Astronomy & Astrophysics manuscript no. output
May 16, 2023

Machine Learning Techniques Applied to Smartphone and Smartwatch Data

David R. Aguilera Dena

e-mail: aguileradena@gmail.com

May 16, 2023

ABSTRACT

Using accelerometer and gyroscope readings from smartphones and smartwatches, we test the ability of five different machine learning algorithms to predict the nature of the physical activity being performed by its user. We use a dataset consisting of readings from 51 participants, engaged in 18 different activities in lapses of approximately 3 minutes per activity.
We find that the machine learning classifiers are effective at classifying physical activities based on whether they are hand-oriented or general physical activities. We find that smartphones are effective at classifying general physical activities, but smartwatches show a better performance with hand based activities and eating activities. We also test the data reduction process by measuring the prediction accuracy of the random forest model, as a function of the number of time steps integrated during the data processing.
Ours finding suggest that a random forest algorithm might be the most suitable for this application, and that a prediction can be made with optimal accuracy if data is integrated for about 200 – 250 times teps, corresponding to 10 – 12.5 s. However, a detailed hyperparameter analysis is required to have a better assessment.

Key words. Activity recognition – Machine learning

1. Introduction

In this report, we analyse a dataset of smartphone and smartwatch accelerometers and gyroscpoes made available by Weiss (2019). The purpose of the analysis is to use machine learning (ML) algorithms as tools to recognise different types of physical activities performed by smartphone and smartwatch users, based on the readings from the devices.

We reduce the data independently and use it to train 5 different ML algorithms. We then measure the accuracy of the predictions obtained from each of the models, as well as the dependency of the accuracy in ML models to the time-averaging process carried out during the processing of the raw data.

We present our data science methods in Section 2, including a description of the data and the data reduction process, as well as the algorithms employed to build ML classifiers based on the data. In Section 3 we present our results.

2. Data science methods

In Section 2.1 we describe the acquisition process and general properties of the dataset. In Section 2.2 we describe the techniques employed to process the data, and compare them to techniques used by other authors in the same dataset. Finally, in Section 2.3 we describe the ML techniques employed for processing the dataset.

2.1. Structure of the raw data

The dataset employed in this report consists of accelerometer and gyroscope readings from smartphones and smartwatches, obtained while asking a sample of 51 users to perform a series of 18 physical activities. The finer details of the dataset are described by Weiss (2019), but here we summarise it and give some relevant features.

Table 1. Strucure of the raw data files

Activity code	Activity
A	walking
B	jogging
C	walking up/downstairs
D	sitting
E	standing
F	typing
G	brushing teeth
H	eating soup
I	eating chips
J	eating pasta
K	drinking from a cup
L	eating a sandwich
M	kicking
O	playing catch
P	dribbling a basketball
Q	writing
R	clapping
S	folding clothes

The participants were instructed to perform each activity for 3 minutes each. The activities included in the study are common activities that involve different kinds of motion. They are summarised in Table 1, along with the label that each activity is given in the data files.

The smartphones employed for the data collection are either the Google Nexus 5/5X or Samsung Galaxy S5, both using Android 6.0. The smartwatch was the LG G Watch running Android

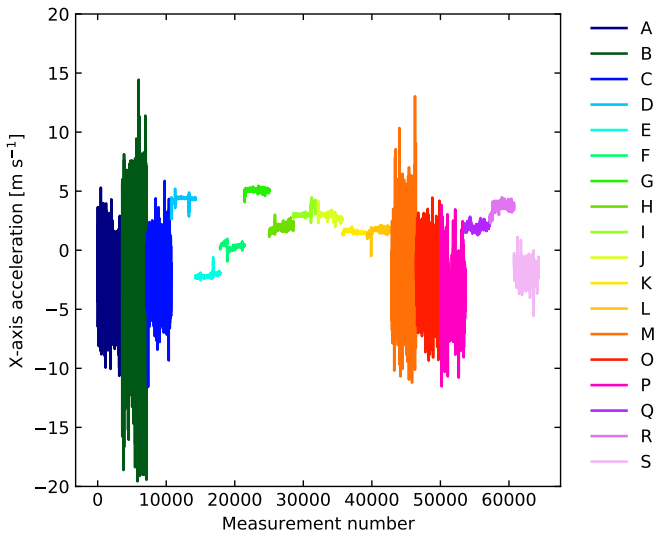

Fig. 1. X-axis acceleration per activity of user 1600.

Table 2. Strucure of the raw data files

Column	Value type
User ID	Integer (0–50)
Activity code	String of text
Unix timestamp	Integer
Measurement on the x-axis	Floating point value
Measurement on the y-axis	Floating point value
Measurement on the z-axis	Floating point value

Wear 1.5, and was worn by the participants on their dominant hand.

The data provided consists of a series of files that contain 5 columns, summarised in Table 2. The accelerometer measurements are in units of m s^{-2}, and the gyroscope measurements are in rad s^{-1}. Time is measured using Unix timestamps. The conversion between physical time and timestamp is not provided in the dataset description, and its values are not unique for each file or activity. However, usable time information comes from the fact that measurements were recorded every 50 ms. An example of a dataset is provided in Figure 1.

In total, the dataset consists of 204 `.txt` files. Each participant has one file with the data collected by the smartphone accelerometer, and another with the readings from the smartphone gyroscope. The two other files contain the data from the smartwatch in a similar way.

2.2. Data processing

The dataset described in Section 2.1 consists of a time series of smartwatch and smartphone measurements. By themselves, the measurements are not informative to describe the activity that the user is performing, but useful features can be produced by processing data points over a time interval.

Weiss (2019) describe the process by which they grouped and process the data and generated a set of 93 quantities from it, some of which have been employed as ML features to train algorithms.

After exploring the data files and the features described (and made available) by Weiss (2019), we performed an independent data processing algorithm to cross-validate their findings. We limited the initial study to 45 features (including the activity label) equivalent to some of the 93 features proposed, which we summarise in Table 3. Given the limited description on how the values of the features provided by Weiss (2019), we were unable to match the exact numeric values they provide, but we found our variables to be useful, likely equivalent parameters nonetheless. The names of the features described in Table 3 correspond to those used in our python scripts, but they are roughly equivalent to those employed by Weiss (2019).

It is noteworthy that some of the raw data files (specifically the files that correspond to participant 1629) was partially redundant, as it contained the same data repeated twice. This data was removed from our analysis to avoid biasing our results towards participant 1629.

To analyse the data in an equivalent way, the features in our fiducial model are generated each 200 time steps. This corresponds to using a time window of 10 s. However, in Section 3.2 we analyse the effect of varying the length of integration time in the accuracy of predictions of one of our ML algorithms.

2.3. Machine learning algorithms

Given the properties of the data and the outcome of the data processing, we decided to implement 5 ML classifier algorithms, using their implementations from Python's `sklearn` package (Pedregosa et al. 2011). We selected 5 supervised ML classifiers, since the labels for the activities were available with the data. We selected the algorithms not only because we felt like they suited the data, but also because they allow a broad comparison to the results from Weiss et al. (2016). We briefly describe the algorithms below. We apply all algorithms with one choice of hyperparameters, but highlight that an exploration of the accuracy of each classifier as a function of the hyperparameters would be necessary before implementing them in new data.

2.3.1. k-nearest neighbours

The k-nearest neighbours (KNN) algorithm is one of the most simple and popular ML algorithms (Fix & Hodges 1989). It is a supervised, non-parametric algorithm used often for classification and regression. In essence, it classifies a point in a given dataset by finding the k points nearest to it in the feature space, and giving it the most popular label in the group of k points.

We employ the `KNeighborsClassifier` algorithm from `sklearn`, and use the default choice of hyperparameters, except for the number of neighbours used for our classifier, which we set to `n_neighbors=3`. The standard settings implement a uniform weight for the k nearest points (i.e. they are all equally important in the classification), and automatically chooses the algorithm to measure distances between points from a set of 3 different options.

2.3.2. Decision tree classifier

A decision tree (DT) classifier is a simple supervised ML classifying algorithm (Breiman et al. 1984). DTs are non-parametric models assembled during the training phase that describe the relations between features and target labels in a tree-like structure.

We employ the `DecisionTreeClassifier` and use the default choice of hyperparameters from `sklearn`. The main hyperparameters include for a DT classifier:

- `max_depth`: the maximum number of levels in each decision tree. Set to `None`.

Table 3. Description of the features calculated and employed to train ML algorithms. The number in parentheses indicates the number of features described in each variable.

Variable	Data type	Description
`activity`	Character string	Activity label (as described in Table 1).
`subject_id`	Integer	Unique identifier for each subject (number between 1600 and 1650)
`X0-X9`	Floating-point number (10)	Binned distribution of measurements in the x-axis, distributed accross 10 bins, between the maximum and minimum value of each time segment. The total sum of these values adds up to 1.
`Y0-Y9`	Floating-point number (10)	Similar to X0–X9, but for y-axis measurements.
`Z0-Z9`	Floating-point number (10)	Similar to X0–X9, but for y-axis measurements.
`XYZ_avg`	Floating-point number (3)	Average value of x, y and z measurements during the assigned time segment.
`XYZ_peak`	Floating-point number (3)	Defined here as the time between the maximum and minimum values of each variable during a given time segment.
`XYZ_stddev`	Floating-point number (3)	Standard deviation of the measurements during the given time segment.
`XYZ_var`	Floating-point number (3)	Variance of the measurements during the given time segment.
`norm`	Floating-point number	Average value of the norm of the measurement vector (norm = $\sqrt{x^2 + y^2 + z^2}$) over the given time segment.

- `min_samples_split`: the minimum number of samples required to split an internal node. Set to 2.
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node. Set to 1.
- `max_leaf_nodes`: The number of nodes grouped to select the best between them; i.e. those that show a relative reduction in impurity. Set to `None`.

2.3.3. Random forest classifier

The random forest (RF) algorithm is a widely used ensemble algorithm for classification and regression (Breiman 2001). Ensemble methods combine several algorithms, or the information of a single algorithm obtained with different training sets to improve the accuracy of their predictions.

RF algorithms rely on a set of many DTs by selecting random subsets of the data and features, and assign a classification to a given data point by choosing the classification obtained by the majority of individual DTs.

We employ the `RandomForestClassifier` from `sklearn` and use the default choice of hyperparameters. The most relevant hyperparameters in this algorithm include the same ones employed in an indidual DT (and we use the same values for them), but also include `n_estimators`, the number of decision trees (set to 100) and `max_samples`, the number of samples to consider when the algorithm draws from the training data set to train each base estimator (set to `None`).

2.3.4. Multi-layer Perceptron classifier

A Multi-layer Perceptron classifier (MLPC) is a supervised ML algorithm, a basic type of artificial neural network that relies on multiple layers of perceptrons, essentially a set of functions that decide whether or not a set of points belongs to a certain classification (Hinton 1990).

We employ `MLPClassifier` from `sklearn` and use the default choice of hyperparameters. The most relevant hyperparameters in this algorithm include:

- `hidden_layer_sizes`: Number of layers and the number of nodes we want to have in the neural network classifier. Set to (100,).
- `activation`: The activation function for the hidden layer. Set to "`relu`".
- `solver`: The solver for weight optimisation. Set to "`adam`".
- `alpha`: Strength of the L2 regularization term. The L2 regularization term is divided by the sample size when added to the loss. Set to 0.0001.
- `learning_rate`: Learning rate schedule for weight updates. Set to "`constant`".

2.3.5. Support Vector Classification

A support Vector Classification (SVC) algorithm is a type of supervised ML algorithm that creates hyperplanes of data and finds the hyperplanes that create the largest distance between classes in training data. It then samples the input data in said hyperplanes and classifies them according to the distance to the training data clusters (Platt et al. 1999).

We employ the `SVC` from `sklearn` and use the default choice of hyperparameters. The most relevant hyperparameters in this algorithm include:

- `C`: A measure of the strength of the regularization. Set to 1.0.
- `kernel`: Specifies the kernel type to be used in the algorithm. Set to "`rbf`".
- `degree`: Degree of the polynomial kernel function. Set to 3.

3. Results

3.1. Overview of the different ML algorithms

The overall accuracy of each of the ML algorithms in classifying our datasets is given in Table 4. Using the hyperparameter values described in Section 2.3, we find that the best algorithm is consistently the RF algorithm. We also find that the best predictor for individual activities is the smartwatch gyroscope, while the worst predictor is the smartphone gyroscope.

As an example confusion matrix from the RF model on the smartwatch gyroscope data is shown in Figure 2. This confusion matrix comes from our best model, that has 80.8% accuracy in predicting the activities performed by the users. It becomes clear from the overall results (available in the attached jupyter notebook) that all the classifiers are relatively able to distinguish groups of activities that are very different from each other, but they are not as effective at distinguishing between activities that produce similar motion patterns.

To test this, we re-grouped the activities, following Weiss et al. (2016), into three broader categories:

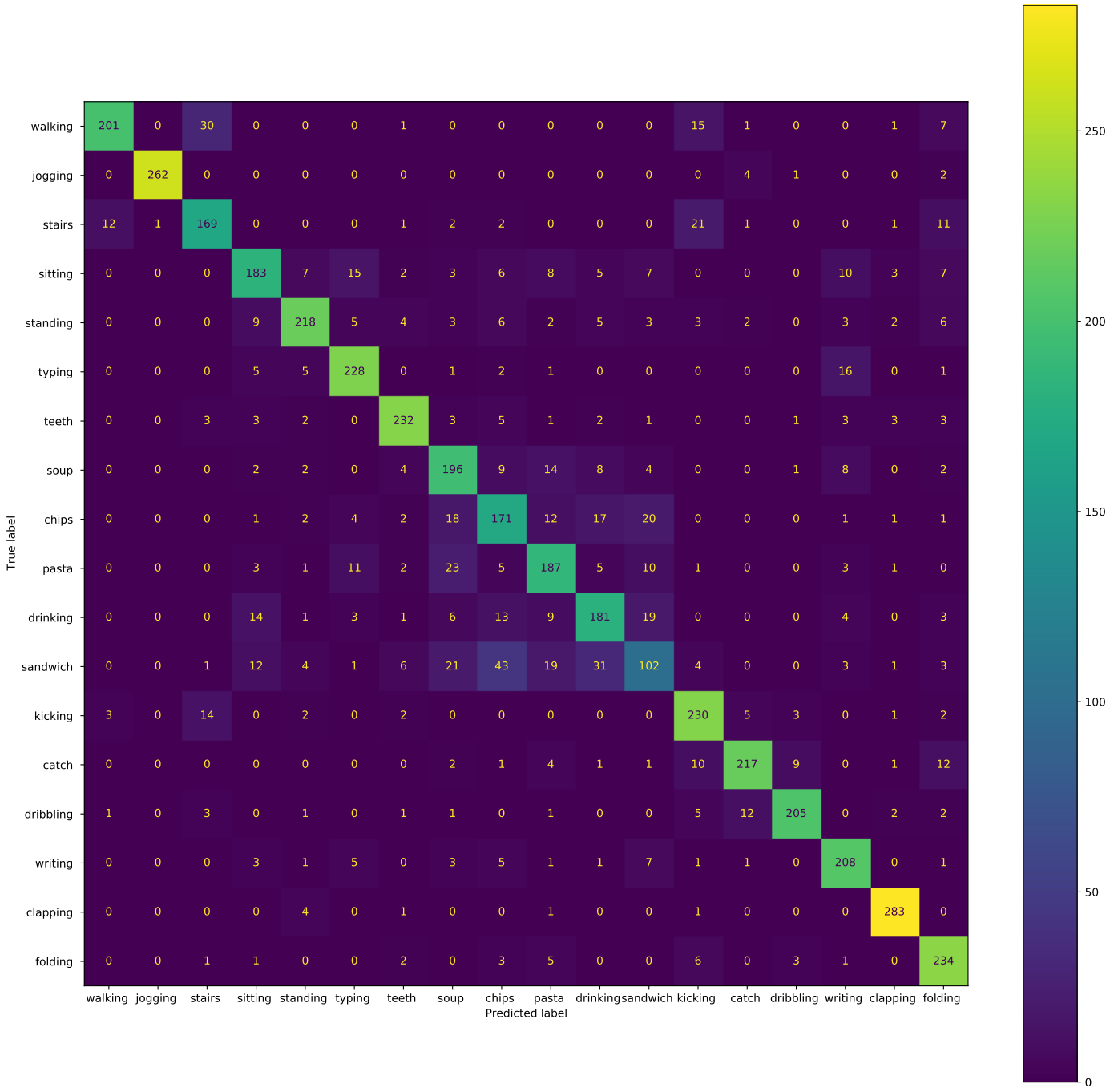

Fig. 2. Confusion matrix of the RF model applied on the smartwatch accelerometer data, and the labels described in Table 1.

Table 4. Accuracy of each ML algorithm on the 4 available datasets, using 43 features and 18 labels.

Algorithm	Phone accel.	Phone gyro.	watch accel.	watch gyro.
KNN	47.9%	17.0%	62.7%	30.4%
DT	62.9%	31.3%	68.5%	46.8%
RF	67.9%	42.8%	80.8%	63.5%
MLPC	48.9%	32.6%	71.9%	57.4%
SVC	27.9%	19.4%	54.5%	43.1%

– General (not hand-oriented): walking, jogging, climbing stairs, sitting, standing and kicking.
– Hand-oriented activities: dribbling a basketball, playing catch, typing, handwriting, clapping, brushing teeth and folding clothes.
– Eating activities: eating pasta, eating soup, eating a sandwich, eating chips and drinking from a cup.

After re-classifying the data in these three categories, our models show a much better performance, as shown in Table 5. As expected, all algorithms have a better performance when the number of labels is reduced. The confusion matrix for the model using the smartwatch accelerometer data is shown in Figure 3.

The models presented in Table 4 correspond to the impersonal models presented by Weiss et al. (2016) in their Table III. We find that our ML models have a much better accuracy than

Table 5. Accuracy of each ML algorithm on the 4 available datasets, using 43 features and 4 labels

Algorithm	Phone accel.	Phone gyro.	watch accel.	watch gyro.
KNN	72.1%	53.3%	82.3%	60.0%
DT	82.1%	56.9%	85.5%	69.0%
RF	80.7%	68.9%	90.1%	80.9%
MLPC	71.2%	64.7%	87.8%	76.5%
SVC	60.1%	60.1%	72.7%	61.2%

Fig. 3. Confusion matrix of the RF model applied on the smartwatch accelerometer data, using a reduced sample of labels.

those presented in the previous study. We attribute this to the data analysis process, as we used very similar algorithms, but a thorough analysis is necessary to confirm this hypothesis.

3.2. Effect of time window variations

Weiss (2019) describe the process of data analysis they employ to reduce the raw data into the features that are used in their ML models. They cluster groups of 200 data points and calculate the distribution of measurements across this time window, suggesting that a time window of 10 s is useful to produce low-latency predictions of the activity performed by the user. However, a choice of 10 s of integration is arbitrary, and the consequences are not explored.

Here, we perform a test to see if the length of the time window affects the accuracy of ML classifiers in predicting the activity performed by the user. We cluster the data in groups of 50 to 500 time steps, increasing 10 steps per measurement, and we compute the accuracy of the RF model using the smartwatch accelerometer data, as it was found in Section 3.1 to be the most accurate model.

Our results are summarised in Figure 4. The measurement of an ideal number of time steps per integration is uncertain due to random fluctuations that arise from the random selection of the data, and they should be account for in a more formal manner. However, this exercise demonstrates that the number of time steps integrated plays a significant role in the accuracy of the ML model, resulting in a variation of up to ∼5% in model accuracy. Too short integrations, of order 100 steps, corresponding to a prediction every 5 s, are far from the point where the ML model has maximum accuracy. Our exercise suggests that a time step

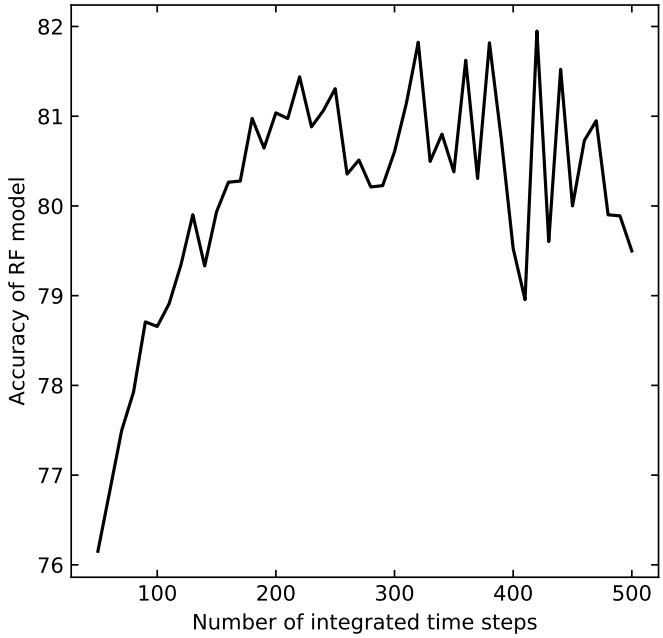

Fig. 4. Accuracy of the ML model as a function of the length number of time steps used to produce a single set of ML features from the raw data.

of 10 s, or 200 integrated timesteps –the value chosen by Weiss et al. (2016)– seems to be close to the area where the accuracy of the ML model is maximised.

References

Breiman, L. 2001, Machine learning, 45, 5
Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. 1984, International Group, 432, 9
Fix, E. & Hodges, J. L. 1989, International Statistical Review/Revue Internationale de Statistique, 57, 238
Hinton, G. E. 1990, in Machine learning (Elsevier), 555–610
Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825
Platt, J. et al. 1999, Advances in large margin classifiers, 10, 61
Weiss, G., Timko, J., Gallagher, C., Yoneda, K., & Schreiber, A. 2016, 426–429
Weiss, G. M. 2019, UCI Machine Learning Repository: WISDM Smartphone and Smartwatch Activity and Biometrics Dataset Data Set, 7, 133190