearn.preprocessing library.

1. **Additional Features Extraction**

Taking into consideration the observations noted during the exploration phase, here is a summary of what features to consider for extraction:

* From the data visualization part, significant peaks were noticed.
* From the box plots, different interquartile values are relevant to discriminate between classes.
* From the KDE plots, skewness and kurtosis are pertinent in characterizing class distribution.
* From the correlation inspection, significant correlations were captured between the axes of the signal data.

The features were extracted in three sequential phases, where at each phase I extracted a set of different features from the signal data and merged them up to the metadata datasets. Overall, 25 additional features were taken.

The first phase of feature extraction:

Skewness: which enables the model to discriminate the inclination of the distribution which may characterize certain activity classes.

Kurtosis: as observed in the density plots, certain activities like walking and jogging have heavy tails compared to the other activities.

iqr, percentile\_10, percentile\_90: statistical measurement of the data dispersion.

num\_peaks: captures intensive accelerations or cyclic patterns in the signal data.

2. The second phase of feature extraction:

* Root Mean Square (RMS): extracted to measure the magnitude of varying signal, this would contribute in quantifying the overall energy content in the signal.
* Energy: calculated as sum of the squared values of the signal to measure the total power of the signal, which we deemed this is useful in enabling the model to distinguish between high and low activities in terms of energy.
* Dominant Frequency: important in determining the dominant frequency of the signal after decomposing it into frequency components. This helps discriminate activities based on signals in terms of frequency-domain features.
* Spectral Entropy: measures the distribution of power across different frequency components.
* Correlation between axes (xy, xz, yz): helps identifying patterns in terms of cross-axes correlations.
* Signal Magnitude Area (SMA): sum of the magnitudes of the signal values across all axes.
* Autocorrelation: useful in measuring the similarity of the signal with delayed lags, to identify cyclic or periodic patterns.

3. The third and final phase of feature extraction:

Correlation Between Axes:

xy\_correlation: Correlation coefficient between the x-axis and y-axis.

xz\_correlation: Correlation coefficient between the x-axis and z-axis.

yz\_correlation: Correlation coefficient between the y-axis and z-axis.

Cross-Correlation Features:

xy\_cross\_corr\_max: Maximum value of the cross-correlation between the x-axis and y-axis.

xz\_cross\_corr\_max: Maximum value of the cross-correlation between the x-axis and z-axis.

yz\_cross\_corr\_max: Maximum value of the cross-correlation between the y-axis and z-axis.

xy\_cross\_corr\_mean: Mean value of the cross-correlation between the x-axis and y-axis.

xz\_cross\_corr\_mean: Mean value of the cross-correlation between the x-axis and z-axis.

yz\_cross\_corr\_mean: Mean value of the cross-correlation between the y-axis and z-axis.

xy\_cross\_corr\_std: Standard deviation of the cross-correlation between the x-axis and y-axis.

xz\_cross\_corr\_std: Standard deviation of the cross-correlation between the x-axis and z-axis.

yz\_cross\_corr\_std: Standard deviation of the cross-correlation between the y-axis and z-axis.

The features extracted from the three phases were employed in the model in a sequential way, which means that each model is fed gradually with the features collected through the phases, this helps assess the model performance by having control over the features provided and the hyperparameters values used to the model.

1. **Machine learning Modeling**

After exploring, pre-processing, and extracting features, it is time to build a model classifier, I have exploited multiple algorithms including the tree-based ones such as the Random Forest classifier and the Extreme Gradient Boosting (XGBoost), and also used the Neural network -based algorithms like convolutional neural network (CNN). Moreover, used the ensemble methods with voting classifiers including ‘RF, GB, and SVM’. All of these models have experienced different configurations to achieve their possible effectiveness in predicting.

The datasets used to feed the algorithms are the *metadata files* including the additional features that we have extracted. The process of modeling the different algorithms went sequentially, where at every attempt, we implemented one algorithm, tuned its hyperparameters, and evaluated its outcomes until it reached the best possible accuracy score, then moved on other remaining algorithms. At the end, picking the best performing one in terms of accurate predictions.

1. **Random Forest algorithm with hyperparameter tunning:**

**Using the metadata + the first phase of features extraction**: The datasets used in this model include the pre-extracted features in addition to the features I have extracted during the first phase which are the skewness, kurtoses, iqt, and percentiles. This model employs the random forest algorithm and to identify the best hyperparameters, I have defined a range of values for the hyperparameters max\_depth, number of estimators, min samples leaf, and min sample split. Thus, created a function evaluate\_model that trains the model iteratively using the different values in the range, for each iteration a prediction is made, and the accuracy score is calculated on the test set, therefore comparing the scores to each other and pick the best performant ones to use in tuning the model. The following figure shows the result of this process:

**A graph of a graph

Description automatically generated with medium confidence**

. The best number of estimators found is equal to 200, and a max\_depth is equal to 15. The other hyperparameters remain ‘none’.Taking this into consideration, after training and predicting on the test set, the model gave 83% accuracy while predicting on the metadata\_Kaggle file and after submission, the model got an accuracy of 84%

**Using the features extracted in the second phase:** the features used in this attempt include the previous features in addition to the features extracted during the second phase of extraction as described in the EDA part. Using the RF model fed with the updated datasets,

The model predicted 86% accuracy on the test set, and 87% accuracy on the Kaggle set. This shows that the additional features extracted during the second phase of extraction have improved the classifier's performance.

**Using the final phase of features extraction:** Now including the correlation and cross-correlation features into the RF model, a result of 87 % accuracy on the Kaggle predictions with no enhancement to the previous score.

**Ensemble methods with Voting Classifier:**

**Using the metadata + the first phase of features extraction:** In our approach to improve the model performance, we utilized the ensemble learning techniques. Specifically, we implemented a Voting Classifier, which combines the predictions from multiple different classifiers to produce a final prediction. This method leverages the strengths of various models, potentially leading to better performance than any individual model.

The selected base models for the ensemble are: Random Forest, Gradient Boosting Classifier (GBM), Support Vector Machines (SVM). Each of these models brings unique strengths to the ensemble. For instance, RF is effective at handling a variety of feature types and can model complex interactions. Gradient Boosting performs in reducing bias and can model complex patterns by focusing on correcting previous errors. While the SVM is useful for classification problems where the decision boundary is complex and non-linear. The aggregation of the three outputs are handled using the voting classifier (soft voting) which averages the predicted probabilities from each base model, to give the final prediction.

As a result, the accuracy prediction score with the voting classifier was 85% on the test set, while on the Kaggle file achieved 86%. The accuracy has increased using this model or ensemble learning compared to the random forest alone.

**Using the features extracted from the second extraction phase:** In this case, the same model configuration fed with the additional features resulted in 86% accurate predictions on the test set and 88% accuracy score on the Kaggle file. One more time the additional features show their positive contribution to more accurate classification.

**Using the final phase of features extraction:** Now including the correlation and cross-correlation features into the ensemble model, a result of 88.191 % accuracy on the Kaggle predictions. A one percent increase in the accuracy percentage compared to previous models.

**Convolutional Neural Network (CNN):** CNNs are powerful models, particularly for tasks involving spatial data, but can also be effective for structured data with feature engineering. After loading the datasets that contain all the features extracted during the three phases of extraction, the target variable “activity” was encoded using LabelEncoder to convert into a numerical value. Thus, features were standardized using StandardScaler to ensure they have a mean of 0 and a standard deviation of 1, which helps in faster convergence during training. Also, the target variable was converted to one-hot encoded format using TensorFlow's utility functions, making it suitable for multi-class classification.

**Model architecture:**

* Input Layer: Receives standardized features.
* Dense layers: Two dense layers with ReLU activation functions.
* Output Layer: A dense layer with the number of neurons equal to the number of classes, using the softmax activation function.

The loss function used is categorical cross-entropy. While the optimizer used is Adam with a learning rate of 0.001. A batch size of 32, and 50 epochs.

Hyperparameter Tuning: To achieve optimal performance of this CNN model, I used a set of values for each parameter and then applied the random search (RandomizedSearchCV) to identify the best combination of hyperparameter values that maximized the performance. I have avoided using grid search due to the lengthy execution time and the high computational cost. The set of hyperparameters that were considered for random search are:

* Number of neurons in the first dense layer (neurons\_1): [64, 128, 256]
* Number of neurons in the second dense layer (neurons\_2): [32, 64, 128]
* Learning rate for the Adam optimizer (learning\_rate): [0.01, 0.001, 0.0001]
* Batch size (batch\_size): [16, 32, 64]
* Number of epochs (epochs): [30, 50, 100]

For the evaluation of the model outcomes, the accuracy score, model loss, and accuracy evolution plots are generated.

The following visuals present tracking the accuracy and the loss over epochs which visualize the model’s learning evolution:

A comparison of graphs with numbers

Description automatically generated with medium confidence

According to the accuracy model plot, it is clear that as the number of epochs increases the training accuracy is performing very well by achieving 0.96 in learning, but when it comes to the validation accuracy curve, it increases as well but still the accuracy score (0.77) is lower compared to the training one, this indicates that this model is struggling to generalize accurately in unseen data. This is confirmed by the loss model plot, where the loss values of the training are doing well by consistently decreasing while on the unseen data during the 5 initial epochs, the model is learning, after that starts increasing the loss values which justify that the model is overfitting the training data.

This model resulted in a 0.87035 accuracy in predicting the Kaggle dataset.

**Extreme Gradient Boosting (XGBoost):** After all previous attempts, I decided to use the XGboost algorithm which is known for achieving high-accuracy predictions. It is an ensemble learning algorithm that relies on boosting by combining predictions from weak learners sequentially.

As the first attempt, the model produced a higher accuracy of 90% on the Kaggle file, and 89% on the test set. So, start using multiple configurations randomly searching the optimal values of the learning rate, number of estimators, maximum depth of trees, eval\_metric, subsampling ratio, L1 and L2 regularization. As a result of using the optimal hyperparameter values detected, the accuracy increased to 90.586 % which is the highest score achieved among all other models we have built.

The model is initialized with the following hyperparameters:

{Learning Rate: 0.1

Number of Estimators: 2500

Maximum Depth of Trees: 5

Subsampling Ratio: 0.8

Column Subsampling Ratio by Tree: 0.8

Gamma: 0

Regularization Alpha: 0

Regularization Lambda: 1}

In order to check how the model is performing in predicting the true classes, a confusion matrix is produced from the predictions done on the test set:

A chart with numbers and a blue square

Description automatically generated

We can see that the model is confused mostly between the classes walking and downstairs, downstairs and upstairs, jogging and upstairs. Other few confusions were made but the truest labels that the model failed to predict accurately were upstairs and downstairs.

**Final Model Selection:** According to the outcomes of the models used, the following table organizes the resulted accuracy scores on the metadata\_kaggle dataset:

|  |  |  |  |
| --- | --- | --- | --- |
| **Models** | **Phase 1 of extracted features** | **Phase 2 of extracted features** | **Phase 3 of extracted features** |
| **Random forest** | 84% | 87% | 87% |
| **Ensemble model** | 86% | 88% | 88.191% |
| **CNN** | 87% | 88.1% | 87.035% |
| **XGBoost** | 85% | 89.9% | **90.586%** |

So, given the comparative table, the XGboost model outperformed all other models, making it the preferred and selected model for this classification task.

1. **Results Discussion**

The modeling process included various ML algorithms with different feature sets, and each of the models have been assessed for its effectiveness in classifying human activities accurately. The Random Forest classifier, Gradient Boosting, and Support Vector Machines were employed individually and within an ensemble learning framework. While each algorithm demonstrated promising performance. As a result, the XGBoost model outperformed the other models with an accuracy score of 90.586% for predicting the activities on the Kaggle datasets. This highlights the suitability of ensemble learning and boosted algorithms dealing with HAR tasks.

**Feature Engineering Impact:**

Feature extraction played a crucial role in improving the performance, by incorporating additional features such as Root Mean Square (RMS), dominant frequency, and Cross-Correlation Features, the classifiers demonstrated improved predictive power which emphasizes the importance of feature engineering contributing to the modeling effectiveness.

Despite the considerable outcomes of the models, challenges were encountered during the project. The imbalance in the distribution of activity classes impacted the model training and generalization. Addressing class imbalance through data augmentation techniques or algorithmic adjustments could improve model robustness in future modeling. Additionally, the overfitting issue observed in the Convolutional Neural Network (CNN) model underscores the importance of regularization techniques in controlling the data fitting.

**Recommendations**

For future analysis, I suggest prioritizing the analysis of outliers present in the signal data. Given that accelerometers are vulnerable to capturing noise or device inaccuracies that may impact model performance, a thorough examination of these outliers could provide valuable insights into potential sources of variability and help enhance model robustness. Also, modeling the signal datasets by relevant algorithms suitable for time series data such as LSTM can be an alternative approach to consider.

END