# INFR11150 PDIoT Coursework 3 (2022-23)
# Final Report: End-to-end Human Activity Recognition and Monitoring System

**GROUP Y**
**Passara Chanchotisatien (s2255740)**
**Joseph Moncrieff (s1732873)**
**Pradnesh Sanderan (s1956488)**

**Abstract**

Human activity recognition (HAR) is a rapidly growing field in computer vision and machine learning. In this report, we propose a real-time HAR system using a mobile application. The system is designed to recognize and classify a total of 14 physical activities using the accelerometer and gyroscope sensors from the RESpeck and Thingy devices. The system uses a dimensionality reduction method-SPCA and machine learning classifiers RandomForest and LightGBM, trained on a dataset of labeled activity data, to make predictions about the current activity of the user. The mobile application allows for easy data collection and real-time activity tracking, providing users with a convenient and personalized way to track their physical activity. The system is also capable of tracking step counts of the user. Experiment results of both offline and online classification performance show that the proposed system achieves high accuracy and is able to run in real-time on a mobile device.

Date: Thursday 19$^{th}$ January, 2023

# 1 Introduction

## 1.1 Project Aims

Human activity recognition (HAR) systems are computer-based systems that are designed to recognize and interpret the actions and behaviors of individuals. These systems use various sensors and machine learning algorithms to interpret data and identify patterns in human activity, and can be used in a wide range of applications. One of the most important applications of HAR systems is in the field of healthcare, where they can be used to monitor patients and track their health status. For example, a HAR system equipped with sensors can detect changes in a patient's activity levels and alert healthcare professionals of potential health issues. This allows for early intervention and can help to prevent serious health problems from developing. The importance of human activity recognition systems lies in their ability to recognize and interpret human behavior, allowing for the automated monitoring of individuals in a variety of settings.

The aim of this project is to implement an end-to-end system consisting of sensors, a machine learning model to classify physical activities in real-time, and a mobile application which processes and displays incoming sensor data and classified activities and keeps a history of physical activities and step counts performed by the user in each day.

## 1.2 Methodology Description

### 1.2.1 HAR Classifier

Various combinations of dimensionality reduction methods (PCA, iPCA, SPCA), traditional machine learning classifiers (KNN, DT, RF, HCRF, ExtraTree, Ridge, SVM, XGBoost, LightGBM), and deep learning methods (1D-CNN, LSTM, CNN-LSTM, ConvLSTM) were tested to see which combination would achieve the best classification performance in terms of accuracy, precision, recall, and f1 scores. After thorough experimentation and obtained classification performance results, the best models are: Sparse Principal Component Analysis (SPCA)→LightGBM and SPCA→Random Forest (RF). These models are then deployed in the cloud to be used in the mobile application.

Sparse Principal Component Analysis (sparse PCA) is a variation of PCA that adds a sparsity constraint to the traditional PCA optimization problem. The goal is to find a low-dimensional representation of the data with sparse principal components. This sparsity constraint helps identify a smaller number of relevant features in the data. The sparsity constraint is achieved by adding a sparsity-inducing penalty term to the PCA optimization problem.

A Random Forest is an ensemble learning method that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. LightGBM builds a tree in a leaf-wise manner, which is different from the traditional level-wise approach used by most other tree-based algorithms. This makes LightGBM more efficient and faster than other methods, especially for datasets with a large number of features. Additionally, LightGBM also supports parallel and GPU learning, which further improves its efficiency.

### 1.2.2 Step Counting Algorithm

After, a classification result has been derived from the HAR classifier, a step counting algorithm is applied (depending on classification result). The algorithm utilizes Fast Fourier Transform (FFT) and thresholding to count the number of steps. Steps are detected through peak detection in incoming acceleration data. A low pass filter is applied to the data to reduce high frequency noise in order to reduce the number of false positives when counting the number of steps.

### 1.2.3 Model Deployment and Mobile Application

The app utilizes Firebase to authenticate users and store user data. The data stored in the Firebase Real-time Database includes the classified activities and step count for each date and time. All machine learning models are deployed using Flask and the classification API from Google Cloud Platform, which outputs the classification result after fed data from the sensors. All data and Machine Learning models are stored on the cloud, making the mobile application extremely lightweight.

Table 1: Table of Classified Activities in Each Model

| Model | Activities |
|---|---|
| 4-class Model | (1) Sitting/Standing, (2) Lying, (3) Walking, (4) Running |
| 8-class Model | (1) Sitting, (2) Standing, (3) Lying, (4) Walking,<br>(5) Running, (6) Ascending, (7) Descending, (8) Movement |
| 14-class Model | (1) sitting straight, (2), sitting bent forward, (3), sitting bent backward,<br>(4) standing, (5) lying down on left side, (6) lying down on right side,<br>(7) lying down on the back, (8) lying down on stomach,<br>(9) walking, (10) running/jogging, (11) ascending stairs,<br>(12) descending stairs, (13) desk work, (14) movement |

Table 2: Summary of offline classification results

| Sensor | Model | Accuracy (%) | Precision (%) | Recall (%) | F1 (%) |
|---|---|---|---|---|---|
| RESpeck (SPCA,RF) | 4-class | 0.9909 | 0.9919 | 0.9909 | 0.9914 |
| | 8-class | 0.9732 | 0.9737 | 0.9732 | 0.9733 |
| | 14-class | 0.9492 | 0.9490 | 0.9497 | 0.9491 |
| Thingy (SPCA,LightGBM) | 4-class | 0.9870 | 0.9747 | 0.9908 | 0.9825 |
| | 8-class | 0.9746 | 0.9749 | 0.9746 | 0.9746 |
| | 14-class | 0.9591 | 0.9593 | 0.9590 | 0.9589 |

## 1.3 Classified Activities

A total of fourteen physical activities can be classified by the Machine Learning model including (1) sitting straight, (2), sitting bent forward, (3), sitting bent backward, (4) standing, (5) lying

down on left side, (6) lying down on right side, (7) lying down on the back, (8) lying down on stomach, (9) walking, (10) running/jogging, (11) ascending stairs, (12) descending stairs, (13) desk work, (14) miscellaneous movement (sudden turns, bending down, getting up from chairs, movement in between activities). Three types of models were implemented to classify activities from incoming data from the RESpeck and Thingy sensors including the 4-class model, 8-class model, and 14-class model. The activities classified by each model can be seen in **??**.

### 1.4 Results Summary

A table depicting the summary of the offline classification performance results for the 4-class model, 8-class model, and 14-class model for each sensor can be seen in **??**. The model with the highest accuracy to classify physical activities from the RESpeck and Thingy sensor are the SPCA→LightGBM and SPCA→RF, respectively.

## 2 Literature Review of HAR Methods

Human activity recognition (HAR) is the task of recognizing the type of physical activity being performed by a person from sensor data. In this project, devices (RESpeck and Thingy) which were used to collect data utilize inertial sensors (accelerometer and gyroscope). As a result, this literature review will primarily focus on and discuss the state-of-the-art methods for HAR using inertial sensors. The implementation process for HAR typically consists of four steps including (1) data collection and pre-processing, (2) data processing and feature extraction, (3) building the classifier, and (4) performance results. The following subsections will discuss various methods utilized by different studies in each step.

### 2.1 Data Collection and Pre-processing

The main segmentation techniques employed by researchers include activity-defined windows, event-defined windows, and sliding windows. The method of determining activity-based windows involves dividing up the stream of sensor data by identifying when an activity changes. The starting and ending points of each activity are established before identifying the specific activities. For example, variations in frequency can be used to detect changes between activities. Sekine et al. suggested a wavelet decomposition model to detect frequency changes for three types of walking in a continuous record. A similar method is used in a referenced study, but only a portion of the activity window is utilized for classification. To enhance the detection of activity changes, Yoshizawa et al. proposed a heuristic method that separates static and dynamic actions.

Some activities may be more easily recognized as a series of movements or actions that are performed in a specific order. This is the case with infrequent activities, such as household tasks (e.g. meal preparation, cleaning), where the activity or gesture takes place infrequently and is interspersed with other activities or gestures. For recognizing gestures or detecting isolated movements, it is particularly useful to identify specific events. The event-based approach involves identifying specific events, which are then used to divide up the data. Since the events may not be evenly distributed in time, the size of the corresponding windows is not fixed. Gait analysis has particularly benefited from this type of analysis.

The sliding window technique, referred to as "windowing," is the most commonly used method for segmenting activities in recognition. Its ease of implementation and lack of pre-processing

makes it well-suited for real-time applications. In this technique, signals are divided into windows of a fixed size without any gaps between them. Although, in some applications, an overlap between neighboring windows is allowed, but it is not as commonly used. Different lengths for windows were implemented in studies ranging from 0.1s to 12.8s. Banos et al conducted a study which investigates the effect of different window sizes on activity classification performance. It was found that irregular or sporadic activities will require a more convoluted process for segmenting windows. Results have shown that a window size of approximately 1-2s will provide the best classification performance. Table **??** shows the list of studies with their respective segmentation approaches.

## 2.2 Data Processing and Feature Extraction

Upon completion of the data cleaning and pre-processing stage, various feature extraction and data processing techniques will be applied. Many studies that use traditional machine learning classifiers will opt to create manually-crafted features. This is not seen as often in deep learning algorithms as deep learning methods are known to automatically perform feature extraction in the process.

Typical features can be categorized into two types: time-domain features, such as mean, standard deviation, and correlation within the window, and frequency-domain features, which are obtained after performing a Fast Fourier Transform (FFT) on the window. The frequency-domain features include entropy, energy, and coherence (correlation in the frequency domain). Using a short window length allows for almost real-time inference of the user's current activity and enables the detection to quickly adapt to changes.

San-Segundo et al. approach the problem of recognizing activities based on acceleration data using strategies commonly used in speech processing, such as Mel Frequency Cepstral Coefficients (MFCCs) and Perceptual Linear Predictions (PLPs) coefficients. They extract a total of 561 time-domain and frequency-domain features. While this approach is noteworthy, it is worth noting that the MFCC features were modeled after mimicking the way the human ear responds to sound, thus the practical application of these features in activity recognition is unclear.

Not all features are equally effective in differentiating activities. Feature selection methods, such as filter, wrapper, or embedded selection, can be used to decrease the number of features. For instance, Dallas et al. used Relief-F, a filter-based method, to select accelerometer features for activity recognition. Other methods like Principal Component Analysis (PCA) are also used to project the original features into a lower-dimensional subspace with uncorrelated components. Decreasing the number of features significantly reduces the computational effort required for the classification process.

A recent study demonstrated that even simple histogram-like features can achieve good performance on various datasets. It would be valuable to more extensively evaluate these features against other types of features that have been mentioned. The statistical features that were extracted were extensive, but many feature sets that are widely used in the field were not included.

Using only simple features has the advantage of requiring minimal computation, which allows for low-power feature extraction on the sensor device before transmission. This idea was thoroughly explored in a study, where the authors presented a comparative performance evaluation of a large number of features from acceleration data computed on embedded hardware platforms. The features were evaluated based on cost and accuracy, and the study concluded that simple time-domain features computed in fixed-point arithmetic had the best cost-to-accuracy ratio.

The results showed that computing and transmitting a few of these time-domain features instead of sending the full acceleration data can reduce energy consumption by a significant amount while still maintaining acceptable accuracy. Other research in this area includes studies where low-complexity estimators of complex features are efficiently produced from carefully designed neural networks.

## 2.3 Classification Algorithms

Traditional machine learning methods and deep learning methods are both commonly used for HAR, but they have some key differences. Traditional machine learning methods rely on hand-crafted features and rely on domain knowledge to extract those features. These methods are typically based on models such as decision trees, k-nearest neighbors, support vector machines, and random forests. These models are relatively simple and easy to interpret, but they may not be able to handle large amounts of data or complex sensor data. Deep learning methods, on the other hand, rely on neural networks, which are able to automatically learn features from raw sensor data. These methods include convolutional neural networks (CNNs) and recurrent neural networks (RNNs) which are able to handle large amounts of data and complex sensor data. These models are more powerful than traditional machine learning methods, but they can be more difficult to interpret. In the following subsections, we will outline the various traditional machine learning methods and deep learning methods used in different studies.

### 2.3.1 Traditional Machine Learning Classifiers

Traditional machine learning methods have been widely used in Human Activity Recognition (HAR) to model and classify human activities from sensor data. Some commonly used traditional ML methods that have been proven to achieve good classification performance results include Random Forest (RF), Support Vector Machines (SVMs), Decision Trees (DT), LightGBM, and so on.

The LightGBM method has been shown to This method combines information from the user and their surrounding environment with computer-based data, and uses a smartphone's inertial sensors to collect action information. The LightGBM algorithm has been shown to have a higher accuracy rate than other algorithms and can more accurately identify various actions. It was shown that the LightGBM method had an accuracy of 96% and an error of only 6.84%. In Har, similar results were obtained with other classifiers but none of them had as high of overall accuracy [1].

The RandomForest method was another method proven to show great accuracy when used as a classifier. When used with sensors in a smartphone in a Smartphone-Based Human Activities Recognition System, the accuracy of the system achieved an accuracy of 98.34% [2].In this system, the Random Forest Algorithm (RFA) is used as a classifier that has several decision trees on subclasses of the dataset. The RFA takes the average to enhance the projecting accuracy of the given dataset. When compared with K-nearest neighbours and support vector machines, the RFAl gave better accuracy. In other studies, where induction-based motion signals were used, it was shown that the Random Forest algorithm again produced the best results with an accuracy of 91.5% [3].

In addition to traditional machine learning methods such as LightGBM and Random Forest, other methods such as Support Vector Machine (SVM) and Deep Learning (DL) methods like Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are also

being used in human activity recognition research [4]. However, it has been found that neither TML nor DL is suitable for all kinds of situations and various datasets. Different sizes of collected HAR datasets may produce different influences on the effectiveness of traditional machine learning methods as well as deep learning architectures. It was shown that when dealing with smaller datasets, TML structures were more suitable and when using a larger dataset, DL approaches such as CNN and LSTM are more sensible.

It was shown in some studies that ensemble ML classifiers are superior to traditional ones like KNN, DT, and Naive Bayes. The experiment found that the Adaboost-RF classifier gave an overall accuracy rate of 98.07% for the holdout method and 98.82% for the 10-fold cross-validation method [5]. Another experiment showed that XGBoost, when incorporated with the mutual information method, gave the best performance of a classifier whose accuracy, precision, recall, and F1 scores are 0.9122, 0.8727, 0.8629, and 0.8640, respectively [6].

### 2.3.2 Deep Learning Algorithms

Deep learning methods have become increasingly popular for Human Activity Recognition (HAR) in recent years, and several state-of-the-art methods have been developed. Some of the most popular deep learning methods for HAR include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) Networks, Attention-based Models, and Transformer-based models.

Mekruksavanich and colleagues examined how well different deep learning techniques can identify everyday human activities and created a deep residual neural network called the ResNeXt model that uses aggregated multi-branch transformation to improve identification performance. To test its performance, they compared it to three standard DL models (CNN, LSTM, and CNN-LSTM) using a standard dataset called the Daily Living Activity dataset. This dataset includes mobility signal data collected from sensors on the wrist, hip, and ankle. The results showed that the proposed model outperforms other benchmark DL models in terms of accuracy and F1-scores. Additionally, the ResNeXt model achieved better performance with fewer training parameters than the other models.

In another study, Rojanavasu et al. conducted a comparative study to investigate the difference in performance between variations of CNNs and LSTMs. The baseline models used consisted of a CNN, LSTM, and a Bi-directional. Along with the baseline models, Rojanavasu et al. also implemented hybrid deep learning models which included a Stacked LSTM, CNN-LSTM, and CNN-Bidirectional-LSTM. From the experimental results, it can be seen that the CNN-Bidirectional-LSTM model achieved the highest accuracy score of 99.15% when compared to other models. However, it is to be noted that that the performance scores between the models (both baseline and hybrid) are quite minimal with the lowest score being 97.67%.

Deep recurrent neural networks (DRNNs) possess the capability to automatically extract features from time-series data, resulting in more accurate recognition in DRNN-based HAR systems compared to traditional machine learning methods. However, the efficiency of training and recognition, particularly in terms of running time, has not been fully considered for resource-constrained smartphones. To address this issue, Li et al. have proposed the PSDRNN and tri-PSDRNN schemes, which utilize explicit feature extraction before DRNN using power spectral density (PSD). The PSD feature, which captures frequency characteristics while preserving successive time characteristics of data from smartphone accelerometers, is extracted from linear and triaxle accelerations and used as input for the DRNN classification model. Results from experiments on a real dataset indicate that the PSDRNN can achieve similar effectiveness as

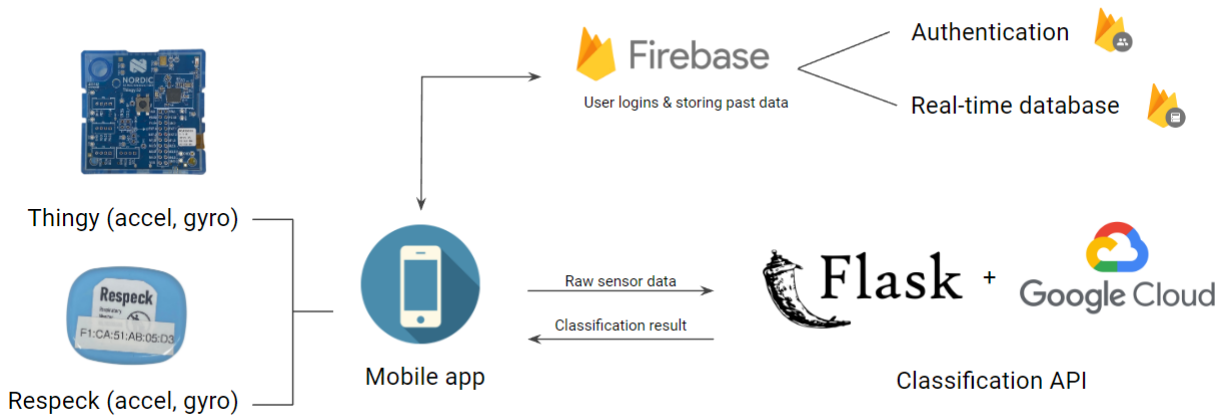
the xyz-DRNN while reducing recognition and training time by 56% and 80%, respectively. Additionally, the tri-PSDRNN outperforms the xyz-DRNN in recognition accuracy and has a lower running time.

# 3 Methodology

## 3.1 Overview of System Implementation

An overview of the system implementation is illustrated in Figure 1. The RESpeck and Thingy sensors are connected and transmit data to an Android phone. The data is processed by the PDIoT mobile application. The app utilizes Firebase to authenticate users and store user data. The data stored in the Firebase Real-time Database includes the classified activities and step count for each date and time. All machine learning models are deployed using Flask and the classification API from Google Cloud Platform, which outputs the classification result after fed data from the sensors. All data and Machine Learning models are stored on the cloud, making the mobile application extremely lightweight. Each aspect of the implementation will be discussed in more detail in the following subsections.

Figure 1: Overview of system implementation




## 3.2 Hardware and Wireless Communication

The device used to collect data is the RESpeck [7] and Thingy device. The RESpeck is a wearable device that includes an accelerometer and gyroscope [8]. It is designed to measure the wearer's breathing, activity, and movement in three dimensions (x, y, and z) [9]. The device is typically worn on the left side of the body below the ribs because this position was found to be the most effective for recording the wearer's movements [7]. The RESpeck sends data packets, which are collected by the sensor at a rate of 25 records per second (Hz), to an Android app via Bluetooth Low Energy technology. Similar to the RESpeck device, the Thingy also samples signals from an accelerometer and gyroscope in the x, y, and z axes [7]. The Thingy [10] is worn in the front right pocket of the trousers. The Thingy also contains a magnetometer device and collects data at 25 Hz [11]. Data from both sensors is transmitted

to the Android app wirelessly via BLE. The RESpeck and Thingy devices are shown in Figure 2.

Figure 2: (a) RESpeck sensor (b) Thingy sensor


(a) (b)

### 3.3 Data Collection and Preprocessing

The data used to train our models is collected from students in the PDIoT course from the years 2021 and 2022. The data is segmented into frames of 2 seconds each. Therefore, each window frame will contain 50 recordings. In HAR, overlapping windows refer to the technique of dividing the time-series data of sensor readings into non-overlapping segments or windows, and then analyzing each window separately [12]. Overlapping windows are a variation of this technique where the windows are not completely non-overlapping, but rather they overlap with each other by a certain percentage [12]. Both overlapping and non-overlapping windows are tested. One frame will only consist of records belonging to one activity. However, since the overlapping windows method did not give any significant improvement towards the classification performance, our group has opted to stick to non-overlapping windows instead. Table 3 shows the number of frames extracted for each activity. From the table, it can be seen that the classes are quite balanced.

### 3.4 Feature Extraction and Processing

Fourteen manually extracted features were extracted from six signals of x-axis acceleration, y-axis acceleration, z-axis acceleration, x-axis angular velocity, y-axis angular velocity, and z-axis angular velocity. The extracted features include mean, median, mode, standard deviation, max, min, range, skew, kurtosis, 10th percentile, 25th percentile, 50th percentile, and 90th percentile. The reason behind these features was that they were shown to work quite well on ensemble machine learning methods [13][14]. A StandardScaler was used to perform data scaling. StandardScaler is a class in the scikit-learn library in Python that is used to standardize a dataset by scaling all of the features to have zero mean and unit variance [15]. This is done by subtracting the mean of each feature from the feature values and dividing the resulting values by the standard deviation of the feature.

Table 3: Distribution of frames in each activity

| Activity type | Num frames (RESpeck) | Num frames (Thingy) |
|---|---|---|
| Ascending stairs | 1897 | 1903 |
| Descending stairs | 1896 | 1887 |
| Desk work | 1923 | 1878 |
| Lying on back | 1890 | 1914 |
| Lying on left side | 1887 | 1880 |
| Lying on right side | 1883 | 1871 |
| Lying on stomach | 1893 | 1897 |
| Movement | 1887 | 1874 |
| Running | 1870 | 1859 |
| Sitting | 1882 | 1879 |
| Sitting bent backward | 1886 | 1879 |
| Sitting bent forward | 1885 | 1872 |
| Standing | 1863 | 1872 |
| Walking | 1874 | 1859 |

### 3.4.1 Dimensionality Reduction Methods

A variety of dimensionality reduction methods were applied to the extracted features of the data. Dimensionality reduction is a technique used in machine learning and data analysis to reduce the number of dimensions or features in a dataset. This can be useful for a variety of purposes, including visualizing data, reducing the storage and computational requirements of algorithms, and improving the performance of machine learning models. The dimensionality reduction methods tested in our experiments include Principal Component Analysis (PCA), Incremental Principal Component Analysis (iPCA), and Sparse Principal Component Analysis (SPCA). Details regarding each method are given below.

#### Principal Component Analysis

Principal component analysis (PCA) is a statistical technique that is used to analyze the variations in a dataset. It is a dimensionality reduction method that can be used to reduce the number of variables in a dataset while retaining as much information as possible. PCA works by finding the directions in which the data varies the most and then projecting the data onto a new set of axes that are defined by these directions. The new axes are called principal components, and the number of principal components is always less than or equal to the number of original features in the dataset.

#### Incremental Principal Component Analysis

Incremental principal component analysis (iPCA) is a variant of principal component analysis (PCA) that is designed to handle large datasets that cannot be fit into memory all at once. It is an iterative algorithm that processes the data one batch at a time, and updates the principal components as new data is processed.

In iPCA, the data is first split into multiple batches, and the principal components are computed for the first batch of data. Then, the second batch of data is processed, and the principal components are updated to account for the new data. This process is repeated until all of the

data has been processed.

One of the main advantages of iPCA is that it allows you to analyze large datasets that would not be possible to fit into memory using standard PCA. It is also useful when the data is too large to fit onto a single machine, as it can be distributed across multiple machines and processed in parallel. However, iPCA can be slower than standard PCA, as it requires multiple passes over the data. It can also be more prone to numerical errors, as the principal components are updated iteratively rather than being computed all at once.

#### Sparse Principal Component Analysis

Sparse principal component analysis (SPCA) is a variant of principal component analysis (PCA) that is designed to handle datasets with a large number of features and a small number of observations. In SPCA, the principal components are obtained by solving a optimization problem that seeks to find a low-dimensional representation of the data that is both interpretable and predictive.

One of the main advantages of SPCA is that it can identify a small number of features that are most relevant for understanding the underlying structure of the data. This can be particularly useful in the context of feature selection, as it allows you to identify a subset of features that are most important for a particular task. In order to achieve sparsity, SPCA solves an optimization problem that seeks to find a low-dimensional representation of the data that is both interpretable and predictive. The optimization problem is typically solved using techniques such as gradient descent or coordinate descent.

One of the main challenges with SPCA is that it can be sensitive to the choice of regularization parameter, which controls the level of sparsity in the solution. It is therefore important to carefully tune this parameter in order to obtain good results.

### 3.5 Human Activity Classification Algorithms

#### 3.5.1 Traditional Machine Learning Methods

A variety of traditional Machine Learning (ML) were tested to see which would obtain the best performance. The classifiers used in our experiments include Decision Tree (DT), Random Forest (RF), Ridge Classifier, Support Vector Machine (SVM), XGBoost (XGB), and LightGBM (LGBM). A description of each classifier is outlined below. The parameters used to train each classifier can be found in Table **??**.

#### Decision Tree

A decision tree classifier is a supervised machine learning algorithm that can be used to predict a class label for a given input sample. It works by constructing a tree-like model of decisions that can be used to predict the class label of a given example. The decision tree classifier works by building a tree of nodes, where each internal node represents a feature or attribute of the input data, and each leaf node represents a class label. The tree is built by recursively partitioning the input data based on the features, such that the data within each partition is more homogeneous with respect to the class labels. The decision tree classifier makes predictions by starting at the root node of the tree and traversing the tree based on the values of the input features until it reaches a leaf node, which represents the final predicted class label.

Decision tree classifiers are widely used due to their simplicity and interpretability. They can

Table 4: ML classifier and respective hyperparameters

| Classifier | Default Hyperparameters |
|---|---|
| DT | ccp_alpha=0.0, class_weight=None, criterion='gini', max_depth=None, max_features=None,max_leaf_nodes=None,min_impurity_decrease=0.0, min_impurity_split=None,min_samples_leaf=1,min_samples_split=2, min_weight_fraction_leaf=0.0, presort='deprecated',random_state=10, random_state=10, splitter='best' |
| SVM | C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',max_iter=-1, probability=False, random_state=10, shrinking=True, tol=0.001, verbose=False |
| RF | bootstrap=True, ccp_alpha=0.0, class_weight=None,criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,min_impurity_split=None, min_samples_split=2,min_weight_fraction_leaf=0.0,n_estimators=100, n_jobs=-1,oob_score=False, random_state=10, verbose=0, max_samples=None,warm_start=False, min_samples_leaf=1 |
| XGBoost | base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1,colsample_bytree=1, gamma=0, learning_rate=0.1,max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,n_estimators=100, n_jobs=1, nthread=None, objective='binary:logistic',random_state=0 reg_alpha=0, reg_lambda=1,scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1 |
| LightGBM | boosting_type='gbdt', class_weight=None, colsample_bytree=1.0, importance_type='split', learning_rate=0.1, max_depth=-1, min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0, n_estimators=100, n_jobs=-1, num_leaves=31, objective=None, random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=1.0, subsample_for_bin=200000, subsample_freq=0 |
| ExtraTree | criterion="gini",splitter="random",max_depth=None, min_samples_split=2,min_samples_leaf=1,min_weight_fraction=1 min_weight_fraction_leaf=0,max_features="sqrt" |
| Ridge | alpha=1.0,fit_intercept=True,cop_X=True,max_iter=True,tol=1e-4 class_weight=None,solver='auto',positive=False,random_state=None |

handle high-dimensional data and can work with both numerical and categorical data. However, they can be prone to overfitting, especially if the tree is allowed to grow too deep. Pruning the tree can help to improve the generalization performance of the model.

### Random Forest

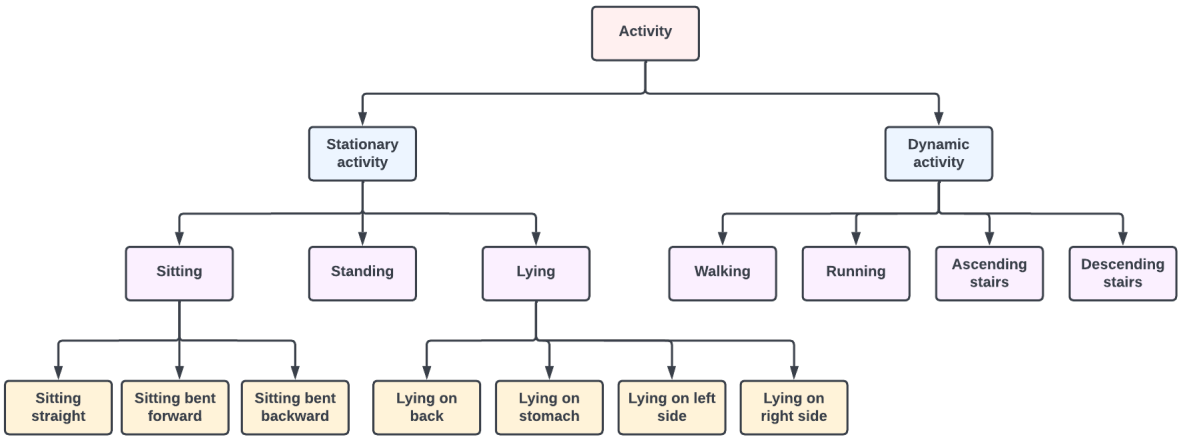
A random forest classifier is an ensemble machine learning model that consists of a number of decision trees trained on different samples of the training data. The idea behind a random forest classifier is that each decision tree will make predictions that are somewhat independent of the predictions made by the other decision trees, and that the combination of the predictions from all of the trees will be more accurate than any of the individual trees. To train a random forest classifier, the algorithm first generates a number of decision trees using a bootstrapped sample of the training data and a random subset of the features. The trees are trained using these subsets of the data and features, and the resulting tree models are stored in the random forest. To make a prediction for a new sample, the random forest classifier feeds the sample to each of the decision trees, and the trees make their predictions. The class label that is predicted by the majority of the trees is then chosen as the final prediction of the random forest.

Random forests are a popular machine learning method due to their ability to handle high-dimensional data, their robustness to overfitting, and their ability to provide feature importance scores. They are often used for both classification and regression tasks.

### Hierarchical Classification with Random Forest

In our experiments we implemented a hierarchical classifier using Random Forest (HCRF). The hierarchy used in the classification process can be seen in Figure 3. A hierarchical classifier is a machine learning classifier that is organized into a hierarchy or tree structure, with the goal of making predictions based on the sequence of decisions made at each level of the hierarchy. At each level of the hierarchy, the classifier makes a decision based on the input data and selects one of the available classes. The decision process continues until the final level of the hierarchy is reached, at which point the classifier makes a final prediction based on the sequence of decisions made at each level. Hierarchical classifiers can be useful in situations where the input data can be organized into a hierarchy of classes, and where the decision process can be represented as a tree structure. They can also be useful for handling multi-class classification problems, where the input data can belong to multiple classes simultaneously.

Figure 3: Diagram of class hierarchy implemented in HCRF



**Extra Tree**

Extra Trees (Extra Randomized Trees) is an ensemble learning method that can be used for both classification and regression tasks. It is a variant of the random forest algorithm, which is a popular ensemble method that combines the predictions of multiple decision trees to make a more accurate and stable prediction. Like random forests, extra trees constructs a set of decision trees at training time and combines their predictions at prediction time. However, extra trees differs from random forests in the way that the decision trees are constructed. In extra trees, each tree in the ensemble is built using a random subset of the features, and the splits at each node are chosen from a random subset of the possible split points. This makes the trees more random, and the overall model less sensitive to the choice of features.

**Ridge Classifier**

A ridge classifier is a type of linear classifier that uses a regularization term to penalize large coefficients for the features. The regularization term is a hyperparameter that controls the complexity of the model, and can help to prevent overfitting. In ridge classification, the objective is to find the coefficients that minimize the sum of the squared residuals, subject to the constraint that the sum of the squares of the coefficients is less than a certain value. This constraint helps to shrink the coefficients towards zero, which can reduce the variance of the model and improve its generalization performance. Ridge classification is often used in situations where the number of features is greater than the number of samples, and can be a good alternative to traditional linear classification methods, such as the logistic regression classifier.

**Support Vector Machine**

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm that can be used for both classification and regression tasks. The goal of an SVM is to find a hyperplane in a high-dimensional space that maximally separates the classes, or to find the hyperplane that maximizes the margin between the classes, if they are not separable. SVM finds the line (or hyperplane) that has the greatest distance to the nearest examples of each class. These nearest examples are known as "support vectors" and the distance between the hyperplane and the support vectors is known as the "margin". The SVM algorithm works by constructing a decision boundary that maximizes the margin between the positive and negative examples. It does this by solving a quadratic optimization problem that seeks to find the hyperplane that has the largest margin. Once the hyperplane has been found, it can be used to make predictions for new examples by assigning them to the class on the correct side of the hyperplane.

SVMs are particularly useful when the data is not linearly separable, as they can use kernel functions to transform the data into a higher-dimensional space where it may be possible to find a linear separation. SVMs are also relatively robust to overfitting and can work well with high-dimensional data. However, they can be computationally expensive to train, especially for large datasets.

**XGBoost**

XGBoost is an implementation of the gradient boosting algorithm, which is a popular ensemble machine learning method used for both classification and regression tasks. It is known for its efficiency and high performance, and has been used to win many machine learning competitions.

Like other gradient boosting algorithms, XGBoost works by training a series of weak learners, where each learner is trained to improve the performance of the model on the training data. The weak learners in XGBoost are decision trees, and the model is trained by iteratively adding trees to the ensemble, with each tree learning to correct the mistakes made by the previous trees. The model is trained using gradient descent to minimize the loss function, and the resulting tree ensemble is used to make predictions on new data. One of the key features of XGBoost is that it includes a number of techniques to improve the efficiency and performance of the model, such as regularization, sparsity-aware split finding, and block structure to reduce the communication cost. These techniques help to reduce overfitting and improve the generalization performance of the model.

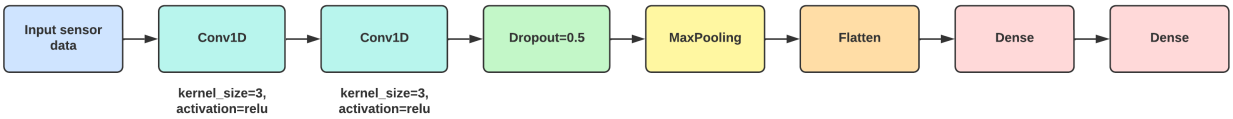
**LightGBM**

LightGBM is an implementation of the gradient boosting algorithm that is designed to be efficient and fast, especially for large-scale data. It is a popular choice for many machine learning tasks, including classification and regression. Like other gradient boosting algorithms, LightGBM works by training a series of weak learners, where each learner is trained to improve the performance of the model on the training data. The weak learners in LightGBM are decision trees, and the model is trained by iteratively adding trees to the ensemble, with each tree learning to correct the mistakes made by the previous trees. Like XGBoost, the model is trained using gradient descent to minimize the loss function, and the resulting tree ensemble is used to make predictions on new data. One of the key features of LightGBM is its use of histogram-based algorithms to approximate the split points in decision trees, which can significantly reduce the training time compared to traditional gradient boosting algorithms. LightGBM also includes techniques to handle large-scale data, such as support for parallel and distributed training, and the use of efficient data structures to store the model.

### 3.5.2 Deep Learning Algorithms

In addition to traditional ML methods, our group also tested several deep learning methods known to work well for HAR tasks including 1D Convolutional Neural Network (1D-CNN), Long Short Term Memory (LSTM), CNN-LSTM, and Convolutional LSTM. As deep learning automatically performs feature extractions, our manually extracted features mentioned in Section [REFERENCE-NUMBER] were not used for these methods. A description of each method is outlined in the following subsections. Model architectures of the 1D-CNN, LSTM, CNN-LSTM, and ConvLSTM can be seen in Figures **??**, **??**, **??**, and **??**, respectively.

Figure 4: Model architecture of 1D CNN




**1D Convolutional Neural Network**

A 1D convolutional neural network (CNN) is a type of CNN that processes one-dimensional input data, such as time series data, text data, or audio data. Like other CNNs, a 1D CNN is composed of layers of interconnected nodes, where each node processes the input data and

passes it on to the next layer. The layers of a 1D CNN include convolutional layers, activation layers, and pooling layers. The convolutional layers of a 1D CNN apply a convolution operation to the input data, which involves sliding a kernel or filter across the input data and computing the dot product between the kernel and the input at each position. This operation helps the CNN learn local patterns or features in the data. The activation layers of a 1D CNN apply an element-wise non-linear activation function to the output of the convolutional layers, which helps the CNN learn more complex patterns in the data. The pooling layers of a 1D CNN perform down-sampling of the input data, which helps to reduce the size of the input and to reduce the computational burden of the model.

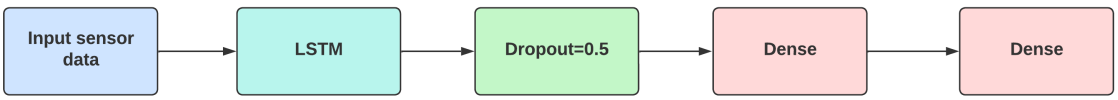
### Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) is a type of recurrent neural network (RNN) that is well-suited for modeling long-term dependencies in time series data. LSTMs are capable of learning and remembering over long periods of time, and is often used for tasks that require the model to remember context from previous inputs. LSTMs work by using special units called memory cells, which are designed to remember information for long periods of time. These memory cells are controlled by three different types of gates: an input gate, an output gate, and a forget gate. The input gate determines which information is allowed to enter the memory cell, the output gate determines which information is allowed to pass out of the memory cell, and the forget gate determines which information is removed from the memory cell.

In an LSTM, the input data is processed by the input gate and is then stored in the memory cell. The output gate then determines which information is allowed to pass out of the memory cell and be used by the rest of the network. The forget gate determines which information is removed from the memory cell, allowing the LSTM to forget irrelevant or outdated information and focus on more relevant information.

One of the main advantages of LSTMs is that they can learn to remember and forget information over long periods of time, allowing them to handle long-term dependencies in the data. This makes them well-suited for tasks such as language translation, where the meaning of a word can depend on the context of the words that come before and after it.

Figure 5: Model architecture of LSTM Model

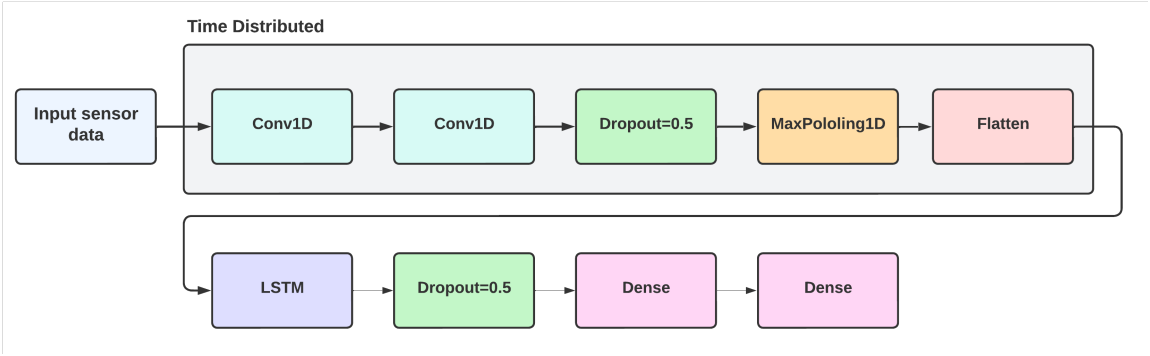



### CNN-LSTM Model

A CNN-LSTM is a type of neural network that combines the strengths of both CNNs and LSTMs. In a CNN-LSTM, the input data is first processed by a CNN, which extracts features from the data and reduces its dimensionality. The output of the CNN is then fed into an LSTM, which processes the data sequentially and learns to model long-term dependencies in the data. One of the main advantages of using a CNN-LSTM is that it allows you to take advantage of the strengths of both CNNs and LSTMs. The CNN is able to extract useful features from the data, while the LSTM is able to learn long-term dependencies in the data. This makes CNN-LSTMs well-suited for tasks that involve both spatial and temporal processing.

The model architecture of the implemented CNN-LSTM model can be seen in Figure **??**. The CNN LSTM model reads parts of the primary sequence as blocks, extracts features from each block, and then uses the LSTM to interpret those features. The entire CNN can be placed within a TimeDistributed layer, allowing it to read all four subsequences in the window. A TimeDistributed layer in deep learning is a layer that applies the same weights to every timestep of a 3D input tensor. This allows the model to process sequences of varying lengths, as the same operation is applied to each timestep independently. It can be used to apply a dense layer to each timestep of a sequence, for example, in order to classify each timestep individually. The extracted features are then condensed and passed to the LSTM, which extracts its own features before making a final mapping to an activity. A common structure for this model includes two CNN layers followed by dropout and a max pooling layer, which is the structure used in the CNN LSTM model described.

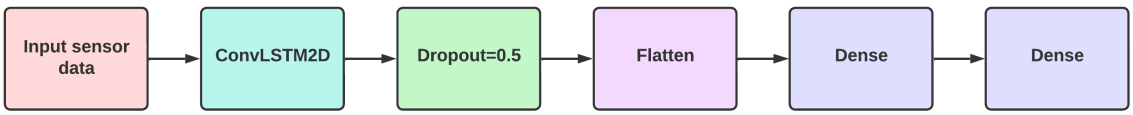
Figure 6: Model architecture of CNN-LSTM Model




### Convolutional LSTM Network

An extension of the CNN-LSTM concept is to incorporate the convolutions of the CNN within the LSTM itself. The Convolutional LSTM (ConvLSTM) model differs from an LSTM and a CNN-LSTM in that it directly applies convolutions to the input data before it is processed by the LSTM units, rather than reading the data in directly or interpreting the output of a CNN model. The idea of extending the CNN LSTM is to integrate the convolution process of the CNN into the LSTM, creating a Convolutional LSTM or ConvLSTM. This is used for spatio-temporal data, and differs from the LSTM and CNN LSTM by using convolutions as a part of reading input directly into the LSTM units. The ConvLSTM2D class needs configuration for both the CNN and LSTM, including the number of filters, kernel size, and activation function. Like other CNN or LSTM models, the output must be converted into a single vector before it can be analyzed by a dense layer. The model architecture of the implemented ConvLSTM model can be seen in Figure **??**.

Figure 7: Model architecture of ConvLSTM Model



## 3.6 Step Counting Algorithm

After applying the HAR classifier, we implemented a common step counting algorithm to detect the number of steps in each window. The step counting algorithm is dependent of the classification result of the HAR classifier as it will only be applied for activities such as walking, running, ascending stairs, and descending stairs. In our algorithm, Fast Fourier Transform (FFT) and thresholding was used to count the number of steps in each segmented window of 2 seconds based on x,y, and z-axis of the acceleration data from the Thingy instead as it is found to be more accurate when classifying activities which involve taking steps (walking, running, etc.) in live-classification, mainly due to the sensor placement. Steps can be detected by looking for peaks in the acceleration data, which correspond to the impact of the foot hitting the ground while walking or running. To detect peaks, FFT is applied to the acceleration data to transform it to the frequency domain. The FFT coefficients can then be thresholded to separate the high-frequency components that correspond to the impact peaks from the low-frequency components that correspond to the background noise and the overall trend of the data. The number of peaks detected in this way can be used to estimate the number of steps taken.

We also applied a low-pass filter to the data in order to remove high-frequency noise from the acceleration measurements that are to be used for step counting. This helps improve the accuracy of the step count by reducing the impact of noise on the measurements. We utilized a moving average filter, which works by taking the average of the data over a certain window of time, which can be used to smooth out the data and remove high-frequency noise.

## 3.7 Mobile Application

### 3.7.1 Database and Model Deployment

On the software engineering aspect, we used Firebase real-time database to store user and sensor data. All machine learning models and algorithms used to process live sensor data are deployed using Flask and Google Cloud Platform with their classification API.

Firebase Real-time Database is a cloud-hosted database service provided by Google as part of the Firebase platform. It allows developers to store and sync data across multiple devices in real-time, enabling users to access the same data on different devices and platforms. The Firebase Real-time Database is a NoSQL database that stores data as JSON documents. It provides a simple API for reading and writing data, and it uses WebSockets to establish a connection between the client and the server and synchronize data in real-time. One of the main advantages of the Firebase Real-time Database is that it allows developers to build real-time, collaborative applications without having to manage the complexity of setting up and maintaining their own real-time synchronization infrastructure. It is also designed to be easy to use and integrate with other Firebase services and Google Cloud Platform products.

Google Cloud Platform is a suite of cloud computing services offered by Google. It allows businesses, developers, and organizations to build, test, and deploy applications and websites on Google's highly scalable and reliable infrastructure. Google Cloud Platform was used to deploy our machine learning models with its services including Cloud AutoML, Cloud TensorFlow and App Engine. Google Cloud Platform is designed to be scalable, secure, and reliable, and it is used by businesses, developers, and organizations around the world to build and run a wide range of applications and services.

In order to send sensor data from the mobile application to Flask and Google Cloud Platform, we

used the Volley library. Volley is an HTTP library for Android that makes it easier to send and receive data from the web. It was developed by Google and is now a part of the Android Open Source Project (AOSP). One of the main benefits of Volley is that it automatically manages the process of sending and receiving network requests, so you don't have to worry about low-level details like creating connections and parsing responses. Volley also includes features like request prioritization, cancelation, and caching, which can make it easier to build a responsive and efficient Android app. With Volley, listeners can be used to receive callbacks when requests are completed. Built-in cache can be used to store data locally and reduce the number of network request the app needs to make; therefore, reducing the latency of data transmission back and forth between Google Cloud Platform.

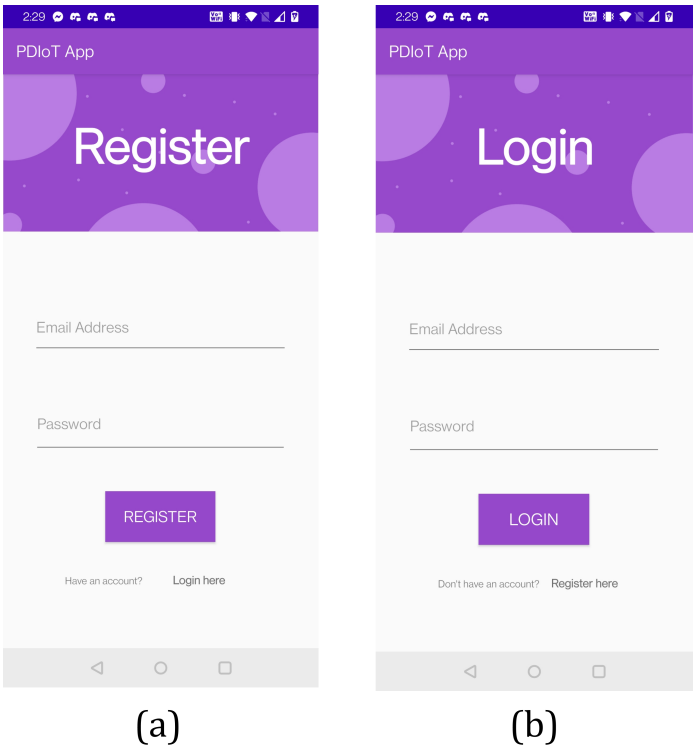
### 3.7.2 Features and User Interface

Our mobile application has the following features.

1. Users are able to log in or register an account.
2. Users can see the live output of sensor values (from both the Thingy and the RESpeck) in the form of graphs.
3. Users are able to see their classified activity in real-time.
4. Users can see the number of steps they have walked in real-time.
5. Users can see the amount of time spent performing each activity in real time.
6. Users can view historical data of their activity and steps.

The user interface of the application can be seen in Figures 8 - 11.

Figure 8: Authentication user interface




(a) (b)

Figure 9: (a) Register (b) Login

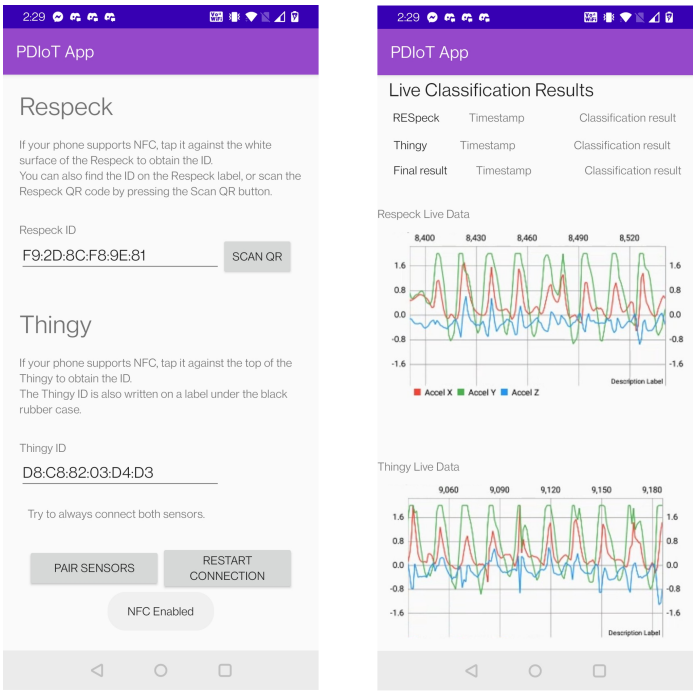
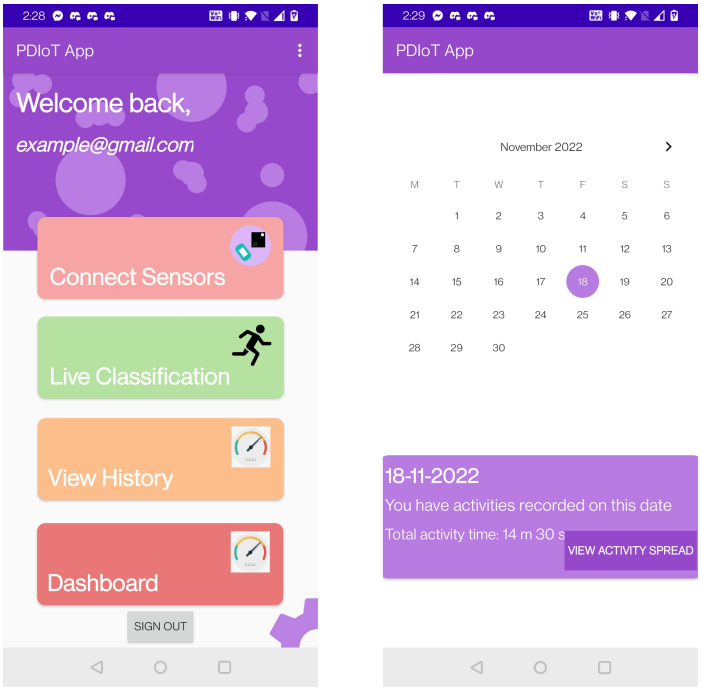



Figure 10: (a) Main menu user interface (b) History calendar




# 4 Results

## 4.1 HAR Offline Classifier Performance

A variety of combinations of ML and DL classifiers and dimensionality reduction methods were tested to see which combination would achieve the best performance. The methods were evalu-

Figure 11: (a) Total steps and activity spread UI (b) Daily dashboard


ated using K-Fold cross-validation with K=10. K-fold cross-validation is a technique to estimate the performance of the model by training it multiple times and averaging the performance metric on different subset of data points. This can prevent overfitting and underfitting and ensure the model generalizes well on unseen data. The classifier with the best performance is selected to be deployed inside the mobile application. To be thorough with our investigation, we also tested the methods without dimensionality reduction methods. A table depicting the accuracy, F1, precision, and recall scores in percentages of each classifier is shown in Table **??**.

From the table, we can see that the methods with the highest performance (in terms of accuracy) are SPCA→RF for the Thingy and SPCA→LightGBM for the Thingy with accuracies of 0.9591 and 0.9513, respectively. We implemented a 4-class and a 14-class version of the models where the 4 classes consisted of basic activities (lying, running, sitting/standing, and walking). As a result, these models were selected to be deployed for the mobile application. The performance results for each class in the models (both 4-class and 14-class) are shown in Figure **??** to Figure **??**, respectively. The confusion matrices for the 4-class and 14-class models are shown in Figure **??** to Figure **??**, respectively.

From our experimental results, we have found that SPCA generally improves the classification performance of most ML classifiers and deep learning methods, while PCA and iPCA degrades it. Additionally, ensemble machine learning methods such as RF, LightGBM, and XGBoost will often achieve better classification accuracies as compared to simple machine learning methods like KNN and DT. Most deep learning methods are also shown to achieve comparable results LightGBM and RF, which are the highest achieving methods. However, due to efficiency and time consumption reasons, we opted to deploy RF and LightGBM as they are more lightweight and take less time to classify activities.

### 4.2 Live Classification Results

To test the live HAR classification in real-time, three participants wore the RESpeck and Thingy sensors and performed several activities to test the accuracy of the actual outputted classification on the mobile app. Both classification models for the Thingy and RESpeck sensor were deployed. Since the Thingy sensor is better at classifying walking, running, ascending and descending stairs activities, the final classification result is conditioned to trust the classification result from the Thingy model for those activities. On the other hand, the RESpeck model is better at classifying activities such as sitting straight, sitting bent forward, sitting bent backward, lying on back, lying on stomach, lying on left side, and lying on right side; therefore, the mobile app is conditioned to follow the result from the RESpeck model for those activities. The results of the live classification can be seen in Table **??** and Table **??** for the 8-class model and 14-class model, respectively.

#### 4.2.1 Mobile Application Performance

Since everything, including the machine learning model and data are stored entirely on the cloud, the mobile application takes very little storage on the phone and is extremely lightweight. The size of the application is only 10.2 Mb. The CPU usage of the application averages at around 90 Mb and reaches a max of 214 Mb. The computer power use in 5 minutes is around 14 mAh. From these benchmarking tests, we can say the app is relatively lightweight and efficient due to various design choices described in earlier sections.

## 5 Conclusions

In this project, we have implemented an end-to-end system human activity recognition system accessible via a mobile phone application. The system deploys two machine learning models: SPCA→LightGBM and SPCA →RF. Before deploying the models on the application to perform live real-time classification, we first experimented with a variety of combinations of dimensionality reduction methods (PCA, iPCA, SPCA), machine learning classifiers (KNN, DT, SVM, ExtraTree, XGBoost, LightGBM, Ridge, and HCRF), and deep learning networks (1D-CNN, LSTM, CNN-LSTM, and ConvLSTM). In terms of model deployment and mobile application implementation, all models and databases are deployed online on the cloud; therefore, making the application extremely lightweight and not computationally expensive. With the mobile app, users are able to authenticate themselves, see their activity classification in real-time, see their step count in real-time, and keep track of their activity history.

### 5.1 Potential Improvements

Some future improvements would consist of allowing users to calibrate the mobile application so that analyses produced are more customized. Additionally, almost all aspects of the mobile application are on the cloud; this makes the system highly reliable in working and preferably fast internet connection as to reduce latency and errors in data transmission. This means that the app may not collect activity data of the user in areas where there is not internet connection.

Table 5: Comparison of classification performance between classifiers - RESpeck sensor

| Dimensionality Reduction | Classifier | Accuracy (%) | F1 (%) | Precision (%) | Recall (%) |
|---|---|---|---|---|---|
| None | KNN | 0.7962 | 0.7939 | 0.8005 | 0.7959 |
| | DT | 0.7737 | 0.7732 | 0.7742 | 0.7735 |
| | RF | 0.9492 | 0.9490 | 0.9497 | 0.9491 |
| | HCRF | 0.9370 | 0.9367 | 0.9380 | 0.9369 |
| | ExtraTree | 0.8060 | 0.8044 | 0.8117 | 0.8058 |
| | Ridge | 0.7787 | 0.7697 | 0.7785 | 0.7786 |
| | SVM | 0.8068 | 0.8018 | 0.8079 | 0.8068 |
| | XGBoost | 0.8819 | 0.8808 | 0.8815 | 0.8818 |
| | LightGBM | 0.9488 | 0.9486 | 0.9492 | 0.9488 |
| | 1D-CNN | 0.9319 | 0.9316 | 0.9323 | 0.9318 |
| | LSTM | 0.9383 | 0.9380 | 0.9385 | 0.9382 |
| | CNN-LSTM | 0.9242 | 0.9242 | 0.9244 | 0.9244 |
| | ConvLSTM | 0.9304 | 0.9304 | 0.9309 | 0.9306 |
| PCA | RF | 0.8148 | 0.8112 | 0.8114 | 0.8147 |
| | HCRF | 0.8053 | 0.8038 | 0.8112 | 0.8051 |
| | XGBoost | 0.8243 | 0.8203 | 0.8233 | 0.8242 |
| | LightGBM | 0.8792 | 0.8784 | 0.8810 | 0.8790 |
| | 1D-CNN | 0.8593 | 0.8591 | 0.8600 | 0.8591 |
| | LSTM | 0.8068 | 0.8017 | 0.8079 | 0.8067 |
| | CNN-LSTM | 0.8133 | 0.8095 | 0.8113 | 0.8131 |
| | ConvLSTM | 0.8491 | 0.8475 | 0.8543 | 0.8490 |
| iPCA | RF | 0.8176 | 0.8141 | 0.8145 | 0.8175 |
| | HCRF | 0.8111 | 0.8073 | 0.8087 | 0.8110 |
| | XGBoost | 0.8245 | 0.8199 | 0.8228 | 0.8245 |
| | LightGBM | 0.8859 | 0.8853 | 0.8862 | 0.8858 |
| | 1D-CNN | 0.8735 | 0.8730 | 0.8741 | 0.8731 |
| | LSTM | 0.8414 | 0.8409 | 0.8427 | 0.8412 |
| | CNN-LSTM | 0.7734 | 0.7729 | 0.7741 | 0.7732 |
| | ConvLSTM | 0.7737 | 0.7731 | 0.7741 | 0.7735 |
| SPCA | RF | 0.9500 | 0.9498 | 0.9502 | 0.9500 |
| | HCRF | 0.9495 | 0.9492 | 0.9496 | 0.9494 |
| | XGBoost | 0.8861 | 0.8850 | 0.8858 | 0.8860 |
| | LightGBM | **0.9513** | **0.9512** | **0.9515** | **0.9512** |
| | 1D-CNN | 0.8859 | 0.8853 | 0.8862 | 0.8858 |
| | LSTM | 0.9489 | 0.9481 | 0.9495 | 0.9486 |
| | CNN-LSTM | 0.9487 | 0.9483 | 0.9488 | 0.9485 |
| | ConvLSTM | 0.9262 | 0.9261 | 0.9265 | 0.9263 |

Table 6: Comparison of classification performance between classifiers - Thingy sensor

| Dimensionality Reduction | Classifier | Accuracy (%) | F1 (%) | Precision (%) | Recall (%) |
|---|---|---|---|---|---|
| None | DT | 0.7140 | 0.7142 | 0.7160 | 0.7137 |
| | RF | 0.9487 | 0.9483 | 0.9488 | 0.9485 |
| | HCRF | 0.8414 | 0.8409 | 0.8427 | 0.8413 |
| | ExtraTree | 0.7034 | 0.6953 | 0.7023 | 0.7034 |
| | Ridge | 0.6957 | 0.6845 | 0.6862 | 0.6959 |
| | SVM | 0.7033 | 0.6950 | 0.7021 | 0.7032 |
| | XGBoost | 0.7797 | 0.7758 | 0.7796 | 0.7796 |
| | LightGBM | 0.9489 | 0.9481 | 0.9495 | 0.9486 |
| | 1D-CNN | 0.9495 | 0.9492 | 0.9496 | 0.9494 |
| | LSTM | 0.9383 | 0.9380 | 0.9385 | 0.9382 |
| | CNN-LSTM | 0.9500 | 0.9498 | 0.9502 | 0.9500 |
| | ConvLSTM | 0.9389 | 0.9387 | 0.9392 | 0.9389 |
| PCA | DT | 0.7034 | 0.6953 | 0.7023 | 0.7034 |
| | RF | 0.7225 | 0.7147 | 0.7146 | 0.7225 |
| | XGBoost | 0.7096 | 0.7011 | 0.7033 | 0.7096 |
| | LightGBM | 0.8061 | 0.8049 | 0.8093 | 0.8061 |
| | 1D-CNN | 0.7672 | 0.7136 | 0.6764 | 0.6807 |
| | LSTM | 0.7734 | 0.7729 | 0.7741 | 0.7732 |
| | CNN-LSTM | 0.7737 | 0.7732 | 0.7742 | 0.7735 |
| | ConvLSTM | 0.8068 | 0.8017 | 0.8079 | 0.8067 |
| iPCA | DT | 0.7098 | 0.7105 | 0.7129 | 0.7096 |
| | RF | 0.7221 | 0.7148 | 0.7143 | 0.7221 |
| | XGBoost | 0.7136 | 0.7058 | 0.7067 | 0.7136 |
| | LightGBM | 0.8123 | 0.8110 | 0.8129 | 0.8122 |
| | 1D-CNN | 0.7033 | 0.6950 | 0.7021 | 0.7032 |
| | LSTM | 0.8777 | 0.8331 | 0.8130 | 0.8167 |
| | CNN-LSTM | 0.7672 | 0.7136 | 0.6764 | 0.6807 |
| | ConvLSTM | 0.8768 | 0.8359 | 0.8163 | 0.8210 |
| SPCA | DT | 0.8282 | 0.8276 | 0.8285 | 0.8279 |
| | LightGBM | 0.9582 | 0.9580 | 0.9586 | 0.9580 |
| | XGBoost | 0.8311 | 0.8295 | 0.8322 | 0.8311 |
| | RF | **0.9591** | **0.9589** | **0.9593** | **0.9590** |
| | 1D-CNN | 0.8278 | 0.8260 | 0.8290 | 0.8277 |
| | LSTM | 0.9591 | 0.9589 | 0.9593 | 0.9590 |
| | CNN-LSTM | 0.8301 | 0.8283 | 0.8313 | 0.8300 |
| | ConvLSTM | 0.9492 | 0.9490 | 0.9498 | 0.9491 |

Table 7: 4-Class Model Accuracies (RESpeck sensor)

| Class | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| Lying | 0.9764 | 0.9938 | 0.9764 | 0.9851 |
| Running | 0.9920 | 1.0000 | 0.9920 | 0.9960 |
| Sitting/Standing | 0.9953 | 0.9819 | 0.9953 | 0.9885 |
| Walking | 1.0000 | 0.9918 | 1.0000 | 0.9959 |
| **Avg.** | **0.9909** | **0.9919** | **0.9909** | **0.9914** |

Table 8: 4-Class Model Accuracies (Thingy sensor)

| Class | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| Lying | 0.9820 | 0.9967 | 0.9820 | 0.9893 |
| Running | 1.0000 | 0.9662 | 1.0000 | 0.9828 |
| Sitting/Standing | 0.9869 | 0.9922 | 0.9869 | 0.9895 |
| Walking | 0.9942 | 0.9438 | 0.9942 | 0.9683 |
| **Avg.** | **0.9870** | **0.9747** | **0.9908** | **0.9825** |

Table 9: 14-Class Model Accuracies (RESpeck sensor)

| Class | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| Ascending stairs | 0.9020 | 0.9046 | 0.9020 | 0.9033 |
| Descending stairs | 0.9231 | 0.9253 | 0.9230 | 0.9242 |
| Desk work | 0.9325 | 0.8829 | 0.9325 | 0.9071 |
| Lying on back | 0.9848 | 0.9773 | 0.9847 | 0.9810 |
| Lying on left | 0.9864 | 0.9838 | 0.9864 | 0.9851 |
| Lying on right | 1.0000 | 1.0000 | 0.9987 | 0.9987 |
| Lying on stomach | 0.9974 | 0.9974 | 0.9974 | 0.9974 |
| Movement | 0.7851 | 0.8404 | 0.7851 | 0.8118 |
| Running | 0.9839 | 0.9760 | 0.9839 | 0.9799 |
| Sitting | 0.9549 | 0.9741 | 0.9549 | 0.9644 |
| Sitting bent backward | 0.9644 | 0.9832 | 0.9645 | 0.9738 |
| Sitting bent forward | 0.9484 | 0.9813 | 0.9485 | 0.9646 |
| Standing | 0.9849 | 0.9703 | 0.9849 | 0.9776 |
| Walking | 0.9225 | 0.8827 | 0.9022 | 0.9022 |

Table 10: 14-Class Model Accuracies (Thingy sensor)

| Class | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| Ascending stairs | 0.8842 | 0.9327 | 0.8842 | 0.9078 |
| Descending stairs | 0.9017 | 0.9191 | 0.9017 | 0.9103 |
| Desk work | 0.9644 | 0.9338 | 0.9644 | 0.9489 |
| Lying on back | 0.9604 | 0.9927 | 0.9604 | 0.9763 |
| Lying on left | 0.9631 | 0.9784 | 0.9631 | 0.9707 |
| Lying on right | 0.9784 | 0.9912 | 0.9784 | 0.9848 |
| Lying on stomach | 0.9455 | 0.9970 | 0.9455 | 0.9706 |
| Movement | 0.8943 | 0.6857 | 0.8943 | 0.7762 |
| Running | 0.9868 | 0.9600 | 0.9868 | 0.9732 |
| Sitting | 0.9342 | 0.9969 | 0.9342 | 0.9645 |
| Sitting bent backward | 0.9526 | 0.9940 | 0.9526 | 0.9729 |
| Sitting bent forward | 0.9427 | 0.9909 | 0.9427 | 0.9662 |
| Standing | 0.9558 | 0.9985 | 0.9558 | 0.9767 |
| Walking | 0.9232 | 0.9177 | 0.9232 | 0.9204 |

Table 11: Live Classification Performance (8-class model)

| Activity | RP1 | RP2 | RP3 | TP1 | TP2 | TP3 | Final res |
|---|---|---|---|---|---|---|---|
| Sitting | Sitting | Sitting | Standing | Sitting | Lying | Sitting | RESpeck |
| Standing | Standing | Standing | Standing | Standing | Standing | Standing | Thingy |
| Lying | Lying | Lying | Lying | Lying | Lying | Lying | RESpeck |
| Walking | Running | Ascend | Walking | Ascend | Walking | Walking | Thingy |
| Running | Running | Running | Running | Running | Running | Running | RESpeck |
| Ascending | Descend | Ascend | Walking | Walking | Ascend | Ascend | Thingy |
| Descending | Descend | Walking | Walking | Descend | Walking | Descend | Thingy |
| Movement | Movement | Movement | Movement | Movement | Movement | Movement | RESpeck |

Table 12: Live Classification Performance (14-class model)

| Activity | RP1 | RP2 | RP3 | TP1 | TP2 | TP3 | Final RES |
|---|---|---|---|---|---|---|---|
| Sitting straight (SS) | SS | DW | DW | DW | SS | SS | RESpeck |
| Sitting forward (SF) | SF | SF | SF | DW | SS | SS | RESpeck |
| Sitting backward (SB) | SS | SB | SB | DW | SF | SW | RESpeck |
| Desk work (DW) | DW | SS | SS | ST | SS | DW | RESpeck |
| Standing (ST) | ST | ST | ST | ST | ST | ST | RESpeck |
| Lying back (LB) | LB | LB | LB | LB | LB | LB | RESpeck |
| Lying stomach (LS) | LS | LS | LS | LS | LS | LS | RESpeck |
| Lying left (LL) | LL | LL | LL | LL | LL | LL | RESpeck |
| Lying right (LR) | LR | LR | LR | LR | LR | LR | RESpeck |
| Walking (W) | W | W | AS | AS | W | W | Thingy |
| Running (R) | R | R | R | R | R | R | Thingy |
| Ascending stairs (AS) | AS | DS | W | W | AS | AS | Thingy |
| Descending stairs (DS) | DS | DS | W | DS | W | DS | Thingy |
| Movement (M) | M | M | M | M | M | M | RESpeck |

# References


[1] Zhaosheng Shao, Jianxin Guo, Yushuai Zhang, Rui Zhu, and Liping Wang. Lightbgm for human activity recognition using wearable sensors. In *2021 International Conference on Intelligent Transportation, Big Data  Smart City (ICITBS)*, pages 668–671, March 2021.

[2] V. Radhika, Ch.Rajendra Prasad, and A. Chakradhar. Smartphone-based human activities recognition system using random forest algorithm. In *2022 International Conference for Advancement in Technology (ICONAT)*, pages 1–4, Jan 2022.

[3] Negar Golestani and Mahta Moghaddam. A comparison of machine learning classifiers for human activity recognition using magnetic induction-based motion signals. In *2020 14th European Conference on Antennas and Propagation (EuCAP)*, pages 1–3, March 2020.

[4] Chengli Hou. A study on imu-based human activity recognition using deep learning and traditional machine learning. In *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, pages 225–234, May 2020.

[5] Ku Nurhanim, I. Elamvazuthi, and L.I. Izhar. Ensemble methods for classifying of human activity recognition. In *2018 IEEE 4th International Symposium in Robotics and Manufacturing Automation (ROMA)*, pages 1–5, Dec 2018.

[6] Chanvichet Vong, Thitiphoom Theptit, Virinya Watcharakonpipat, Passara Chanchotisatien, and Seksan Laitrakun. Comparison of feature selection and classification for human activity and fall recognition using smartphone sensors. In *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*, pages 170–173, March 2021.

[7] D. K. Arvind, C. A. Bates, D. J. Fischer, and J. Mann. A sensor data collection environment for clinical trials investigating health effects of airborne pollution. In *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 88–91, March 2018.

[8] D. K. Arvind, D. J. Fischer, C. A. Bates, and S. Kinra. Characterisation of Breathing and Physical Activity Patterns in the General Population Using the Wearable Respeck Monitor. In Lorenzo Mucchi, Matti Hämäläinen, Sara Jayousi, and Simone Morosi, editors, *Body Area Networks: Smart IoT and Big Data for Intelligent Health Management*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 68–78, Cham, 2019. Springer International Publishing.

[9] Gordon B. Drummond, Darius Fischer, Margaret Lees, Andrew Bates, Janek Mann, and D. K. Arvind. Classifying signals from a wearable accelerometer device to measure respiratory rate. *ERJ Open Research*, 7(2), April 2021.

[10] Soyoung Jeong, Taewoo Kim, and Rasit Eskicioglu. Human Activity Recognition Using Motion Sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, pages 392–393, New York, NY, USA, November 2018. Association for Computing Machinery.

[11] Florenc Demrozi, Fabio Chiarani, and Graziano Pravadelli. A low-cost BLE-based distance estimation, occupancy detection and counting system. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1430–1433, February 2021. ISSN: 1558-1101.

[12] Akbar Dehghani, Omid Sarbishei, Tristan Glatard, and Emad Shihab. A Quantitative Comparison of Overlapping and Non-Overlapping Sliding Windows for Human Activity Recognition Using Inertial Sensors. *Sensors*, 19(22):5026, January 2019.

[13] Chanvichet Vong, Thitiphoom Theptit, Virinya Watcharakonpipat, Passara Chanchotisatien, and Seksan Laitrakun. Comparison of Feature Selection and Classification for Human Activity and Fall Recognition using Smartphone Sensors. In *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*, pages 170–173, March 2021.

[14] Passara Chanchotisatien and Chanvichet Vong. Implementation of Machine Learning and Deep Learning Algorithms with Dimensionality Reduction Methods for Internet of Things Gait Analysis and Monitoring Systems. *Sensors and Materials*, 33(12):4245, December 2021.

[15] Ekaba Bisong. Introduction to Scikit-learn. In Ekaba Bisong, editor, *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pages 215–229. Apress, Berkeley, CA, 2019.