1. Introduction

Human activity recognition (HAR for short) is a broad field of study concerned with identifying the specific movement or action of a subject based on sensor data. The sensor data may be remotely recorded, such as video, radar, ultrasound or other wireless position. Also, data may be recorded directly on the subject such as by carrying custom hardware with sensors.
For our project, we are going to try classifying different activities performed by several persons through the accelerometer and gyroscope that subjects will carry on their smartphones and smartwatch during the activity lapse.
The dataset to work with has been obtained from the University of California Irvine Machine Learning Repository, and it is called WISDM Smartphone and Smartwatch Activity and Biometrics Dataset. This dataset has been created by the Fordham University (New York, USA) and donated to UCI's dataset repository with date of October 6, 2019.
In order to do the task of classification, we are going to compare the following techniques:

- **Random Forests** applying statistics metrics.
- **Long-short Term Memory Recurrent Neural Network** applying timeseries.

Description of the Dataset

The dataset contains 204 csv files distributed into 2 folders splitting data from smartphone and smartwatch. In each folder there are two new folders splitting files that come from the accelerometer and the gyroscope sensors. In each of these 4 folders (phone/accel, phone/gyros, watch/accel, and watch/gyro) there are 51 files, one for each subject.
All the csv files include the sensor data for the 18 activities at a rate of 20 Hz (data every 0.05 seconds), and all the activities have lasted around 3 minutes. The activities recorded can be classified into 2 main categories:

- Non-hand-oriented activities: walking, jogging, stairs, sitting, standing and kicking (a soccer ball).
- Hand-oriented activities: typing, brushing teeth, eating soup, eating chips, eating pasta, drinking from cup, eating a sandwich, playing catch with tennis ball, dribbling (basketball), writing, clapping and folding clothes.

So, we have around 3600 datapoint (for each sensor) for each activity done by one specific subject. Both accelerometer and gyroscope print the instantaneous lecture in x, y, and z axis. These datapoints are grouped into timeseries (from the start to the end of each activity), so, as there are 51 subjects and 18 activities, there is a total of 918 timeseries to analyse.
Inspecting one of these datasets (e.g. phone/accel) we have the following datapoints for each subject:

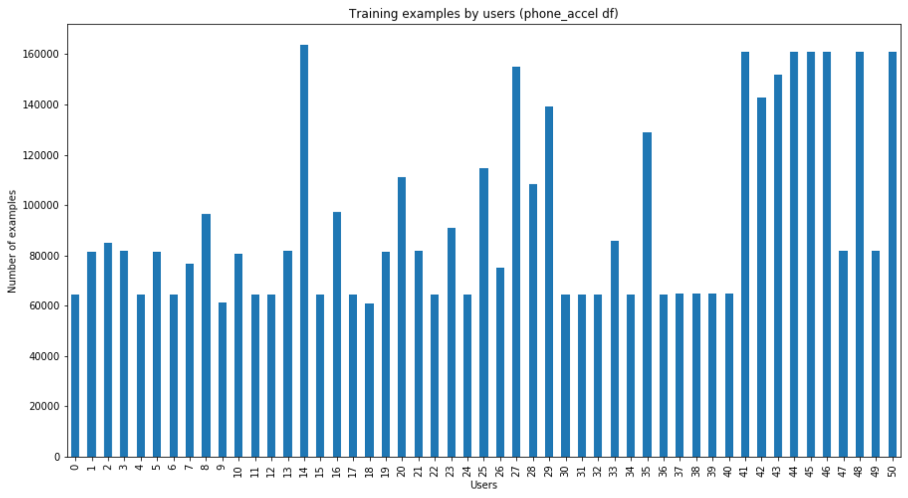


Figure 1. Number of datapoints for phone/accel datasets

2. Data Pre-Processing

Visual and numerical inspection

First, we have checked the missing values included on datasets to remove them. Inspecting Figure 1, we have noticed that datasets do not contain the same number of datapoints for every user and activity (even for certain pair user-activity there are not data records). Due to this, we have removed all datapoints above the index 3567 (in every timeseries), in order to have the same long for all data (or 0 if we do not have data). Plotting one dataset such as phone/accel for user 0 and activity 10 we have as follows:

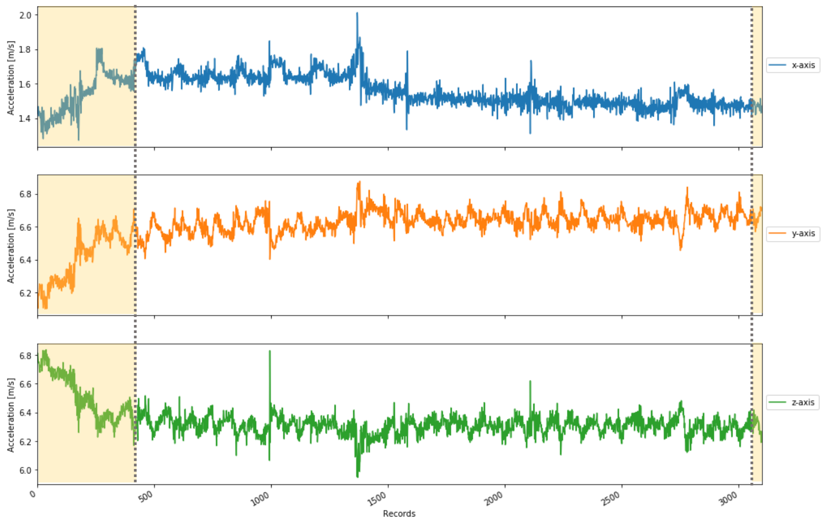


Figure 2. X, Y, and Z axis records of phone accelerometer for user 0 and activity 10

Doing a visual inspection for different users and activities, we have found the following:

- At the start (first 500 records) in most of the plots appears a non-seasonal behaviour. We have remarked this behaviour on the semi-transparent orange squares plotted on Figure 3 at the very beginning of each plot.
- At the end (first 500 records) in some of the plots appears a non-seasonal behaviour. We have remarked this behaviour on the semi-transparent orange squares plotted on Figure 3 at the very end of each plot.

As the records are showing a temporal repetitive behaviour, we have decided to trim values at the beginning and at the end, having to analyse the values from index 500 to index 3100 in every dataserie (from second 25 to second 155, indicated between grey dotted lines in Figure 3). Then, all single timeseries will have 2600 entries.

Data augmentation (feature engineering)

In order to have more data to run the classification algorithms, we have obtained different features from the original dataset. From the accelerometer, which records the instant linear acceleration (in $m \cdot s^{-2}$), we have calculated the over-acceleration (or jerk) and linear velocity. From the gyroscope, which records the angular velocity (in $rad \cdot s^{-1}$), we have calculated the angular acceleration and over-acceleration. All features are represented in 3 orthogonal axes.

Normalization

Before creating the dataset to be passed on the different classification algorithms, we have normalized the data between -1 and 1. This allows us to compare same activities, even when they have been realized with different strong or intensity (e.g. the activity of kicking a ball, it must not matter if one subject kicks stronger than a second one). We have joined all data into Pandas *DataFrames* encoded by the pair activity-user.

Data has been normalized through the function:

$$x_i = 2\left(\frac{x_i - x_{min}}{x_{max} - x_{min}}\right) - 1$$

Being x_{min} and x_{max} the global maximums for one specific features of all users and activities.

Data preparation

In order of having more datasets to use as training, validating and testing features, we have cropped all timeseries to 200 entries. This means that, from the previously trimmed timeseries with 2600 entries, we have split it in 13 timeseries of 200 entries each. As we have stated before, as all the activities done by the subjects are repetitive, we can assume that no information has been lost in this process. For all experiments, and in order to be able to compare performances between techniques, we have set the test data as all activities done by subjects from 40 to 51.

3. Comparison of Classify Methods

Random Forest

Summary of the model

Random forests are an ensemble learning method for classification or regression that trains several decision trees and outputs the class that is the mode of the predicted classes or mean prediction of the individual trees. These models have been one of the most successful machine learning models in Kaggle competition history.

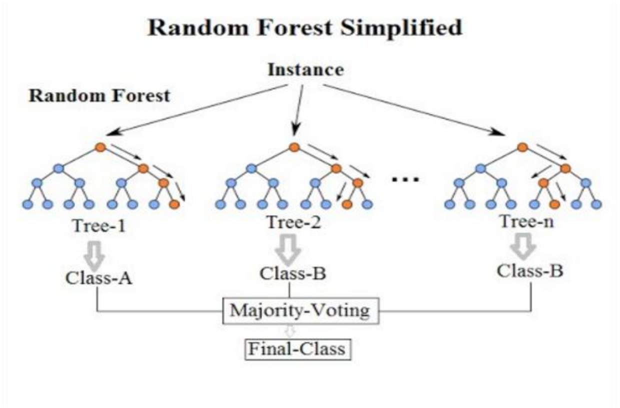


Figure 3. Random Forest tree [image from https://medium.com]

Furthermore, we assessed the following pros and cons.

Pros	Cons
It can handle higher dimensionality very well, and identify important features	Doesn't perform well for regression problems as it doesn't predict beyond the range in the training data
Overcomes high variance with techniques such as bootstrap sampling	Hard to fine-tune and interpret the model
Balancing errors in data sets where classes are imbalanced.	High computation cost and memory with growing data and features

Table 1. Random forests pros and cons

Hypothesis and model selection

We have engineered a total of 108 features, making our training very hard using Tree based algorithms such as Random Forest. To strategize feature reduction, we have used an open-source (Koehrsen, 2019) *Github* repository (feature-selector) which employs the following 5 methodologies to reduce our total features:

a. Missing Values: find columns with a missing fraction greater than a specified threshold. Our selected threshold is 60%. In our case, 0 such features were found.
b. Single Unique Value: find any features that have only a single unique value. In our case, 0 such features were found.
c. Collinear (highly correlated) Features: this method finds pairs of collinear features based on the Pearson correlation coefficient. For each pair above the specified threshold (in terms of absolute value), it identifies one of the variables to be removed. 16 features were identified for removal.
d. Zero Importance Features: this method relies on a machine learning model to identify features to remove. It therefore requires a supervised learning problem with labels. The method works by finding feature importance using a gradient boosting machine implemented in the *LightGBM* library. 0 such features were identified.
e. Low Importance Features: this method builds off the feature importance from the gradient boosting machine (zero importance features must be run first) by finding the lowest importance features not needed to reach a specified cumulative total feature importance. For example, if we pass in 0.99, this will find the lowest important features that are not needed to reach 99% of the total feature importance. 7 such features were identified.

We have selected 88 features out of our 108 features after processing these 5 steps. We have used the default parameters of the Random Forest Classifier from *Scikit-learn* library and used it as a baseline for comparison. We have chosen Random-Cross Validation search methodology to narrow down our search space by randomly selecting a combination of parameters from a pool of choices such as number of estimators, tree. It greatly reduces the training time for the search space and returns the best model after running cross-validation.

- N estimators: number of trees in random forest.
- Max depth: number of features to consider at every split.
- Maximum number of levels in tree.
- Min samples split: minimum number of samples required to split a node.
- Min samples leaf: minimum number of samples required at each leaf node.
- Bootstrap. Method of selecting samples for training each tree.

We have used the following initial Random search:

N estimators	Min samples split	Min samples leaf	Max features	Max depth	Bootstrap
[200 – 2000] increment of 10	{2, 5, 10}	{1, 2, 4}	{‘auto’, ‘sqrt’}	[10 – 110] increment of 10	{‘True’, ‘False’}

Table 2. Initial hyperparameters grid search

We further did a grid search on the narrowed down search space to reach the optimal parameters, which exhausts all the parameter options provided.

N estimators	Min samples split	Min samples leaf	Max features	Max depth	Bootstrap
{900, 1000, 1100}	{2, 4}	{1, 2, 3}	{‘sqrt’}	{100, 110, 120}	{‘True’}

Table 3. Fine-tuning hyperparameters grid search

Results

Our base model achieved a classification accuracy of 64%. We achieved 70% accuracy after running the Randomized Cross-Validation Search and further improved our accuracy to 71% after running the Grid-Search Cross-Validation.
Our final best parameters were as follows:

N estimators	Min samples split	Min samples leaf	Max features	Max depth	Bootstrap
1100	2	1	‘sqrt’	110	‘True’

Table 4. Final hyperparameters

We have obtained the following confusion matrix:

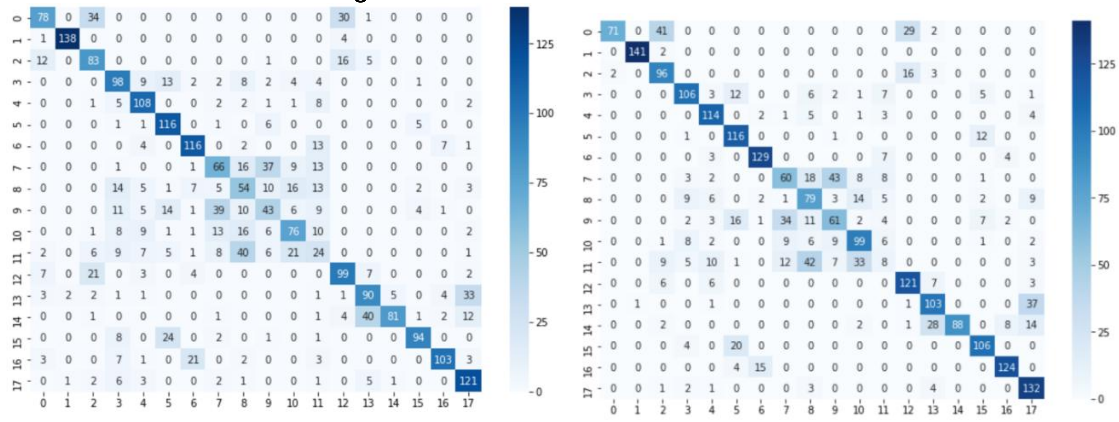


Figure 4. Confusion matrix for base model (left, 64% accuracy) and final model (right, 71% accuracy)

Also, we have accomplished the following F1-Score:

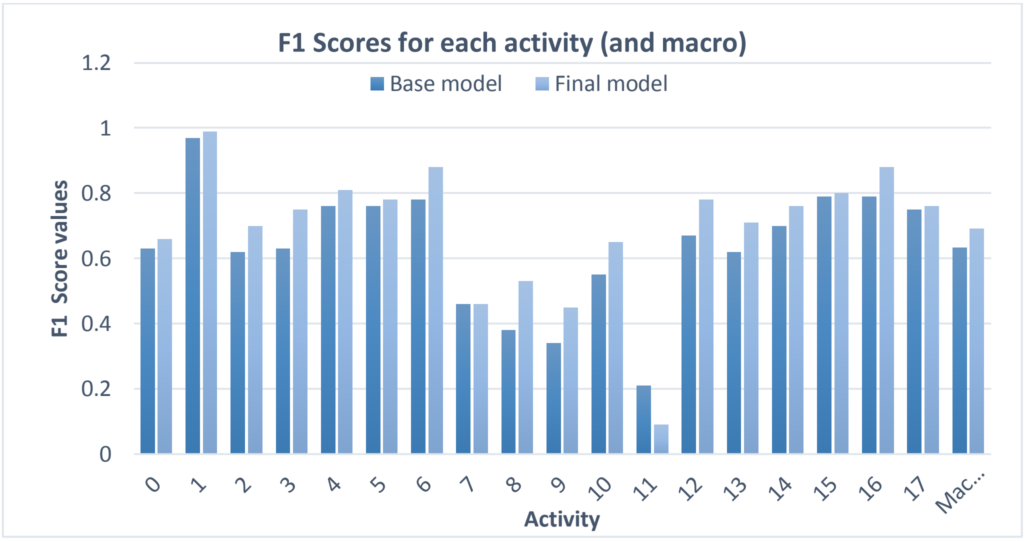


Figure 5. F1 Scores for each activity (and macro-averaged F1 Score)

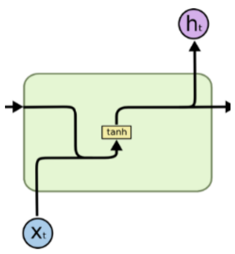
Evaluation of results

As evident by the F1 scores of the different classes, our final model performs much better as compared to the base model. The number of estimators i.e. 1100, is very large as compared to the base model i.e. 100 which improves the performance drastically.
We also observe that both our models perform poorly for class 11, which is Eating Sandwich, and upon examining the confusion matrix, this class is often confused with other similar activities such as Eating Soup, Eating Chips, Eating Paste and Drinking. The results do make sense as it is very hard to differentiate the readings of the Gyroscope and Accelerometer in all these cases.

LSTM Recurrent Neural Network

Summary of the model

Long-short term memory recurrent neural networks (LSTM, for short), are a variation of classical recurrent neural networks (RNN, for short). RNN deals very well with short term memory, but many problems arise when we try to capture elements stored in memory very far away from the current state. As the gradient descent decays quickly during backpropagation, the further the interest input information (past state) is from the actual state, the past state will be more vanished. In order to solve this, LSTM networks were designed. Classical RRN have an addition function (usually tanh) than sums up the actual input with the previous state. LSTM neurons have more elements, being the so-called cell state the main one, which is the arrow that pass horizontally through the neuron in from state to state. This allows the information to go through different states almost with no modification. Also, the have some features named gates that allows to do small modifications on the cell state (adding or forgetting information, changing the weights of the actual information, and giving the output).



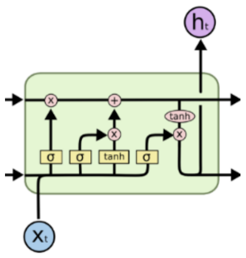


Figure 6. Simplest model of RNN neuron (left) and LSTM neuron (right)
[images from https://medium.com]

The pros and cons of LSTM networks are the same ones than for RNN (excluding the long-short term problem of RNN). They are listed on the following table:

Pros	Cons
Possibility of processing input of any length	Computational expensive
Model size not increasing with size of input	They cannot process very long sequences due to vanishing problems (as they sue sigmoid and tanh activation functions)
Weights are shared across time	Requires much more training

Table 5. Main pros and cons of LSTM networks

Hypothesis and model selection

We have selected a simple LSTM network[1]. In order to reach the best performances of the model and trying to obtain more conclusions, we are going to follow the next steps:

1. Choose dataset for obtaining best accuracy input (with hyperparameters fixed):

Dataset	Phone/accel	Phone/gyro	Watch(accel)	Watch/gyro	All
Timeseries	11934	11934	11934	11934	47736

Table 6. Dataset configuration for LSTM inputs

2. Being fixed the dataset that has performed better on previous step, we are going to do the following grid search for hyperparameters optimization:

Learning rate	Lambda L2 reg.	Batch size	N hidden units
{0.1, 0.01, 0.001, 0.0001}	{2, 0.2, 0.02, 0}	{500, 1000, 1500, 2000}	{20, 40, 60, 80}

Table 7. Hyperparameters grid-search

3. Run the best model for 150 epochs to get final results.

In order to compare performances, we are going to plot test and train loss and accuracy, confusion matrix, F1 Score for each activity and macro-averaged F1 Score.

Results

The results for step 1 are plotted on the next figure:

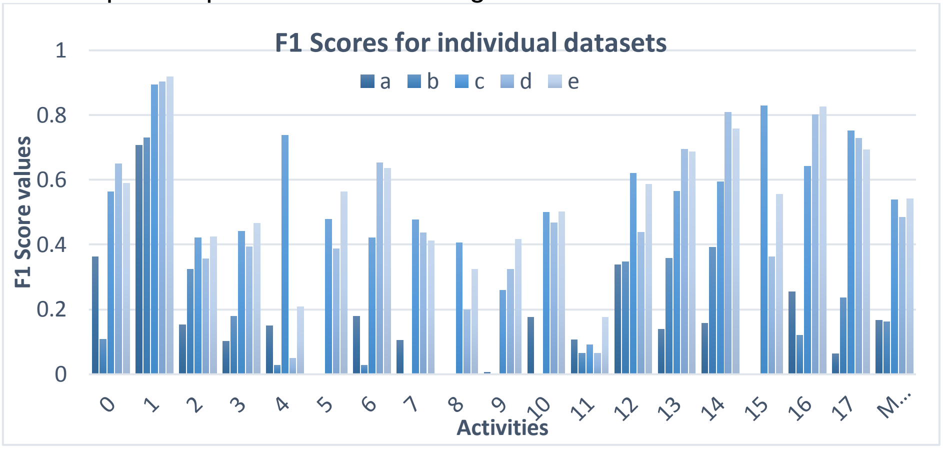


Figure 7. F1 Scores for different datasets (as indicated on step 1)

Secondly, being set the watch/accel as the reference dataset for do the hyperparameters optimization, we have done all the grid search indicated on step 2, having the following results (test accuracy last epoch inside parenthesis):

Learning Rate	**0.1**	**0.01**	**0.001**	**0.0001**

[1] Hyperparameters initial setting: 50 epochs, batch size 500, learning rate 0.01 and lambda L2 regularization 0.02, from recommendations found in (Heaton, 2008).

Macro F1 Score	0.0450 (7.87%)	0.4375 (50.21%)	0.5329 (56.24%)	0.5026 (52.00%)

Lambda L2 reg.	**0.2**	**0.02**	**0.002**	**0**
Macro F1 Score	0.0058 (5.55%)	0.4375 (50.21%)	0.5413 (56.07%)	0.5470 (57.95%)

Batch size	**500**	**1000**	**1500**	**2000**
Macro F1 Score	0.4375 (50.21%)	0.4124 (46.41%)	0.4499 (47.4%)	0.5160 (52.82%)

N hidden neurons	**20**	**40**	**60**	**80**
Macro F1 Score	0.2830 (36.02%)	0.3515 (40.85%)	0.4375 (50.21%)	0.3174 (37.82%)

Table 8. Results from step 2 (on yellow, baseline state for other hyperparameters during specific hyperparameter optimization)

Finally, step 3 has been run with following hyperparameters:

Learning rate	Lambda L2 reg.	Batch size	N hidden units
0.001	0	2000	60

Table 9. Best model hyperparameters

Giving as a result:

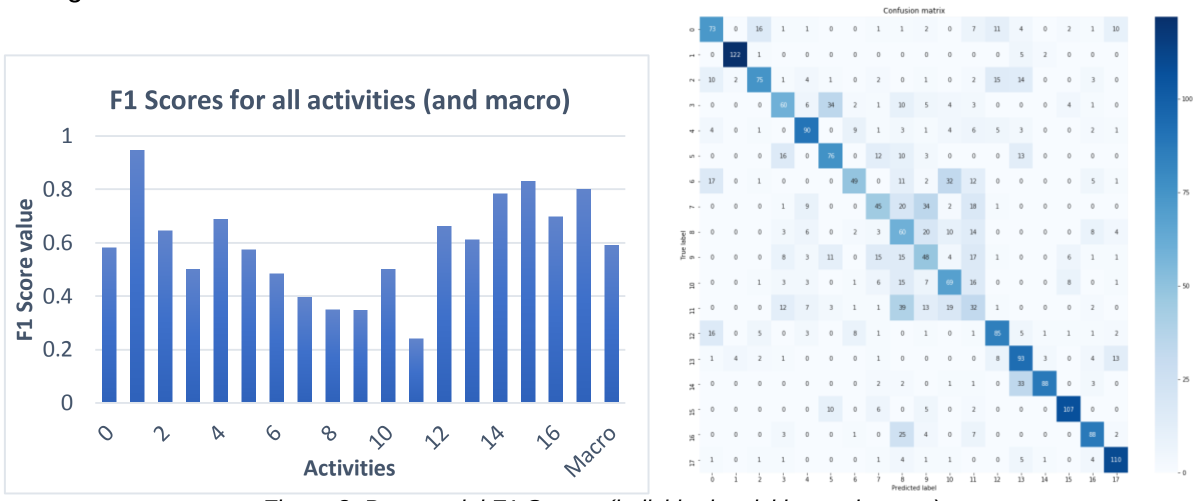


Figure 8. Best model F1 Scores (individual activities and macro)

Evaluation of results

From the first step, we can infer that information coming from the smartphone (in the way that the experiment has been carried out) is more useful than information coming from the smartphone. This can be explained through the fact that most of the activities are hand-oriented (12 vs. 6 activities non-hand-oriented). The next figure is a comparison between the accuracy of LSTM with the four different datasets, doing only a binary classification between non-hand and hand-oriented activities:

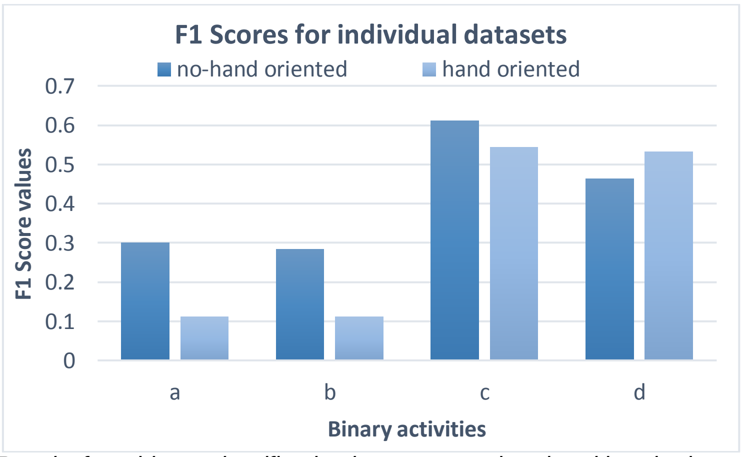


Figure 9. Results for a binary classification between non-hand and hand-oriented activities

As we can observe, non-hand-oriented activities are much better recognized through smartphones' sensors. However, the smartwatch's sensors are also able to classify these activities, even with considerable much better accuracy.
About final results, we have reached an overall F1 Score about 60%. Also, we can observe that most of activities are recognized with overalls scores above 50%, while only a few of them have less than 40% score.

4. Conclusions

Main conclusions of the project

As our main conclusion, we can state that both the algorithms have performed well in the task of time-series classification with a large number of labels. Both Random Forests and Long-Short Term Memory networks are well-known as good classifiers, even for timeseries, where the data is harder to process due to the coupled trend, seasonality and noise. However, we have reached more than 70% averaged accuracy with random forests around 60% with LSTM. So, random forests perform better the assigned task. Probably, there is still way to improve LSTM hyperparameters and reach better accuracies.
We can observe that there are some tasks are easier to classify and some are comparatively harder. The task of eating a sandwich (indexed as 11) has been almost indetectable in both methods. Tasks like eating soup (7), chips (8) and pasta (9) are the following in the worst detectable tasks, having less 50% accuracy in both techniques. On the other hand, jogging (task 1) is the best recognized in both techniques, with more than 90% of accuracy. The best tasks recognized are standing (4), brushing teeth (6) and clapping (16). From this we can infer that our algorithms do not perform very well on eating activities. We also observe in both the confusion matrix (for random forests and LSTM) that, on these specific tasks, algorithms tend to miss between them (they are confusing eating activities with other eating activities, not with non-eating related task):

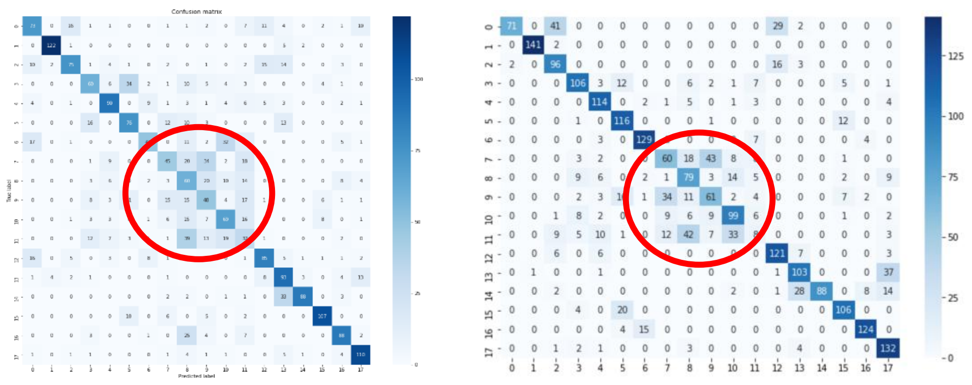


Figure 10. Circled in red the confusion between eating activities (random forests at left and LSTM at right)

So, if we join all eating tasks on one single label, probably the overall performance will increase substantially.

In a nutshell, we have proved that both Random Forests and LSTM networks can do the task of timeseries classification with just a few parameters optimization, having good enough results for most of the problems.

Future work

As one of the main handicaps found doing this project has been the limited dataset for each feature to classify, one possible future work could be increasing the size of datapoints through different techniques of data augmentation. One could be switching the sign of the original dataset: since the orientation of the sensor devices (smartwatch and smartphone) does not have to influence on the classification results, changing the sign of one full column of the dataset (e.g. -x acceleration instead of x acceleration) should carry to the same activity.
We have explored other classification techniques such as Dynamic Time Warping, Multi-Layer Perceptron, Support Vector Machines, and Xgboost which have showed great performance on classification tasks, but we have not mentioned in this work due to not being developed enough by us. A future work might be implementing these algorithms (and others) and compare them with the tested ones.
Finally, we have not mentioned anything about time and memory cost of the techniques applied. In future works, we think that this is an important aspect to be considered before deciding between different classification techniques.

5. References

Atienza, R. (2017). *LSTM by Example Using TensorFlow*, [Online]. Available at: https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537

Brownlee, K. (2018). *A Gentle Introduction to Standard Human Activity Recognition Problem,* [Online]. Available at: https://machinelearningmastery.com/how-to-load-and-explore-a-standard-human-activity-recognition-problem/

Koehrsen, W. (2019). *WillKoehrsen/Feature-Selector*, [Online]. Available at: https://github.com/WillKoehrsen/feature-selector

Nakisa, B., Rastgoo, M. N., Rakotonirainy, A., Maire, F. and Chandran, V. (2018). '*Long Short Term Memory Hyperparameter Optimization for a Neural Network Based on Emotion Recognition Framework'*, IEEE Access, 6, pp. 49325 – 49338.

University of California, Irvine. (2019). *UCI Machine Learning Repository,* [Online]. Available at: https://archive.ics.uci.edu/ml/index.php

Weiss, G. M., Yoneda, K. and Hayajneh, T. (2018). '*Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living',* IEEE Access, 7, pp. 133190-133202.

Heaton, J. (2008). *Introduction to Neural Networks with Java*, 2nd edn, USA: Heaton Research, Inc.