
LSM6DSOX: Machine Learning Core

Introduction

This document is intended to provide information on the Machine Learning Core feature available in the [LSM6DSOX](#). The Machine Learning processing capability allows moving some algorithms from the application processor to the MEMS sensor, enabling consistent reduction of power consumption.

The Machine Learning processing capability is obtained through decision-tree logic. A decision tree is a mathematical tool composed of a series of configurable nodes. Each node is characterized by an “if-then-else” condition, where an input signal (represented by statistical parameters calculated from the sensor data) is evaluated against a threshold.

The LSM6DSOX can be configured to run up to 8 decision trees simultaneously and independently. The decision trees are stored in the device and generate results in the dedicated output registers.

The results of the decision tree can be read from the application processor at any time. Furthermore, there is the possibility to generate an interrupt for every change in the result in the decision tree.

1 Machine Learning Core in the LSM6DSOX

The Machine Learning Core (together with the Finite State Machine) is one of the main embedded features available in the LSM6DSOX. It is composed of a set of configurable parameters and decision trees able to implement algorithms in the sensor itself.

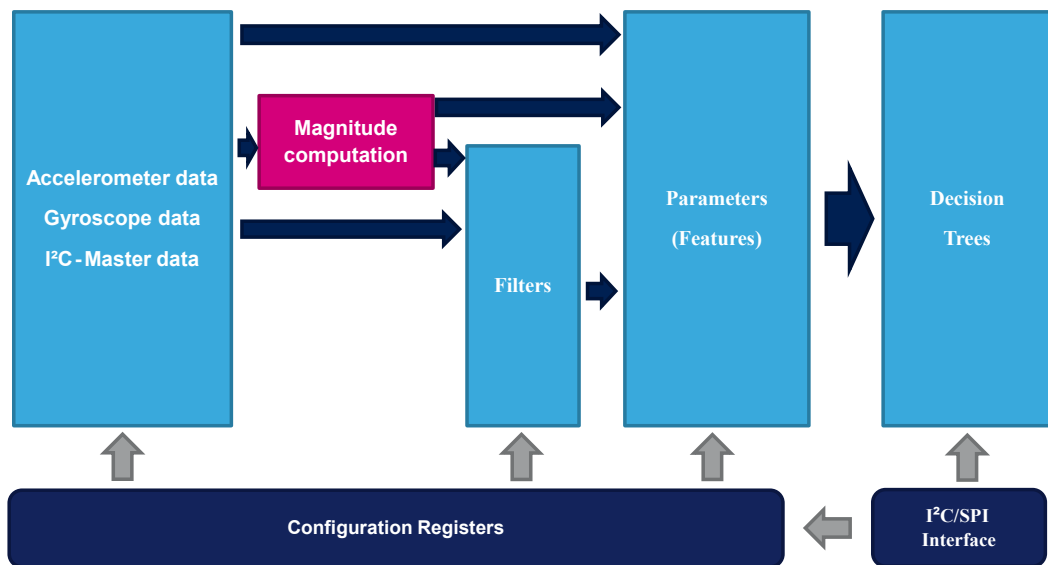
The kind of algorithms suitable for the Machine Learning Core are those which can be implemented by following an inductive approach, which involves searching patterns from observations. Some examples of algorithms which follows this approach are: activity recognition, fitness activity recognition, motion intensity detection, vibration intensity detection, carrying position recognition, context awareness, false positive rejection, etc...

The idea behind the Machine Learning Core is to use the accelerometer, gyroscope and external sensor data (readable through the I²C master interface) to compute a set of statistical parameters selectable by the user (such as mean, variance, energy, peak, zero crossing, etc...) in a defined time window. In addition to the sensor input data, some new inputs can be defined by applying some configurable filters available in the device.

The Machine Learning Core parameters are called "Features" and can be used as input for a configurable decision tree which can be stored in the device.

The decision tree which can be stored in the LSM6DSOX is a binary tree composed of a series of nodes. In each node, a statistical parameter (feature) is evaluated against a threshold to establish the evolution in the next node. When a leaf (one of the last nodes of the tree) is reached, the decision tree generates a result which is readable through a dedicated device register.

Figure 1. Machine Learning Core in the LSM6DSOX



The Machine Learning Core output data rate can be configured among one of the four available rates from 12.5 to 104 Hz. The bits MLC_ODR in the embedded function register EMB_FUNC_ODR_CFG_C (60h) allow selecting one of the four available rates as shown in the following table.

Table 1. Machine Learning Core output data rates

| MLC_ODR bits in EMB_FUNC_ODR_CFG_C (60h) | Machine Learning Core output data rate |
|--|--|
| 00 | 12.5 Hz |
| 01 | 26 Hz (default) |
| 10 | 52 Hz |
| 11 | 104 Hz |

The machine learning processing capability of the LSM6DSOX is a “supervised learning” which consists of:

- identifying some classes to be recognized;
- collecting multiple data logs for each class;
- performing some data analysis from the collected logs to learn a generic rule which allows mapping inputs (data logs) to outputs (classes to be recognized).

In an activity recognition algorithm, for instance, the classes to be recognized might be: stationary, walking, jogging, biking, driving, etc... Multiple data logs have to be acquired for every class, e.g. multiple people performing the same activity.

The analysis on the collected data logs has the purpose of:

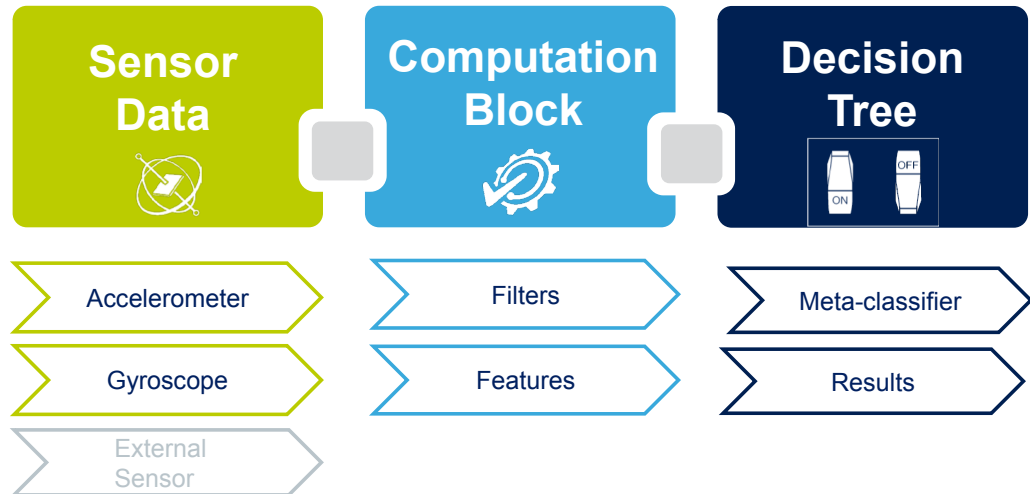
- defining the features to be used to correctly classify the different classes;
- defining the filters to be applied to the input data to improve the performance using the selected features;
- generating a dedicated decision tree able to recognize one of the different classes (mapping inputs to outputs).

Once a decision tree has been defined, a configuration for the device can be generated by the software tool provided by STMicroelectronics (described in [Section 2 Machine Learning Core tools](#)). The decision tree will run on the device, minimizing the power consumption.

Going deeper in detail on the Machine Learning Core feature inside the LSM6DSOX, it can be thought of as three main blocks ([Figure 2](#)):

1. Sensor data
2. Computation block
3. Decision tree

Figure 2. Machine Learning Core blocks



The first block, called “Sensor Data”, is composed of data coming from the accelerometer and gyroscope which are built in the device, or from an additional external sensor which might be connected to the LSM6DSOX through the I²C master interface (sensor hub).

The Machine Learning Core inputs defined in the first block are used in the second block, the “Computation Block”, where filters and features can be applied. The features are statistical parameters computed from the input data (or from the filtered data) in a defined time window, selectable by the user.

The features computed in the computation block will be used as input for the third block of the Machine Learning Core. This block, called “Decision Tree” includes the binary tree which evaluates the statistical parameters computed from the input data. In the binary tree the statistical parameters are compared against certain thresholds to generate results (in the example of the activity recognition described above, the results were: stationary, walking, jogging, biking, etc...). The decision tree results might also be filtered by an optional filter called meta-classifier. The Machine Learning Core results will be the decision tree results which include the optional meta-classifier.

The Machine Learning Core memory is organized in a “dynamic” or “modular” way, in order to maximize the number of computation blocks which can be configured in the device (filters, features, etc...). A dedicated tool has been designed to generate the configuration of the LSM6DSOX, in order to automatically manage memory usage. The tool is available in the Unico GUI and it is described later in [Section 2 Machine Learning Core tools](#).

The following sections explain in detail the three main blocks of the Machine Learning Core in the LSM6DSOX described in [Figure 2](#).

1.1 Inputs

The LSM6DSOX works as a combo (accelerometer + gyroscope) sensor, generating acceleration and angular rate output data. The 3-axis data of the acceleration and angular rate can be used as input for the Machine Learning Core.

The rate of the input data must be equal to or higher than the Machine Learning Core data rate configurable through the embedded function register EMB_FUNC_ODR_CFG_C (60h), as described in [Table 1](#).

Example: In an activity recognition algorithm running at 26 Hz, the Machine Learning Core ODR must be selected at 26 Hz, while the sensor ODRs must be equal to or higher than 26 Hz.

The Machine Learning Core uses the following unit conventions:

- Accelerometer data in [g]
- Gyroscope data in [rad/sec]
- External sensor data in [Gauss] for a magnetometer, [Bar] for a pressure sensor

Since it is possible to connect an external sensor (e.g. magnetometer) to the LSM6DSOX through the Sensor Hub feature (Mode 2), the data coming from an external sensor can also be used as input for machine learning processing.

When using an external sensor, the sensitivity of the external sensor has to be set through registers SENSITIVITY_EXT_SENSOR_L (E8h) and SENSITIVITY_EXT_SENSOR_H (E9h).

Example: For a magnetometer like the LIS2MDL, the sensitivity is 1.5 mG/LSB, which becomes 0.0015 G/LSB after converting it to Gauss, and becomes 1624h converted as HFP (half-precision floating point value for the LSM6DSOX sensitivity registers).

| Sensitivity [mG/LSB] | Sensitivity [G/LSB] | Sensitivity HFP |
|----------------------|---------------------|-----------------|
| 1.5 mG/LSB | 0.0015 G/LSB | 1624h |

Note: The half-precision floating-point format is expressed as:

SEEEEEFFFFFFFFFFFF (S: 1 sign bit; E: 5 exponent bits; F: 10 fraction bits).

The following procedure allows changing the conversion factor for the external magnetometer data:

1. Write 80h to register 01h // Enable access to the embedded function registers
2. Write 40h to register 17h // PAGE_RW (17h) = '40h': enable write transaction
3. Write 11h to register 02h // PAGE_SEL (02h) = '11h'
4. Write E8h to register 08h // PAGE_ADDRESS (08h) = 'E8h'
5. Write [LSB] conversion factor (LIS2MDL example, 24h) to register 09h
6. Write 11h to register 02h // PAGE_SEL (02h) = '11h'
7. Write E9h to register 08h // PAGE_ADDRESS (08h) = 'E9h'
8. Write [MSB] conversion factor (LIS2MDL example, 16h) to register 09h
9. Write 00h to register 17h // PAGE_RW (17h) = '00h': disable read / write transaction
10. Write 00h to register 01h // Disable access to the embedded function registers

The example of the procedure above to change the sensitivity for the external sensor is included in the configuration generated by the Machine Learning Core tool (described in [Section 2 Machine Learning Core tools](#)),

so the user just needs to set a sensitivity value in the GUI, which will be translated in the register setting from the software.

To summarize the Machine Learning Core inputs:

- Accelerometer data conversion factor is automatically handled by the device;
- Gyroscope data conversion factor is automatically handled by the device;
- External sensor data conversion factor is not automatically handled by the device. A conversion factor must be set by the user in order to make the Machine Learning Core work with the correct unit of measurement.

An additional input available for all sensor data (accelerometer, gyroscope, and external sensor) is the norm. From the 3-axis data the Machine Learning Core (in the LSM6DSOX) internally computes the norm and the norm squared. These two additional signals can be used as inputs for machine learning processing.

The norm and the norm squared of the input data are computed with the following formulas:

$$V = \sqrt{x^2 + y^2 + z^2}$$

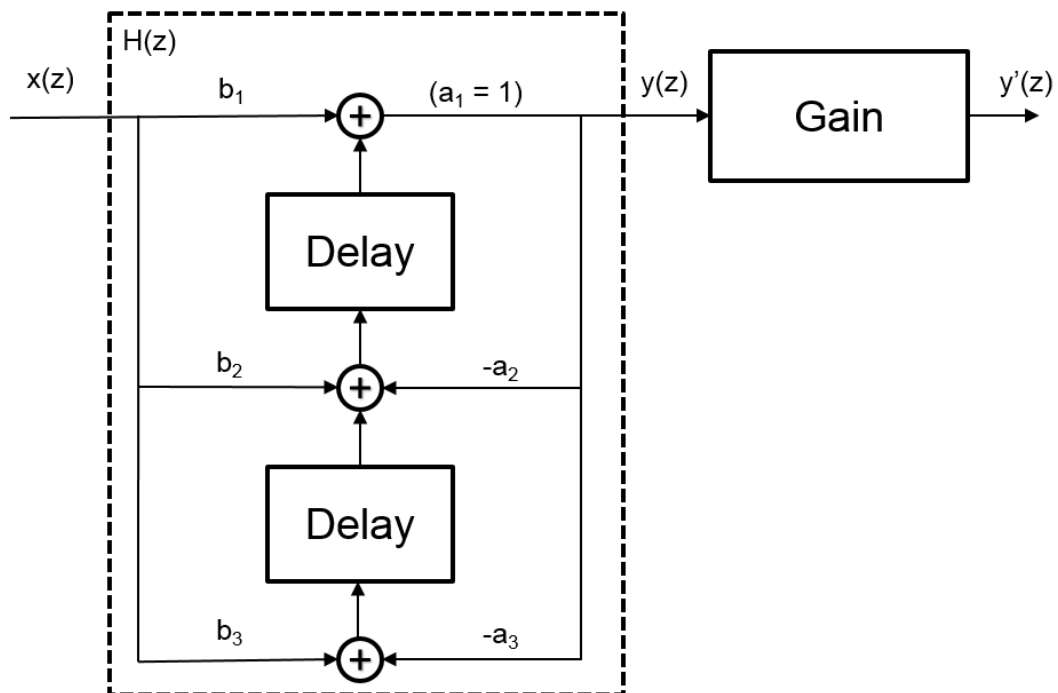
$$V^2 = x^2 + y^2 + z^2$$

Norm and norm squared data can be used in the decision trees in order to guarantee a high level of program customization for the user.

1.2 Filters

The input data seen in the previous section can be filtered by different kinds of filters available in the Machine Learning Core logic. The basic element of the Machine Learning Core filtering is a second order IIR filter, as shown in the following figure.

Figure 3. Filter basic element



The transfer function of the generic IIR 2nd order filter is the following:

$$H(z) = \frac{b_1 + b_2z^{-1} + b_3z^{-2}}{1 + a_2z^{-1} + a_3z^{-2}}$$

From Figure 3, the outputs can be defined as:

$$y(z) = H(z) \cdot x(z)$$

$$y'(z) = y(z) \cdot Gain$$

To optimize memory usage, the Machine Learning Core has default coefficients for the different kinds of filters (high-pass, band-pass, IIR1, IIR2). The Machine Learning Core tool will help in configuring the filter by asking for the filter coefficients needed after selecting the kind of filter. The following table shows the default values and the configurable values for the coefficients, depending on the filter type chosen. By setting different coefficients it is possible to tune the filter for the specific application.

Table 2. Filter coefficients

| Filter type / Coefficients | b_1 | b_2 | b_3 | a_2 | a_3 | Gain |
|----------------------------|--------------|--------------|--------------|--------------|--------------|--------------|
| High-pass filter | 0.5 | -0.5 | 0 | 0 | 0 | 1 |
| Band-pass filter | 1 | 0 | -1 | Configurable | Configurable | Configurable |
| IIR1 filter | Configurable | Configurable | 0 | Configurable | 0 | 1 |
| IIR2 filter | Configurable | Configurable | Configurable | Configurable | Configurable | 1 |

The filter coefficient values are expressed as half-precision floating-point format: S EEEEEFFFFFFFFFFF (S: 1 sign bit; E: 5 exponent bits; F: 10 fraction bits).

1.2.1 Filter coefficients

The IIR filter coefficients can be computed with Matlab by using the following function:

```
[b, a] = butter( N, f_cut / (ODR/2), 'low' )
```

Where:

- N is the order of the IIR filter (1 for IIR1, 2 for IIR2)
- f_cut is the cutoff frequency [Hz] of the filter
- ODR is the Machine Learning Core data rate [Hz]
- 'low' (or 'high') is the kind of filter to be implemented (low-pass or high-pass)

The following table shows some examples of filter coefficients (most of them considering an ODR of 26 Hz).

Table 3. Examples of filter coefficients

| Filter type / Coefficients | b_1 | b_2 | b_3 | a_2 | a_3 | Gain |
|---|-----------|-----------|-----------|------------|----------|------|
| High-pass IIR1, f_cut = 1 Hz, ODR = 26 Hz | 0.891725 | -0.891725 | - | -0.783450 | - | 1 |
| High-pass IIR1, f_cut = 2 Hz, ODR = 26 Hz | 0.802261 | -0.802261 | - | -0.604521 | - | 1 |
| High-pass IIR1, f_cut = 5 Hz, ODR = 26 Hz | 0.591628 | -0.591628 | - | -0.183257 | - | 1 |
| High-pass IIR1, f_cut = 10 Hz, ODR = 26 Hz | 0.274968 | -0.274968 | - | 0.450063 | - | 1 |
| High-pass IIR2, f_cut = 1 Hz, ODR = 26 Hz | 0.8428435 | -1.685687 | 0.8428435 | -1.6608344 | 0.710540 | 1 |
| High-pass IIR2, f_cut = 2 Hz, ODR = 26 Hz | 0.709560 | -1.419120 | 0.709560 | -1.332907 | 0.505334 | 1 |
| High-pass IIR2, f_cut = 5 Hz, ODR = 26 Hz | 0.4077295 | -0.815459 | 0.407730 | -0.426937 | 0.203981 | 1 |

| Filter type / Coefficients | b_1 | b_2 | b_3 | a_2 | a_3 | Gain |
|---|----------|-----------|-----------|-----------|----------|------|
| High-pass IIR2, $f_{cut} = 10$ Hz, ODR = 26 Hz | 0.085605 | -0.171209 | 0.085605 | 1.019146 | 0.361564 | 1 |
| Low-pass IIR1, $f_{cut} = 1$ Hz, ODR = 26 Hz | 0.108275 | 0.108275 | - | -0.783450 | - | 1 |
| Low-pass IIR1, $f_{cut} = 2$ Hz, ODR = 26 Hz | 0.197739 | 0.197739 | - | -0.604521 | - | 1 |
| Low-pass IIR1, $f_{cut} = 5$ Hz, ODR = 26 Hz | 0.408372 | 0.408372 | - | -0.183257 | - | 1 |
| Low-pass IIR1, $f_{cut} = 10$ Hz, ODR = 26 Hz | 0.725032 | 0.725032 | - | 0.450063 | - | 1 |
| Low-pass IIR2, $f_{cut} = 1$ Hz, ODR = 26 Hz | 0.012426 | 0.024853 | 0.012426 | -1.660834 | 0.710540 | 1 |
| Low-pass IIR2, $f_{cut} = 2$ Hz, ODR = 26 Hz | 0.043107 | 0.086213 | 0.043107 | -1.332907 | 0.505333 | 1 |
| Low-pass IIR2, $f_{cut} = 5$ Hz, ODR = 26 Hz | 0.194261 | 0.388522 | 0.194261 | -0.426937 | 0.203981 | 1 |
| Low-pass IIR2, $f_{cut} = 10$ Hz, ODR = 26 Hz | 0.595178 | 1.190355 | 0.595178 | 1.019146 | 0.361564 | 1 |
| Band-pass IIR2, $f_1 = 1.5$ Hz, $f_2 = 5$ Hz, ODR = 26 Hz | 0.310375 | 0 | -0.310375 | -1.069500 | 0.379250 | 1 |
| Band-pass IIR2, $f_1 = 0.2$ Hz, $f_2 = 1$ Hz, ODR = 100 Hz | 0.0236 | 0 | -0.0236 | -1.9521 | 0.9528 | 1 |

1.3 Features

The features are the statistical parameters computed from the Machine Learning Core inputs. The Machine Learning Core inputs which can be used for features computation are:

- the sensor input data which includes
 - sensor data from the X, Y, Z axes (e.g. Acc_X, Acc_Y, Acc_Z, Gyro_X, Gyro_Y, Gyro_Z);
 - external sensor data (e.g. ExtSens_X, ExtSens_Y, ExtSens_Z);
 - norm and norm squared signals of sensor / external sensor data (Acc_V, Acc_V2, Gyro_V, Gyro_V2, ExtSens_V, Ext_Sens_V2);
- the filtered data (e.g. high-pass on Acc_Z, band-pass on Acc_V2, etc...)

All the features are computed within a defined time window, which is also called “window length” since it is expressed as the number of samples. The size of the window has to be determined by the user and is very important for the machine learning processing, since all the statistical parameters in the decision tree will be evaluated in this time window.

The window length can have values from 1 to 255 samples. The choice of the window length value depends on the sensor data rate (ODR) and on the specific application or algorithm. In an activity recognition algorithm for instance, it can be decided to compute the features every 2 or 3 seconds, which means that considering sensors running at 26 Hz, the window length should be around 50 or 75 samples respectively.

Some of the features in the Machine Learning Core require some additional parameters for the evaluation (e.g. an additional threshold). The following table shows all the features available in the Machine Learning Core including additional parameters.

Table 4. Features

| Feature | Additional parameter |
|------------------------|----------------------|
| MEAN | - |
| VARIANCE | - |
| ENERGY | - |
| PEAK TO PEAK | - |
| ZERO CROSSING | Threshold |
| POSITIVE ZERO CROSSING | Threshold |
| NEGATIVE ZERO CROSSING | Threshold |
| PEAK DETECTOR | Threshold |
| POSITIVE PEAK DETECTOR | Threshold |
| NEGATIVE PEAK DETECTOR | Threshold |
| MINIMUM | - |
| MAXIMUM | - |

1.3.1 Mean

The feature “*Mean*” computes the average of the selected input (*I*) in the defined time window (*WL*) with the following formula:

$$Mean = \frac{1}{WL} \sum_{k=0}^{WL-1} I_k$$

1.3.2 Variance

The feature “*Variance*” computes the variance of the selected input (*I*) in the defined time window (*WL*) with the following formula:

$$Variance = \frac{\sum_{k=0}^{WL-1} I^2}{WL} - \left(\frac{\sum_{k=0}^{WL-1} I}{WL} \right)^2$$

1.3.3 Energy

The feature “*Energy*” computes the energy of the selected input (*I*) in the defined time window (*WL*) with the following formula:

$$Energy = \sum_{k=0}^{WL-1} I^2$$

1.3.4 Peak-to-peak

The feature “*Peak to peak*” computes the maximum peak-to-peak value of the selected input in the defined time window.

1.3.5 Zero-crossing

The feature “*Zero-crossing*” computes the number of times the selected input crosses a selected threshold in the defined time window.

1.3.6 Positive zero-crossing

The feature “*Positive zero-crossing*” computes the number of times the selected input crosses a selected threshold in the defined time window. Only the transitions with positive slopes are considered for this feature.

1.3.7 Negative zero-crossing

The feature “*Negative zero-crossing*” computes the number of times the selected input crosses a selected threshold in the defined time window. Only the transitions with negative slopes are considered for this feature.

1.3.8 Peak detector

The feature “*Peak detector*” counts the number of peaks (positive and negative) of the selected input in the defined time window.

A threshold has to be defined for this feature, and a buffer of three values is considered for the evaluation. If the second value of the three values buffer is higher (or lower) than the other two values of a selected threshold, the number of peaks is increased.

1.3.9 Positive peak detector

The feature “*Positive peak detector*” counts the number of positive peaks of the selected input in the defined time window.

A threshold has to be defined for this feature, and a buffer of three values is considered for the evaluation. If the second value of the three values buffer is higher than the other two values of a selected threshold, the number of peaks is increased.

1.3.10 Negative peak detector

The feature “*Negative peak detector*” counts the number of negative peaks of the selected input in the defined time window.

A threshold has to be defined for this feature, and a buffer of three values is considered for the evaluation. If the second value of the three values buffer is lower than the other two values of a selected threshold, the number of peaks is increased.

1.3.11 Minimum

The feature “*Minimum*” computes the minimum value of the selected input in the defined time window.

1.3.12 Maximum

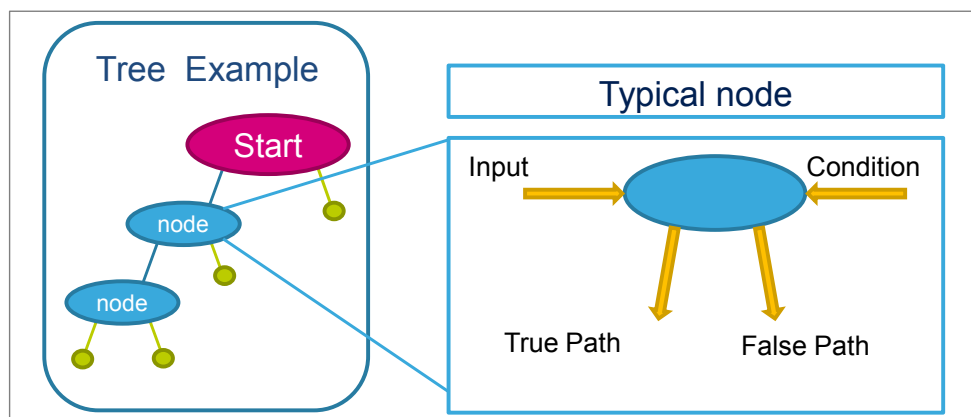
The feature “*Maximum*” computes the maximum value of the selected input in the defined time window.

1.4 Decision tree

The decision tree is the predictive model built from the training data which can be stored in the LSM6DSOX. The training data are the data logs acquired for each class to be recognized (in the activity recognition example the classes might be walking, jogging, driving, etc.).

The outputs of the computation blocks described in the previous sections are the inputs of the decision tree. Each node of the decision tree contains a condition, where a feature is evaluated with a certain threshold. If the condition is true, the next node in the true path is evaluated. If the condition is false, the next node in the false path is evaluated. The status of the decision tree will evolve node by node until a result is found. The result of the decision tree is one of the classes defined at the beginning of the data collection.

Figure 4. Decision tree node



The decision tree generates a result every sample. These results can also be filtered by an additional (optional) filter called Meta-Classifer, which is described in [Section 1.5](#).

The Machine Learning Core results (decision tree results filtered or not filtered) are accessible through dedicated registers in the embedded advanced features page 1 of the LSM6DSOX registers (as shown in [Table 5](#)). These registers can be continuously read (polled) to check the decision tree outputs. It is also possible to set an interrupt on the change in the decision tree result ([Table 6](#)). Furthermore, the interrupt signal generated can also be driven to the INT1 pad, so that an MCU performing other tasks, or sleeping (to save power), can be awakened when the Machine Learning Core result has changed.

Table 5. Decision tree results

| Register | Content |
|----------------|---------------------------|
| MLC0_SRC (70h) | Result of decision tree 1 |
| MLC1_SRC (71h) | Result of decision tree 2 |
| MLC2_SRC (72h) | Result of decision tree 3 |
| MLC3_SRC (73h) | Result of decision tree 4 |
| MLC4_SRC (74h) | Result of decision tree 5 |
| MLC5_SRC (75h) | Result of decision tree 6 |
| MLC6_SRC (76h) | Result of decision tree 7 |
| MLC7_SRC (77h) | Result of decision tree 8 |

Table 6. Decision tree interrupts

| Register | Content |
|---------------------------|--|
| MLC_STATUS_MAINPAGE (38h) | Contains interrupt status bits for changes in the decision tree result |
| MLC_STATUS (15h) | Contains interrupt status bits for changes in the decision tree result |
| MLC_INT1 (0Dh) | Allows routing of interrupt status bits for decision trees to INT1 pad |

1.4.1 Decision tree limitations in the LSM6DSOX

The LSM6DSOX has limited resources for the Machine Learning Core in terms of number of decision trees, size of the trees, and number of decision tree results.

Up to 8 different decision trees can be stored in the LSM6DSOX, but the sum of the number of nodes for all the decision trees must not exceed 256 (*). Every decision tree can have up to 16 results in the LSM6DSOX.

(*) This number might also be limited by the number of features and filters configured. In general, if using few filters and features, there is no further limitation on the size of the decision tree. However, when using many filters and features, the maximum number of nodes for the decision trees must be reduced, since some more memory will be allocated for the additional decision tree nodes. For instance, if the number of filters configured is 10 and the number of features configured is 50, the maximum number of nodes might be reduced by 100.

The table below summarizes the limitations of the LSM6DSOX.

Table 7. Decision tree limitations in the LSM6DSOX

| | LSM6DSOX |
|---|----------|
| Maximum number of decision trees | 8 |
| Maximum number of nodes (Total number for all the decision trees) | 256 (*) |
| Maximum number of results per decision tree | 16 |

1.5 Meta-classifier

A meta-classifier is a filter on the outputs of the decision tree. The meta-classifier uses some internal counters in order to filter the decision tree outputs.

Decision tree outputs can be divided in subgroups (e.g. similar classes can be managed in the same subgroup). An internal counter is available for all the subgroups of the decision tree outputs. The counter for the specific subgroup is increased when the result of the decision tree is one of the classes in the subgroup and it is decreased otherwise. When the counter reaches a defined value, which is called “end counter” (set by the user), the output of the Machine Learning Core is updated.

Table 8. Meta-classifier example

| Decision tree result | A | A | A | B | A | B | B | B | A | B | B | B | A | A | A |
|---|---|---|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Counter A (End counter = 3) | 1 | 2 | 3 | 2 | 3 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 3 |
| Counter B (End counter = 4) | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 3 | 2 | 3 | 4 | 5 | 4 | 3 | 2 |
| Machine Learning Core result (including meta-classifier) | x | x | A | A | A | A | A | A | A | A | B | B | B | B | A |

The previous table shows the effect of filtering the decision tree outputs through a meta-classifier. The first line of the table contains the outputs of the decision tree before the meta-classifier. Counter A and Counter B are the internal counters for the two decision tree results (“A” and “B”). In the activity recognition example, the result “A” might be walking and the result “B” jogging. When the internal counter “A” reaches the value 3 (which is the End Counter for counter “A”), there is a transition to result “A”. When the internal counter “B” reaches value 4, there is a transition to result “B”.

The purpose of the meta-classifier is to reduce the false positives, in order to avoid generating an output which is still not stable, and to reduce the transitions on the decision tree result.

1.5.1 Meta-classifier limitations in the LSM6DSOX

The meta-classifier has a limited number of sub-groups, 4 sub-groups can be used in LSM6DSOX. Similar classes may need grouped in the same subgroup to use the meta-classifier.

Table 9. Meta-classifier limitations in the LSM6DSOX

| | LSM6DSOX |
|---|----------|
| Maximum number of results per decision tree | 16 |
| Result sub-groups for meta-classifier per decision tree | 4 |

1.6 Finite State Machine interface

The LSM6DSOX also provides a configurable Finite State Machine which is suitable for deductive algorithms and in particular gesture recognition.

Finite state machines and decision trees can be combined to work together in order to enhance the accuracy of motion detection.

The decision tree results generated by the Machine Learning Core can be checked by the Finite State Machine available in the LSM6DSOX; this is possible through the command CHKDT available in the state machine

2 Machine Learning Core tools

The Machine Learning Core programmability in the device is allowed through a dedicated tool, available as an extension of the Unico GUI.

2.1 Unico GUI

Unico is the Graphical User Interface for all the MEMS sensor demonstration boards available in the STMicroelectronics portfolio. It has the possibility to interact with a motherboard based on the STM32 microcontroller (Professional MEMS Tool), which enables the communication between the MEMS sensor and the PC GUI.

The details of the Professional MEMS Tool board can be found at the following page:

https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mems-motion-sensor-eval-boards/steval-mki109v3.html

The Unico GUI is available in three software packages for the three operating systems supported.

- Windows
 - https://www.st.com/content/st_com/en/products/embedded-software/evaluation-tool-software/stsw-mki109w.html
- Linux
 - https://www.st.com/content/st_com/en/products/embedded-software/evaluation-tool-software/stsw-mki109l.html
- Mac OS X
 - https://www.st.com/content/st_com/en/products/embedded-software/evaluation-tool-software/stsw-mki109m.html

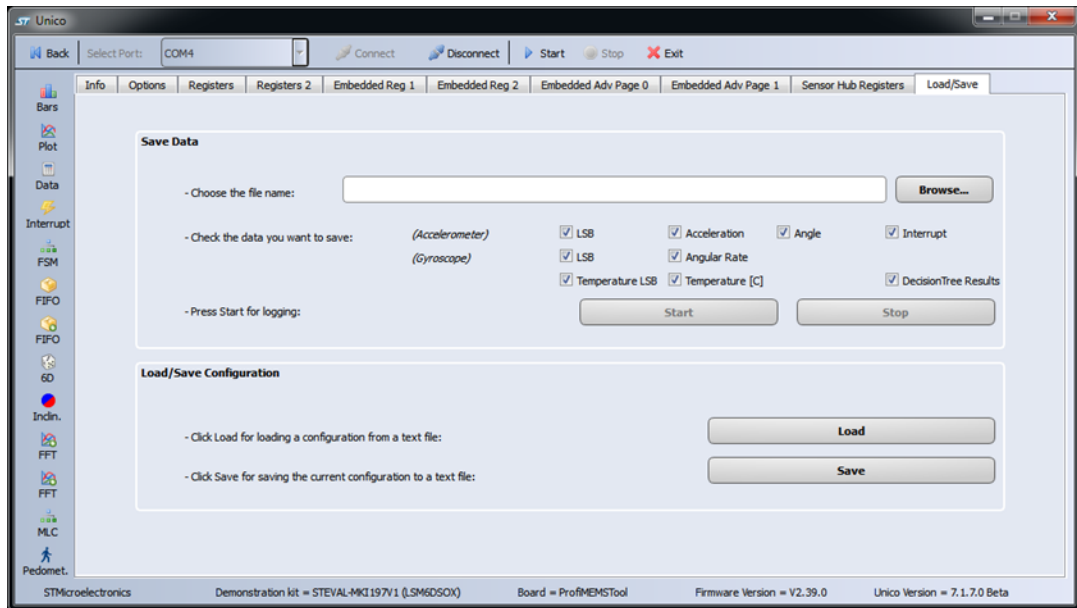
Unico GUI allows visualization of sensor outputs in both graphical and numerical format and allows the user to save or generally manage data coming from the device.

Unico allows access to the MEMS sensor registers, enabling fast prototyping of register setup and easy testing of the configuration directly on the device. It is possible to save the current register configuration in a text file and load a configuration from an existing file. In this way, the sensor can be re-programmed in few seconds.

The Machine Learning Core tool available in the Unico GUI abstracts the process of register configuration by automatically generating configuration files for the device. The user just needs to set some parameters in the GUI and by clicking a few buttons, the configuration file is already available. From these configuration files, the user can create his own library of configurations for the device.

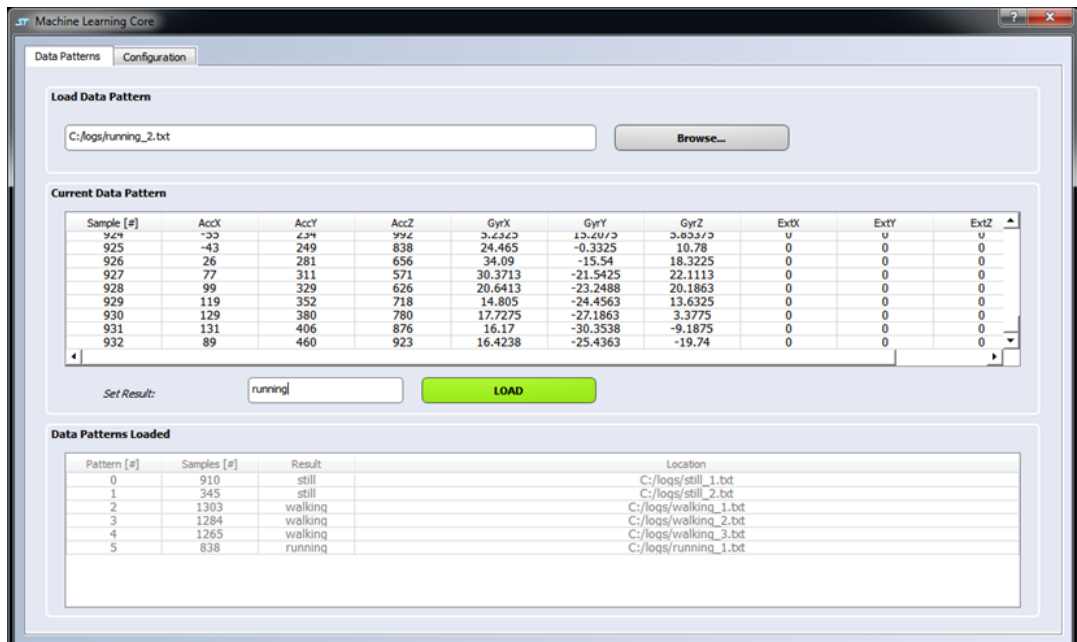
Since the machine learning approach requires the collection of data logs, they can be acquired through the load/save tab of Unico (Figure 5). For the accelerometer, the checkbox "Acceleration" allows saving data in [mg]. For the gyroscope, the checkbox "Angular rate" allows saving data in [dps].

Figure 5. Unico Load/Save tab



The data logs collected can then be used in the Machine Learning Core tool of Unico, available on the left side of the GUI in the Data Patterns tab of the Machine Learning Core (Figure 6), the data logs can be loaded. An expected result must be assigned to each data pattern loaded (for instance, in the the activity recognition algorithm, the results might be: still, walking, jogging, etc..).

Figure 6. Machine Learning Core tool - Data Patterns



The unit of measurement for the data expected in the data patterns tab of the Machine Learning Core tool are:

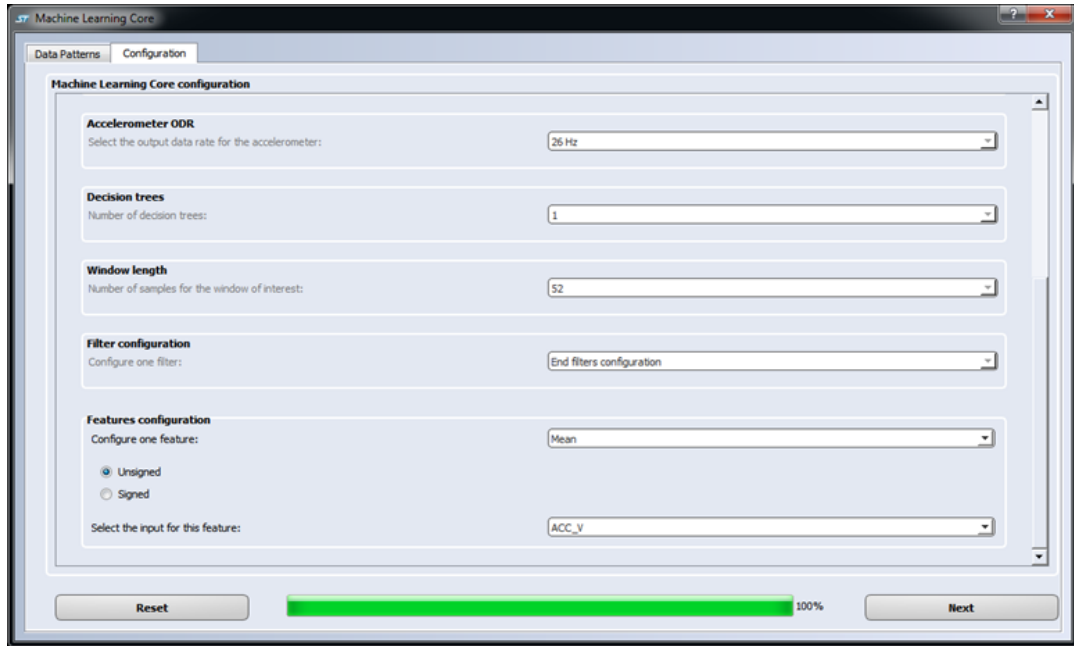
- [mg] for the accelerometer
- [dps] for the gyroscope

The conversion from [mg] to [g] for the accelerometer, and [dps] to [rad/s] for the gyroscope, is automatically managed internally by the Machine Learning Core tool, to allow the Machine Learning Core logic to work with the

correct data ([g] and [rad/s]). For the external sensor data, the user will be required at a later stage in the configuration to set the proper sensitivity.

In the “Configuration” tab of the Machine Learning Core tool (Figure 7), all the parameters of the Machine Learning Core (such as ODR, full scales, window length, filters, features, meta-classifier) can be configured. The tool allows selecting multiple filters which can be applied to the raw data, and multiple features to be computed from the input data or from the filtered data. The features computed will be the attributes of the decision tree.

Figure 7. Machine Learning Core tool - Configuration



The “Configuration” tab of the Machine Learning Core tool generates an Attribute-Relation File (ARFF), which is the starting point for the decision tree generation process. The decision tree can be generated by different machine learning tools (Section 2.2).

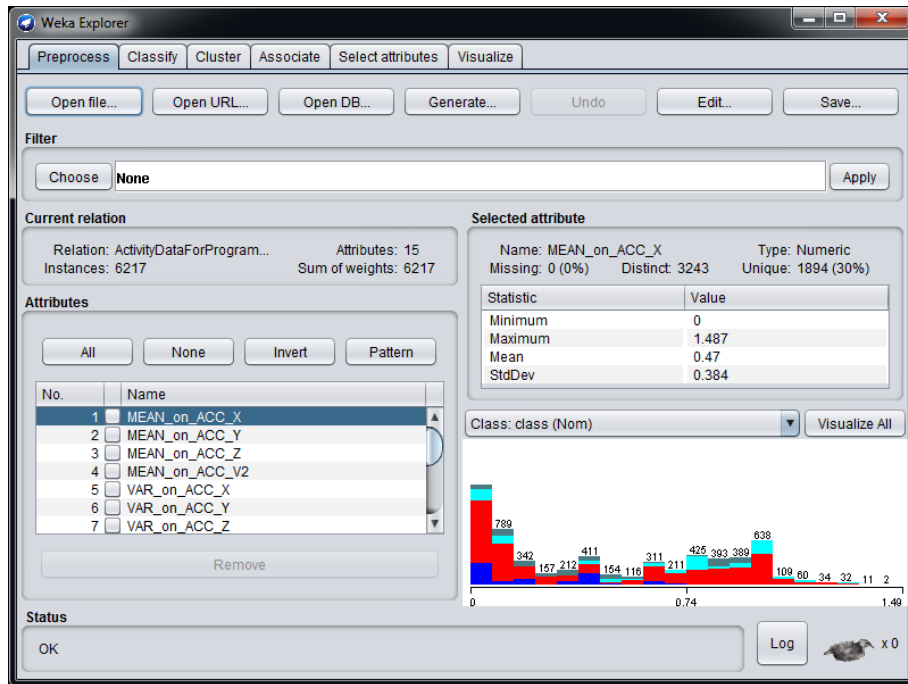
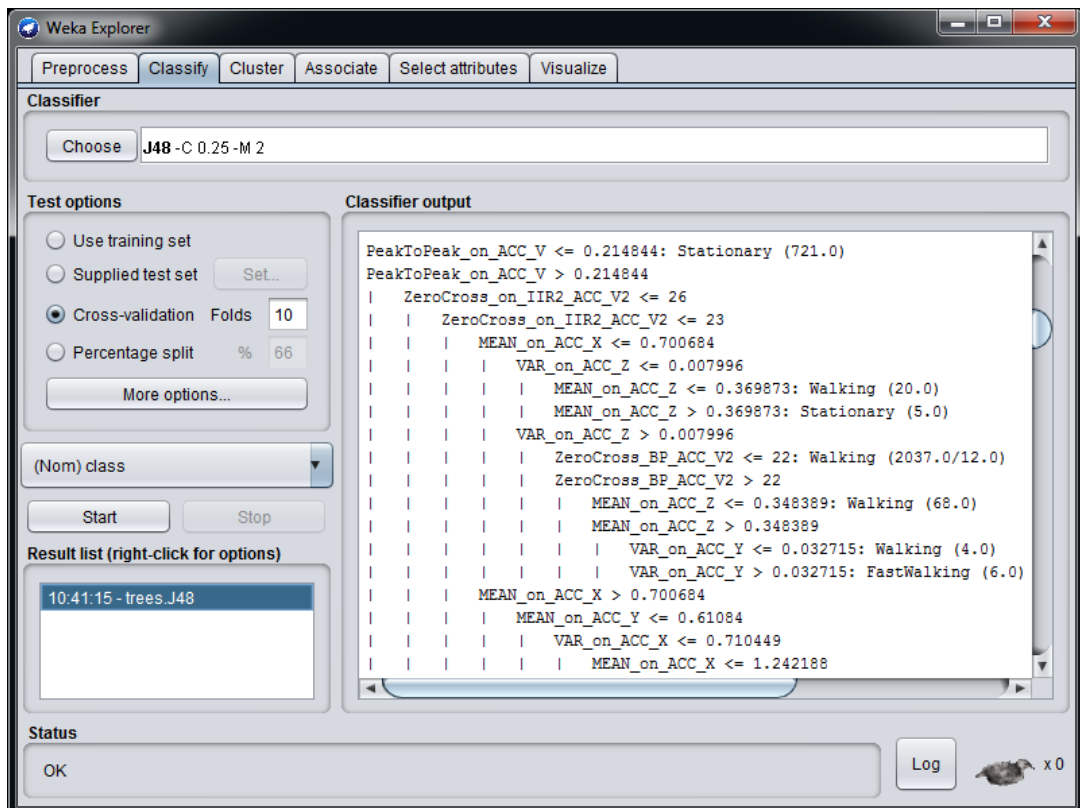
Once the decision tree has been generated, it can be uploaded to the Machine Learning Core tool in Unico to complete the generation of the register configuration for the LSM6DSOX.

The Unico GUI, by accessing the sensor registers, can read the status of the decision tree outputs, visualize them together with sensor data, and make it possible to log all the data (sensor outputs and decision tree outputs) together in the same text file.

2.2 Decision tree generation

Several machine learning tools are able to generate a decision tree. One of the most used tool is Weka, software developed by the University of Waikato, more details about this software can be found in Section A.1 WEKA.

Weka is able to generate a decision tree starting from an Attribute-Relation File (ARFF). Through Weka it is possible to evaluate which attributes are good for the decision tree, and different decision tree configurations can be implemented by changing all the parameters available in Weka. Figure 8 and Figure 9 show the “Preprocess” and “Classify” tabs of Weka which allow evaluating the attributes and generating the decision tree.

Figure 8. Weka preprocess

Figure 9. Weka classify


Once the decision tree has been generated, it can be uploaded to the Machine Learning Core tool in Unico, to complete the generation of the register configuration for the LSM6DSOX.

The Machine Learning Core tool in Unico accepts as input the decision tree files in a textual format (.txt). The textual file must contain the decision tree in the Weka J48 format (an example of a decision tree is shown in [Figure 10](#)). From the Weka classifier output ([Figure 9](#)), the decision tree has to be selected starting from the first line (first node) and including the last two rows (number of leaves and size of the tree). The selected output from Weka has to be copied to a text file.

Figure 10. Decision tree format

```

example_DecisionTree_format.txt - Notepad
File Edit Format View Help
|average_z <= 37.62
| |zerocross_y <= 0
| | |average_z <= -28.35
| | | |average_z <= -57.24: nowalk (11.0)
| | | |average_z > -57.24: walk (44.0/6.0)
| | |average_z > -28.35: nowalk (1398.0/341.0)
| | |zerocross_y > 0
| | | |average_z <= -60.28: nowalk (82.0/1.0)
| | | |average_z > -60.28: walk (11393.0/472.0)
|average_z > 37.62
| |average_z <= 43.4
| | |zerocross_y <= 5: nowalk (253.0/22.0)
| | |zerocross_y > 5
| | | |zerocross_y <= 11: walk (161.0/39.0)
| | | |zerocross_y > 11: nowalk (53.0/10.0)
| |average_z > 43.4: nowalk (5361.0/634.0)

Number of Leaves : 9
Size of the tree : 17
    
```

If the decision tree has been generated from a different tool, the format must be converted to the Weka J48 format in order to allow the Machine Learning Core tool in Unico to read the decision tree correctly.

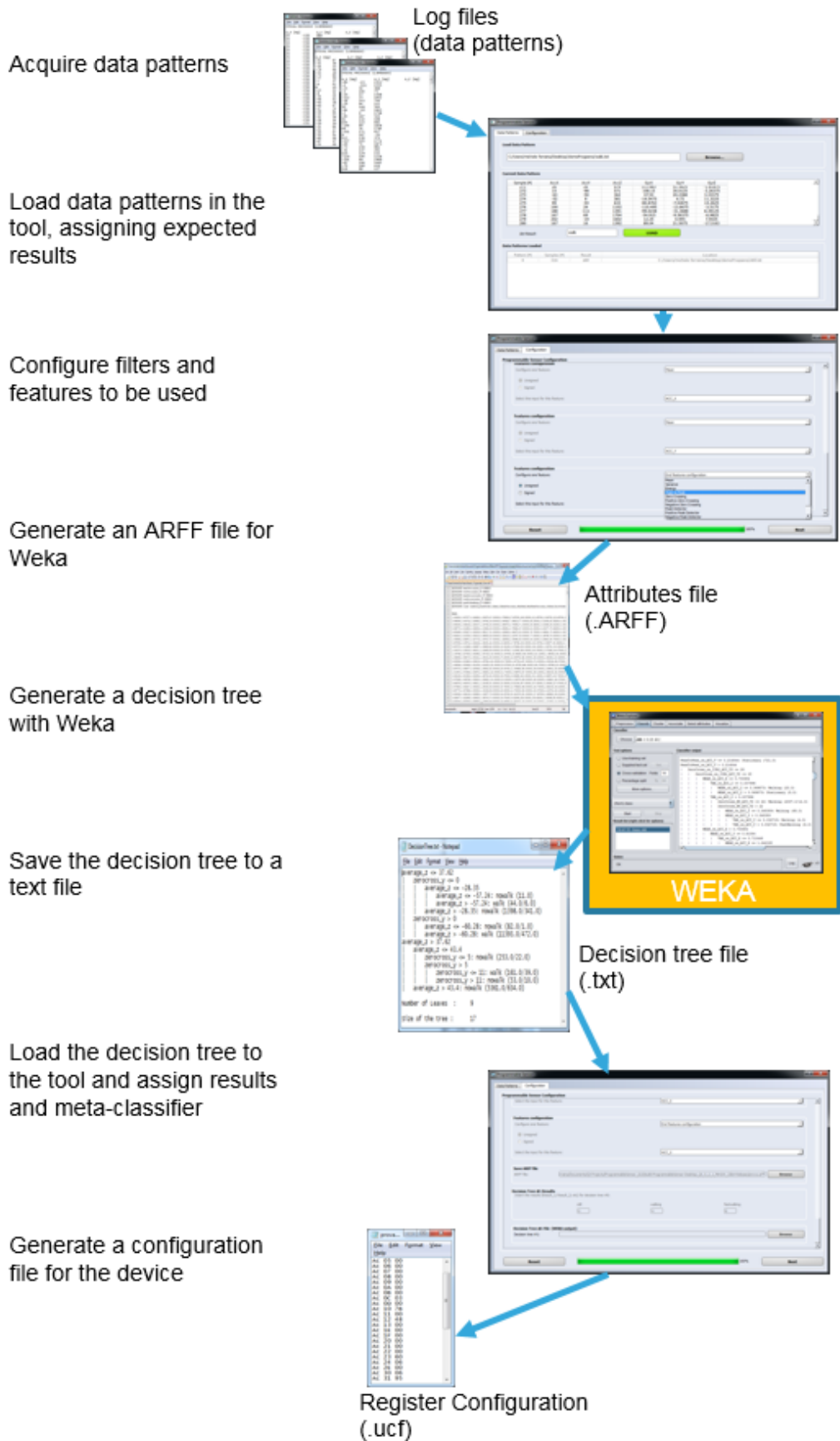
2.3 Configuration procedure

[Figure 11](#) shows the whole procedure of the machine learning processing, from the data patterns to the generation of a register setting for the device (LSM6DSOX).

As seen in [Section 2.1](#) the data patterns can be acquired in the “Load/Save” tab of the Unico GUI. If this is not possible or if the user wants to use some different data patterns, they can still be uploaded in the Machine Learning Core tool of Unico, with a few limitations:

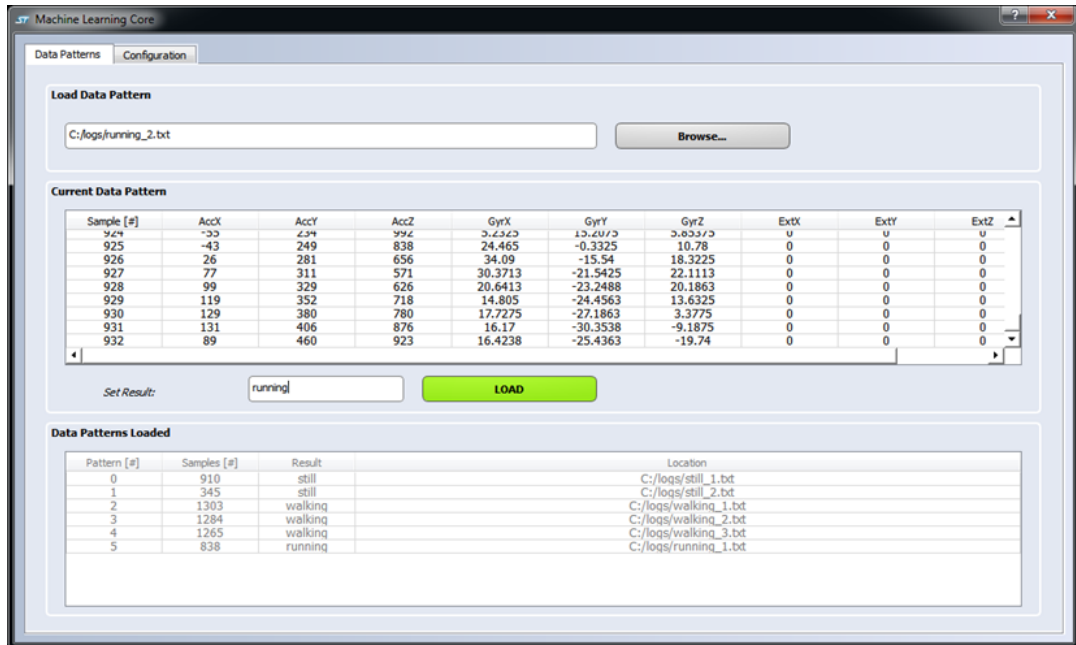
- Every data patterns has to start with a header line, containing the unit of measurement of the data
 - A_X [mg] A_Y [mg] A_Z [mg] G_X [dps] G_Y [dps] G_Z [dps]
- The data after the header line must be separated by “tab” or “space”.
- The order of sensors in the file columns must be accelerometer data (if available), gyroscope data (if available), external sensor data (if available).
- The order of the axes in the columns of any sensor is X, Y, Z.

Figure 11. Configuration procedure



Opening the Machine Learning Core tool available in Unico, the data patterns, acquired in the format described above, can be loaded assigning the expected result for each data log (as shown in the following figure).

Figure 12. Assigning a result to a data pattern



When all the data patterns have been loaded, the Machine Learning Core parameters can be configured through the configuration tab. These parameters are ODR, full scales, number of decision trees, window length, filters, features, etc... (as shown in Figure 13, Figure 14, Figure 15, Figure 16).

Figure 13. Configuration of Machine Learning Core

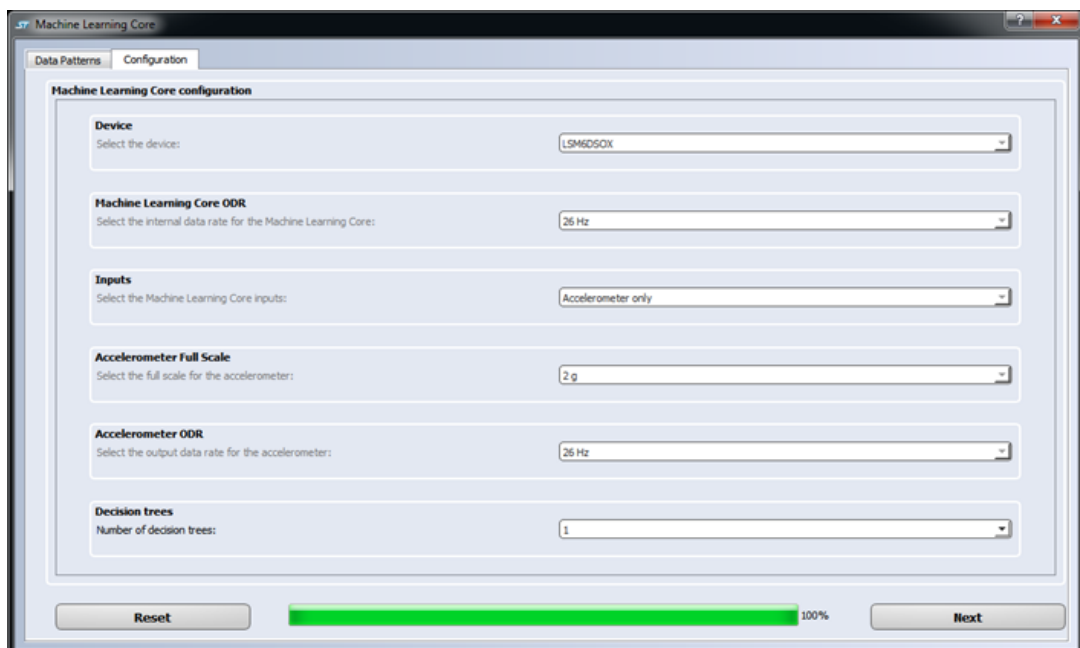


Figure 14. Configuration of filters

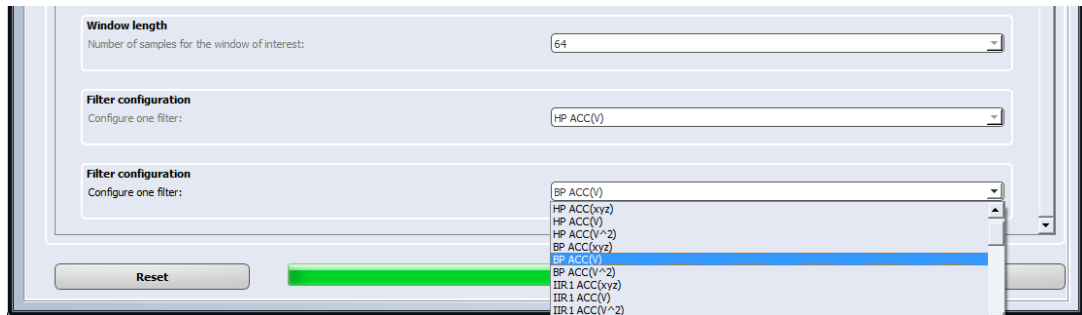


Figure 15. Configuration of features

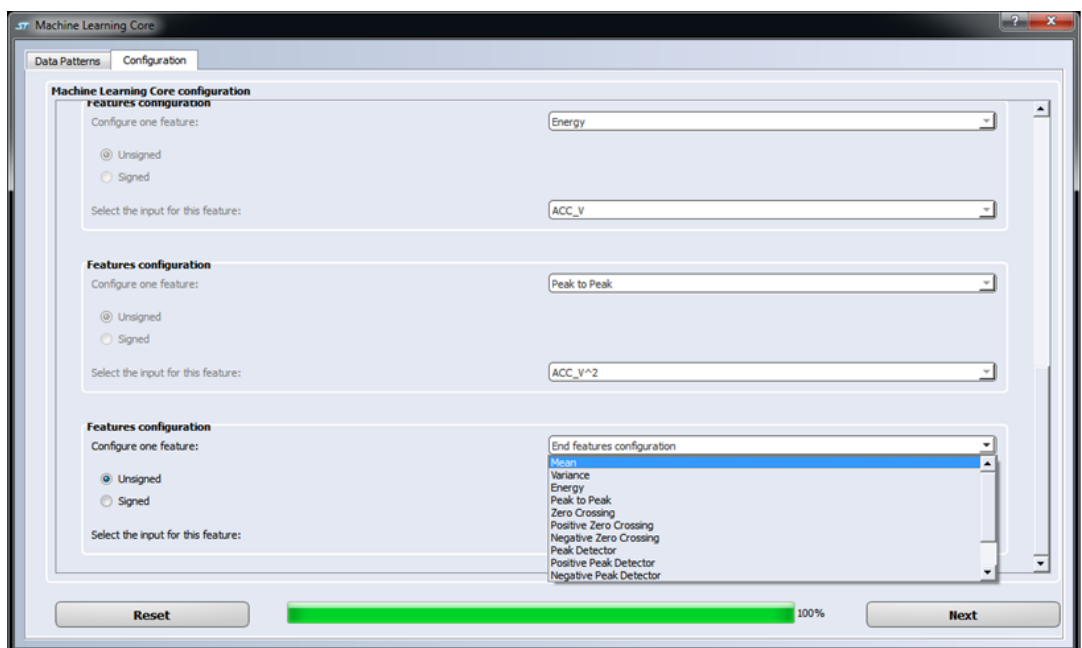
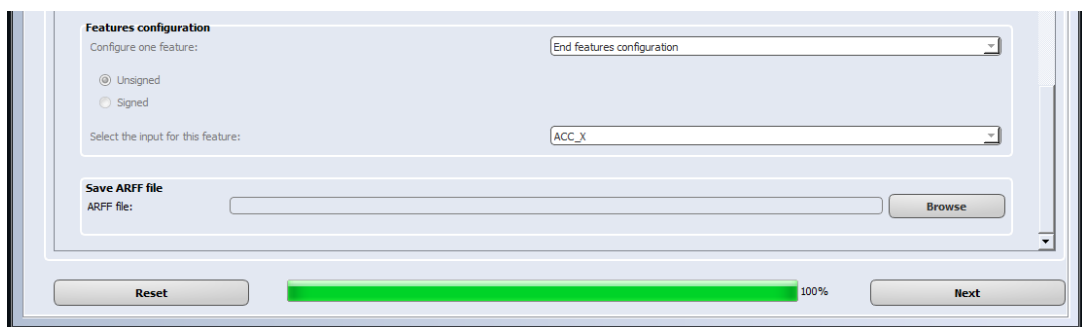


Figure 16. ARFF generation

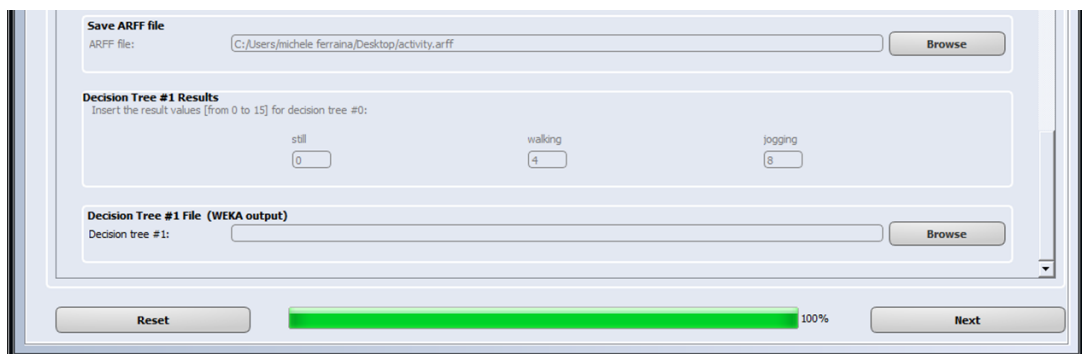


Multiple filters and multiple features can be chosen. The GUI will iteratively ask for another filter (or another feature) until the parameter “End filter configuration” (or “End features configuration”) is chosen (Figure 16). Once all the features have been configured, the Machine Learning Core tool in Unico will generate an ARFF file, which is the file containing all the features computed from the training data. Figure 17 shows an example of an ARFF file generated by the Machine Learning Core tool in Unico. The ARFF file generated can be loaded into Weka to build a decision tree. If the decision tree is not going to be built with Weka, the user must adapt the ARFF file to the file format required by any other tool for decision tree

generation. In this particular case, the decision tree format may also be needed to be adapted to the Weka J48 format described in [Section 2.2](#).

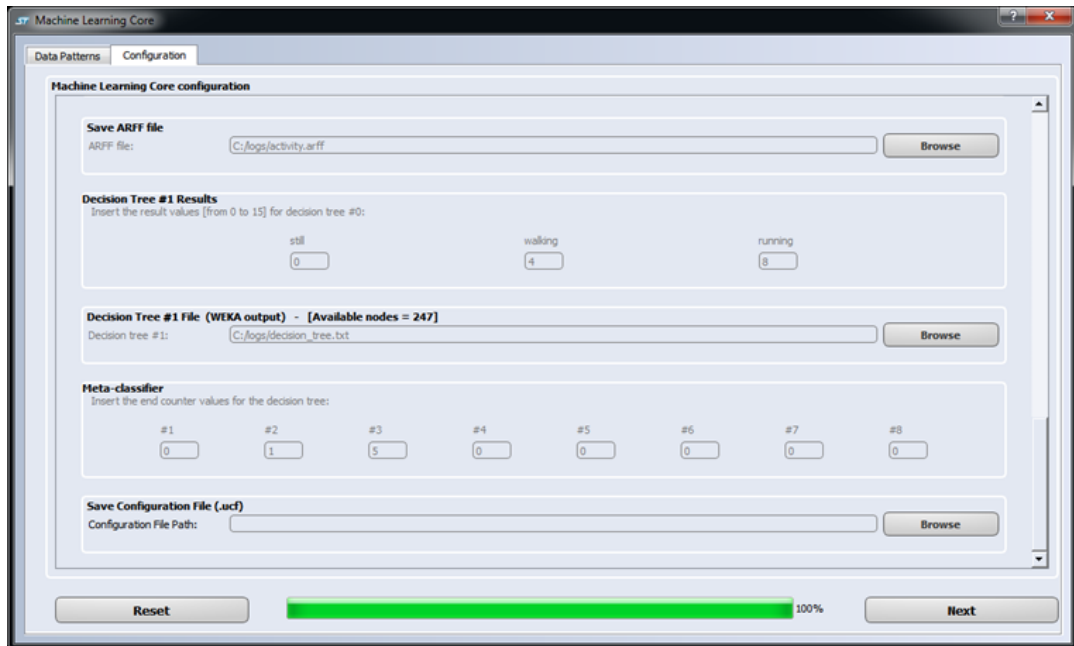
Figure 17. ARFF file

After configuring the result values for the decision tree, the decision tree (in the Weka J48 format) can be loaded in the Machine Learning Core tool of the Unico GUI in order to complete the configuration and get the register settings for the device (LSM6DSOX).

Figure 18. Configuration of results and decision tree


The last step of the configuration process is to configure the meta classifier, which is the optional filter for the generation of the decision tree results. After that, the tool is ready to generate a configuration for the device ([Figure 18](#)).

Figure 19. Meta classifier and device configuration



When the register configuration for the device has been saved, it can be loaded in the device using the Load/Save tab of the Unico GUI.

Figure 20. Unico load configuration



When the device is programmed, the Machine Learning Core results can be monitored in the Data window of Unico (Figure 21) or in one of the registers tabs containing the Machine Learning Core source registers (Figure 22).

Figure 21. Unico Data window

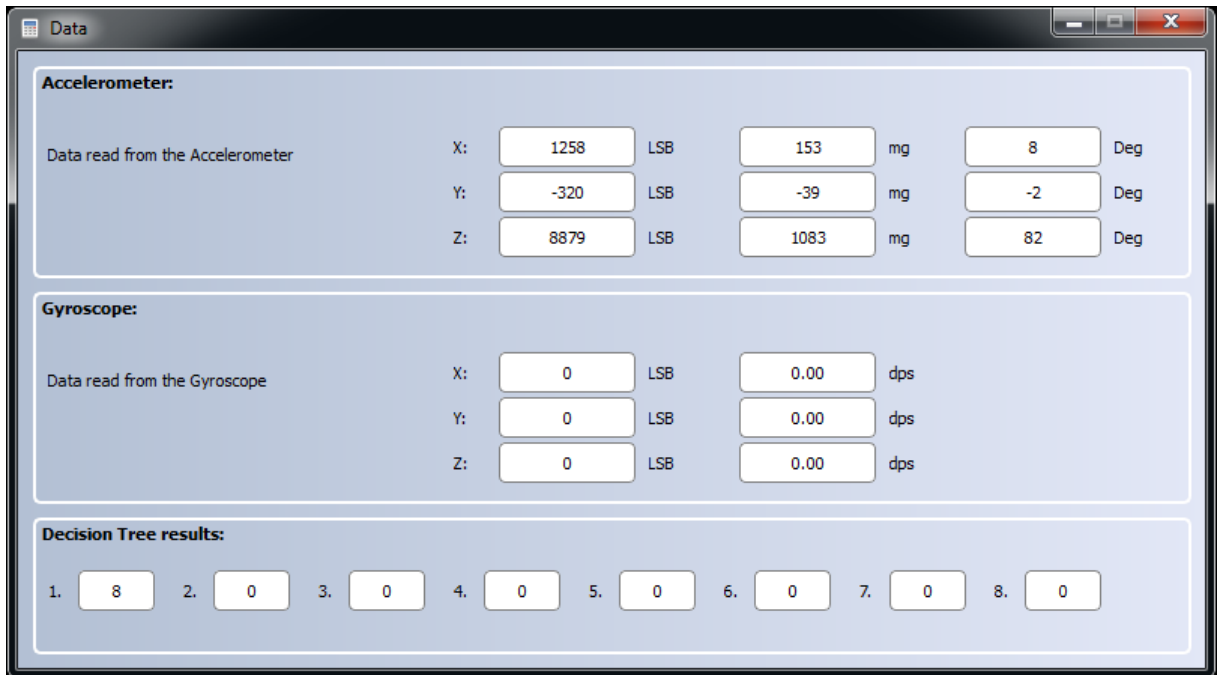


Figure 22. Unico - Machine Learning Core source registers

| | | | | | |
|----------|-------|----|------|-------|---------|
| MLC0_SRC | (70h) | 01 | Read | Write | Default |
| MLC1_SRC | (71h) | 00 | Read | Write | Default |
| MLC2_SRC | (72h) | 00 | Read | Write | Default |
| MLC3_SRC | (73h) | 00 | Read | Write | Default |
| MLC4_SRC | (74h) | 00 | Read | Write | Default |
| MLC5_SRC | (75h) | 00 | Read | Write | Default |
| MLC6_SRC | (76h) | 00 | Read | Write | Default |
| MLC7_SRC | (77h) | 00 | Read | Write | Default |

3 Decision tree examples

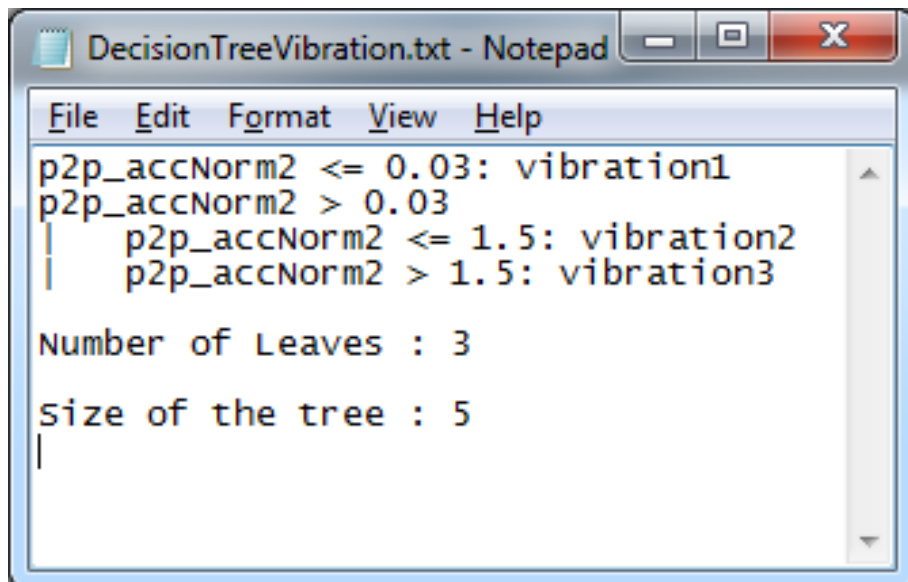
This section describes some examples of a decision tree which can be loaded in the LSM6DSOX.

3.1 Vibration monitoring

The decision tree in the following figure shows a simple example of vibration monitoring. Three different levels of vibrations are recognized (vibration1, vibration2, vibration3) using a simple decision tree with just one feature, the peak-to-peak feature in the accelerometer norm squared input (p2p_accNorm2).

The vibration monitoring example runs at 26 Hz, computing features in a window of 16 samples. The current consumption of the LSM6DSOX is around 162 μA at 1.8 V. Turning off the Machine Learning Core, the current consumption of the LSM6DSOX would be around 161 μA , so just 1 μA is the additional current consumption of the Machine Learning Core.

Figure 23. Vibration monitoring decision tree



```
DecisionTreeVibration.txt - Notepad
File Edit Format View Help
p2p_accNorm2 <= 0.03: vibration1
p2p_accNorm2 > 0.03
|   p2p_accNorm2 <= 1.5: vibration2
|   p2p_accNorm2 > 1.5: vibration3

Number of Leaves : 3
Size of the tree : 5
|
```

3.2 Motion intensity

The decision tree in the following figure shows a simple example of motion intensity implemented using just the feature “variance” in the accelerometer norm. Eight different intensity levels are recognized by this decision tree. The configuration for motion intensity described in this example runs at 12.5 Hz, computing features in a window of 39 samples. The current consumption of the LSM6DSOX is around 162 μA at 1.8 V. Turning off the programmable sensor, the current consumption of the LSM6DSOX would be around 161 μA , so just 1 μA is the additional current consumption of the Machine Learning Core.

Figure 24. Motion intensity decision tree

```

dec_tree.txt - Notepad
File Edit Format View Help
module_variance <= 0.009: Intensity_0
module_variance > 0.009
| module_variance <= 0.013671875: Intensity_1
| module_variance > 0.013671875
| | module_variance <= 0.0234375: Intensity_2
| | module_variance > 0.0234375
| | | module_variance <= 0.033203125: Intensity_3
| | | module_variance > 0.033203125
| | | | module_variance <= 0.078125: Intensity_4
| | | | module_variance > 0.078125
| | | | | module_variance <= 0.1640625: Intensity_5
| | | | | module_variance > 0.1640625
| | | | | | module_variance <= 0.3125: Intensity_6
| | | | | | module_variance > 0.3125: Intensity_7
Number of Leaves : 8
Size of the tree : 15
    
```

3.3 6D position recognition

The LSM6DSOX already has a 6D position recognition algorithm embedded in the device. The example described in this section shows just a different implementation using a decision tree.

The six different positions (Figure 25) can be easily recognized by a simple decision tree (Figure 26) using the following features:

- *meanx_abs*: Mean of the accelerometer X axis (unsigned)
- *meany_abs*: Mean of the accelerometer Y axis (unsigned)
- *meanz_abs*: Mean of the accelerometer Z axis (unsigned)
- *meanx_s*: Mean of the accelerometer X axis (signed)
- *meany_s*: Mean of the accelerometer Y axis (signed)
- *meanz_s*: Mean of the accelerometer Z axis (signed)

The configuration for 6D position recognition described in this example runs at 26 Hz, computing features in a window of 16 samples. The current consumption of the LSM6DSOX is around 163 μA at 1.8 V. Turning off the Machine Learning Core, the current consumption of the LSM6DSOX would be around 161 μA , so just 2 μA is the additional current consumption of the Machine Learning Core.

Figure 25. 6D positions

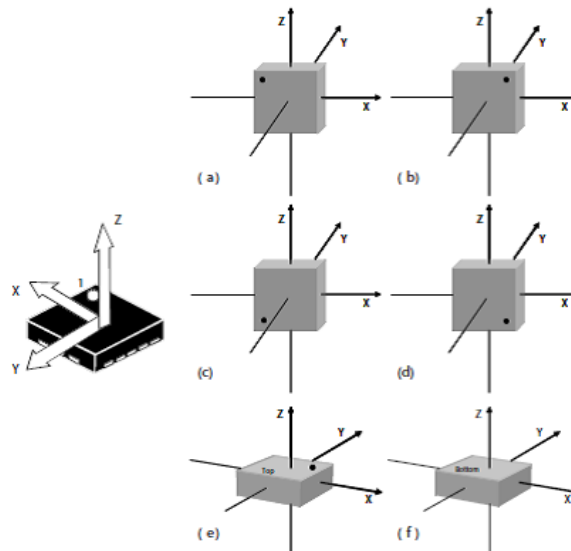


Figure 26. 6D decision tree

```

six_d.txt - Notepad
File Edit Format View Help
meanx_abs <= 0.3
  meany_abs <= 0.3
    meanz_s <= 0.3: zdw
    meanz_s > 0.3: zup
  meany_abs > 0.3
    meanz_abs <= 0.3
      meany_s <= 0.3: ydw
      meany_s > 0.3: yup
    meanz_abs > 0.3 : others
meanx_abs > 0.3
  meanz_abs <= 0.3
    meany_abs <= 0.3
      meanx_s <= 0.3 : xdw
      meanx_s > 0.3: xup
    meany_abs > 0.3 : others
  meanz_abs > 0.3: others

Number of Leaves :    9
Size of the tree :    17
  
```

3.4 Activity recognition for smartphone applications

The activity recognition algorithm described in this example is intended for smartphone applications, since all the data logs collected for this purpose have been acquired with a smartphone carried in the user pocket. Hundreds of data logs have been acquired from different people, since different people walk or run in different ways which increases the complexity of the algorithm.

A small subset of all the possible activities has been selected in order to improve the accuracy of the recognition algorithm. The subset of activities recognized in this example are: Stationary, Walking, Jogging and Biking.

Four features have been used (mean, variance, peak-to-peak, zero-crossing), and two different filters have been applied to the accelerometer input data. The following table shows the activity recognition configuration.

Table 10. Activity recognition for smartphone configuration

| Configuration | Accelerometer, 26 Hz ODR, 4 g full scale |
|-----------------|--|
| Window length | 75 samples (around 3 seconds) |
| Filters | Band-pass on Accelerometer Norm |
| | IIR2 on Accelerometer Norm Squared |
| Features | Mean |
| | Variance |
| | Peak-to-peak |
| | Zero-crossing |
| Outputs | Stationary (0) |
| | Walking (1) |
| | Jogging (4) |
| | Biking (8) |
| Meta-classifier | 0 for Stationary and Walking |
| | 1 for Jogging |
| | 4 for Biking |

Figure 27 shows the decision tree generated by Weka. The cross-validation results of Weka (Figure 28) show that 96.7% of the instances have been classified in the correct way.

The configuration for the activity recognition example runs at 26 Hz, computing features in a window of 75 samples. The current consumption of the LSM6DSOX is around 165 μ A at 1.8 V. Turning off the Machine Learning Core, the current consumption of the LSM6DSOX would be around 161 μ A, so just 4 μ A is the additional current consumption of the Machine Learning Core.

Figure 27. Activity recognition for smartphone decision tree

```

C:\Users\nichele ferraina\Documents\ProgrammableSensor\Configurations_MPL31852\ActivityRecognition_forMobile\v0.7_good\dectree.txt - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
dectree.txt
1 ABS (VAR) _on_ACC_V~2 <= 0.032227
2 ABS (VAR) _on_ACC_V~2 <= 0.012696
3 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.023254: Stationary (6920.0/17.0)
4 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 > 0.023254
5 ABS (MEAN) _on_ACC_V~2 <= 0.944824
6 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 <= 64.8125: Biking (26.0)
7 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 > 64.8125: Walking (8.0)
8 ABS (MEAN) _on_ACC_V~2 > 0.944824
9 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 <= 11: Stationary (2393.0/103.0)
10 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 > 11
11 ABS (MEAN) _on_ACC_V~2 <= 0.999023: Walking (13.0)
12 ABS (MEAN) _on_ACC_V~2 > 0.999023
13 ABS (VAR) _on_ACC_V~2 <= 0.01123: Stationary (27.0/1.0)
14 ABS (VAR) _on_ACC_V~2 > 0.01123: Biking (6.0)
15 ABS (VAR) _on_ACC_V~2 > 0.012696
16 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 <= 75.125
17 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 <= 8
18 ABS (MEAN) _on_ACC_V~2 <= 0.95752: Walking (25.0/2.0)
19 ABS (MEAN) _on_ACC_V~2 > 0.95752
20 ABS (PeakToPeak) _on_ACC_V <= 0.371094
21 ABS (VAR) _on_ACC_V~2 <= 0.016113
22 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 <= 7: Stationary (8.0/1.0)
23 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 > 7: Biking (6.0)
24 ABS (VAR) _on_ACC_V~2 > 0.016113: Biking (32.0)
25 ABS (PeakToPeak) _on_ACC_V > 0.371094
26 ABS (VAR) _on_ACC_V~2 <= 0.025391: Stationary (33.0/1.0)
27 ABS (VAR) _on_ACC_V~2 > 0.025391
28 ABS (MEAN) _on_ACC_V~2 <= 0.977539: Stationary (7.0/3.0)
29 ABS (MEAN) _on_ACC_V~2 > 0.977539: Biking (16.0)
30 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 <= 8
31 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.427734: Walking (152.0/5.0)
32 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 > 0.427734
33 ABS (ENERGY) _on_ACC_V~2 <= 74: Biking (22.0/1.0)
34 ABS (ENERGY) _on_ACC_V~2 > 74
35 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.525391: Walking (9.0)
36 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 > 0.525391: Stationary (7.0/3.0)
37 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 > 75.125
38 ABS (MEAN) _on_ACC_V~2 <= 1.13867
39 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.072449: Stationary (61.0)
40 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 > 0.072449
41 ABS (VAR) _on_ACC_V~2 <= 0.021973
42 ABS (MEAN) _on_ACC_V~2 <= 0.993164: Biking (13.0/1.0)
43 ABS (MEAN) _on_ACC_V~2 > 0.993164
44 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 <= 90.625
45 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 <= 5
46 ABS (VAR) _on_ACC_V~2 <= 0.018556
47 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.079529: Biking (5.0)
48 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 > 0.079529
49 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.235962: Stationary (20.0)
50 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 > 0.235962: Biking (7.0/3.0)
51 ABS (VAR) _on_ACC_V~2 > 0.018556: Biking (11.0/1.0)
52 ABS (ZeroCross) _on_filter_IIR2_on_ACC_V~2 > 5
53 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 <= 77.5625
54 ABS (MEAN) _on_ACC_V~2 <= 0.998047: Stationary (6.0)
55 ABS (MEAN) _on_ACC_V~2 > 0.998047: Biking (10.0/1.0)
56 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 > 77.5625: Stationary (132.0/23.0)
57 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 > 90.625
58 ABS (PeakToPeak) _on_ACC_V <= 0.329102
59 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 <= 92.75: Biking (25.0)
60 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 > 92.75
61 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.290527
62 ABS (PeakToPeak) _on_ACC_V <= 0.319824: Stationary (18.0)
63 ABS (PeakToPeak) _on_ACC_V > 0.319824: Biking (5.0)
64 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 > 0.290527: Biking (20.0/2.0)
65 ABS (PeakToPeak) _on_ACC_V > 0.329102: Stationary (12.0)
66 ABS (VAR) _on_ACC_V~2 > 0.021973
67 ABS (PeakToPeak) _on_ACC_V <= 0.352539: Biking (54.0/6.0)
68 ABS (PeakToPeak) _on_ACC_V > 0.352539
69 ABS (MEAN) _on_ACC_V~2 <= 1.06348
70 ABS (ENERGY) _on_ACC_V~2 <= 84.9375
71 ABS (ENERGY) _on_filter_IIR2_on_ACC_V~2 <= 84.125
72 ABS (MEAN) _on_ACC_V~2 <= 1.02539
73 ABS (ENERGY) _on_filter_BP_on_ACC_V~2 <= 0.184937: Stationary (7.0)
Normal text file length:19,828 lines:256 Ln:1 Col:1 Sel:0|0 Windows (CR LF) UTF-8 INS
    
```

Figure 28. Weka cross-validation

```

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      30331          96.7311 %
Incorrectly Classified Instances    1025           3.2689 %
Kappa statistic                    0.9421
Mean absolute error                0.0296
Root mean squared error            0.1202
Relative absolute error            10.4519 %
Root relative squared error        31.9379 %
Total Number of Instances          31356

=== Detailed Accuracy By Class ===
          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
          -----  -----  -
          0.975   0.011   0.976     0.975   0.976     0.965   0.997    0.992    Stationary
          0.989   0.028   0.979     0.989   0.984     0.962   0.990    0.986    Walking
          0.966   0.000   0.995     0.966   0.980     0.980   0.989    0.968    Jogging
          0.769   0.014   0.815     0.769   0.791     0.775   0.944    0.763    Biking
Weighted Avg.   0.967   0.021   0.967     0.967   0.967     0.950   0.989    0.971

=== Confusion Matrix ===
          a      b      c      d  <-- classified as
9738   33      0   212 |  a = Stationary
  12 17665      6   187 |  b = Walking
   0   36  1151      4 |  c = Jogging
  225   310      0  1777 |  d = Biking

```

3.5 Gym activity recognition

Gym activity recognition is intended as a fitness example for a wearable device, like a smartwatch or a wristband. To implement this algorithm with a decision tree, all the data logs have been acquired using the device (LSM6DSOX) mounted on a wristband.

The inputs of two sensors have been used (accelerometer and gyroscope at 104 Hz data rate) and six different features computed in a window of 208 samples (mean, variance, peak-to-peak, min, max, zero-crossing), as shown in Table 11.

The decision tree in Figure 29 generated by Weka allows recognizing five different gym activities including bicep curls, jumping jacks, lateral raises, push-ups, squats.

The configuration for the gym activity recognition described in this example runs at 104 Hz, computing features in a window of 208 samples. The current consumption of the LSM6DSOX is around 569 μA at 1.8 V. Turning off the Machine Learning Core, the current consumption of the LSM6DSOX (with accelerometer and gyroscope at 104 Hz) would be around 556 μA , so 13 μA is the additional current consumption of the Machine Learning Core for this algorithm.

Table 11. Configuration for gym activity recognition

| Configuration | Accelerometer, 104 Hz ODR, 4 g full scale |
|---------------|--|
| | Gyroscope, 104 Hz ODR, 2000 dps full scale |
| Window length | 208 samples (around 2 seconds) |
| Features | Mean |
| | Variance |
| | Peak-to-peak |
| | Min |
| | Max |
| | Zero-crossing |

| Configuration | Accelerometer, 104 Hz ODR, 4 g full scale |
|-----------------|---|
| Outputs | No activity (0) |
| | Bicep curls (4) |
| | Jumping jacks (5) |
| | Lateral raises (6) |
| | Push-ups (7) |
| | Squats (8) |
| Meta-classifier | 0 for No activity |
| | 2 for all the other outputs |

Figure 29. Gym activity recognition decision tree

```

1 ABS (VAR) _on_ACC_V^2 <= 0.032227
2 | ABS (VAR) _on_ACC_V^2 <= 0.012695
3 | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.023254: Stationary (6920.0/17.0)
4 | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 > 0.023254
5 | | | | ABS (MEAN) _on_ACC_V^2 <= 0.944824
6 | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 <= 64.8125: Biking (26.0)
7 | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 > 64.8125: Walking (8.0)
8 | | | | | ABS (MEAN) _on_ACC_V^2 > 0.944824
9 | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 <= 11: Stationary (2393.0/103.0)
10 | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 > 11
11 | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 0.999023: Walking (13.0)
12 | | | | | | | ABS (MEAN) _on_ACC_V^2 > 0.999023
13 | | | | | | | | ABS (VAR) _on_ACC_V^2 <= 0.01123: Stationary (27.0/1.0)
14 | | | | | | | | ABS (VAR) _on_ACC_V^2 > 0.01123: Biking (6.0)
15 | | | | | | | | | ABS (VAR) _on_ACC_V^2 > 0.012695
16 | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 <= 75.125
17 | | | | | | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 <= 8
18 | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 0.95752: Walking (25.0/2.0)
19 | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 > 0.95752
20 | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V <= 0.371094
21 | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 <= 0.016113
22 | | | | | | | | | | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 <= 7: Stationary (8.0/1.0)
23 | | | | | | | | | | | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 > 7: Biking (6.0)
24 | | | | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 > 0.016113: Biking (32.0)
25 | | | | | | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V > 0.371094
26 | | | | | | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 <= 0.025391: Stationary (33.0/1.0)
27 | | | | | | | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 <= 0.025391
28 | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 0.977539: Stationary (7.0/3.0)
29 | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 > 0.977539: Biking (16.0)
30 | | | | | | | | | | | | | | | | | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 > 8
31 | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.427734: Walking (152.0/5.0)
32 | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 > 0.427734
33 | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_ACC_V^2 <= 74: Biking (22.0/1.0)
34 | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_ACC_V^2 > 74
35 | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.525391: Walking (9.0)
36 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 > 0.525391: Stationary (7.0/3.0)
37 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 > 75.125
38 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 1.13867
39 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.072449: Stationary (61.0)
40 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 > 0.072449
41 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 <= 0.021973
42 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 0.993164: Biking (13.0/1.0)
43 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 > 0.993164
44 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 <= 90.625
45 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 <= 5
46 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 <= 0.018555
47 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.079529: Biking (5.0)
48 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 > 0.079529
49 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.235962: Stationary (20.0)
50 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 > 0.235962: Biking (7.0/3.0)
51 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 > 0.018555: Biking (11.0/1.0)
52 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ZeroCross) _on_filter_IIR2_on_ACC_V^2 > 5
53 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 <= 77.5625
54 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 0.998047: Stationary (6.0)
55 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 > 0.998047: Biking (10.0/1.0)
56 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 > 77.5625: Stationary (132.0/23.0)
57 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 > 90.625
58 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V <= 0.329102
59 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 <= 92.75: Biking (25.0)
60 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 > 92.75
61 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.290527
62 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V <= 0.319824: Stationary (18.0)
63 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V > 0.319824: Biking (5.0)
64 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 > 0.290527: Biking (20.0/2.0)
65 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V > 0.329102: Stationary (12.0)
66 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (VAR) _on_ACC_V^2 > 0.021973
67 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V <= 0.352539: Biking (54.0/6.0)
68 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (PeakToPeak) _on_ACC_V > 0.352539
69 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 1.06348
70 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_ACC_V^2 <= 84.9375
71 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_IIR2_on_ACC_V^2 <= 84.125
72 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (MEAN) _on_ACC_V^2 <= 1.02539
73 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ABS (ENERGY) _on_filter_BP_on_ACC_V^2 <= 0.184937: Stationary (7.0)
    
```

A Appendix

A.1 WEKA

Weka is free software developed at the University of Waikato, New Zealand. It contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions.

All of Weka's techniques are predicated on the assumption that the data is available as one flat file or relation, where each data point is described by a fixed number of attributes.

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. The ARFF files have two distinct sections, as shown in Figure 30: a header section containing the attributes (features, classes), and a data section containing all the feature values together with the corresponding class to be associated to that set of features.

Figure 30. ARFF example

Figure 31. Weka GUI Chooser



When launching Weka, the Weka GUI Chooser window appears (Figure 31), and the “Explorer” section, selectable through the first button, is the Weka main user interface.

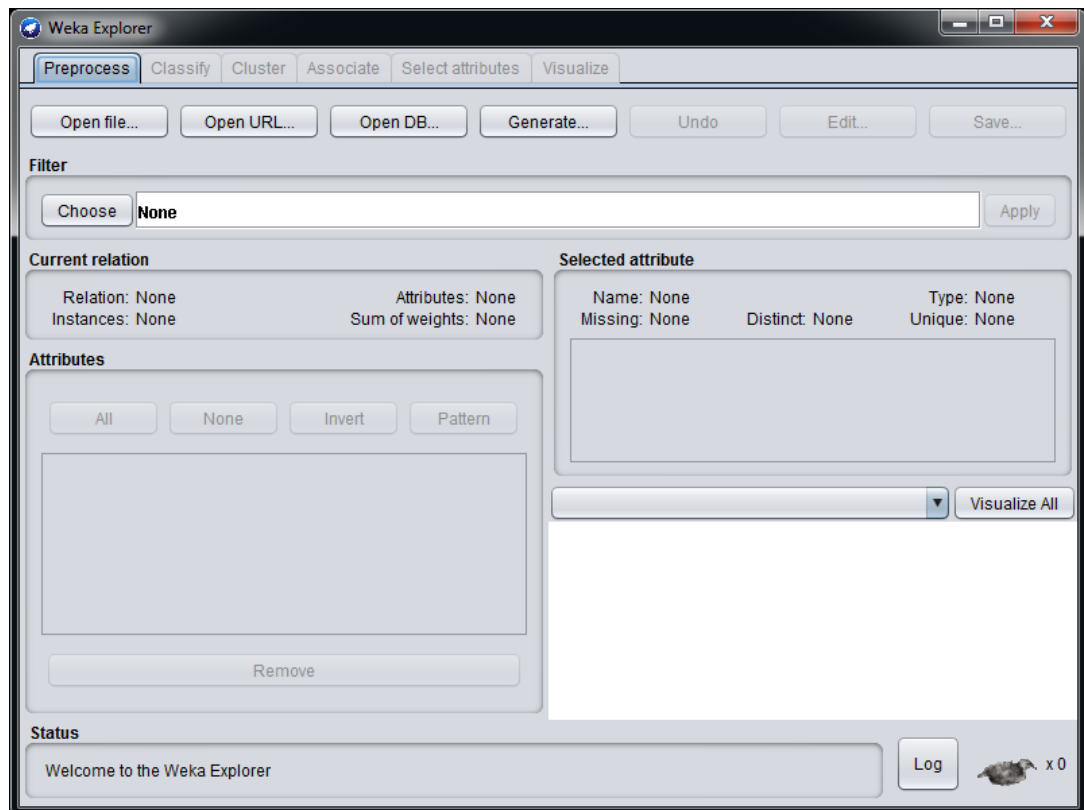
When selecting the Weka Explorer a new interface appears (Figure 32). Several panels are available in the Explorer interface:

- The *Preprocess* panel has facilities for importing data.
- The *Classify* panel allows applying classification and regression algorithms to the dataset in order to estimate accuracy of the resulting predictive model and to visualize erroneous predictions.
- The *Associate* panel provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data.
- The *Cluster* panel gives access to the clustering techniques in Weka.
- The *Select attributes* panel provides algorithms for identifying the most predictive attributes in a dataset.
- The *Visualize* panel shows a scatter plot matrix.

In this appendix section, only the *Preprocess* and *Classify* panels are described.

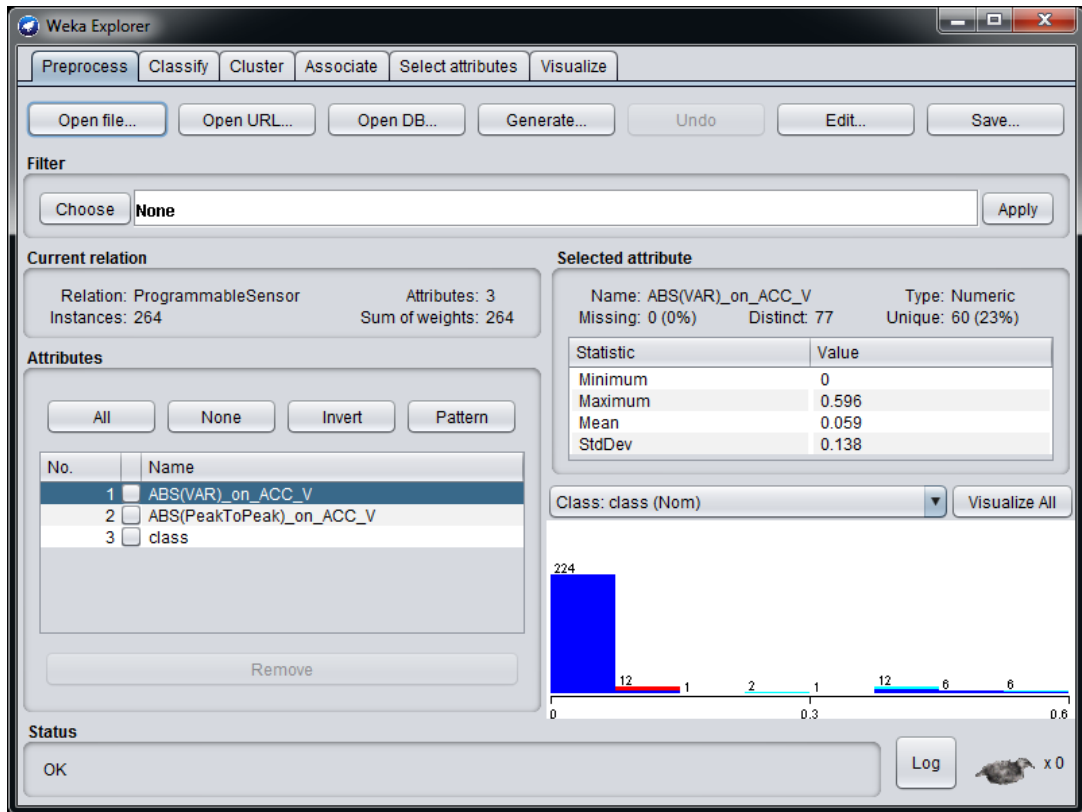
The Preprocess panel is shown in Figure 32, it allows loading an ARFF file from the “Open file” button.

Figure 32. Weka Explorer



When the ARFF file has been loaded, the preprocess panel shows all the attributes (features and classes) of the imported ARFF file. The attributes can be visualized in a graphical way and the user can select the attributes to be used for the classification.

Figure 33. Weka Explorer - Attributes



After choosing the attributes, a classifier can be configured in the *Classify* panel of Weka Explorer (Figure 34). There are many classifiers available in Weka, choosing the classifier J48 (under trees), a decision tree can be generated (Figure 35. Weka Classify J48).

Figure 34. Weka Classify

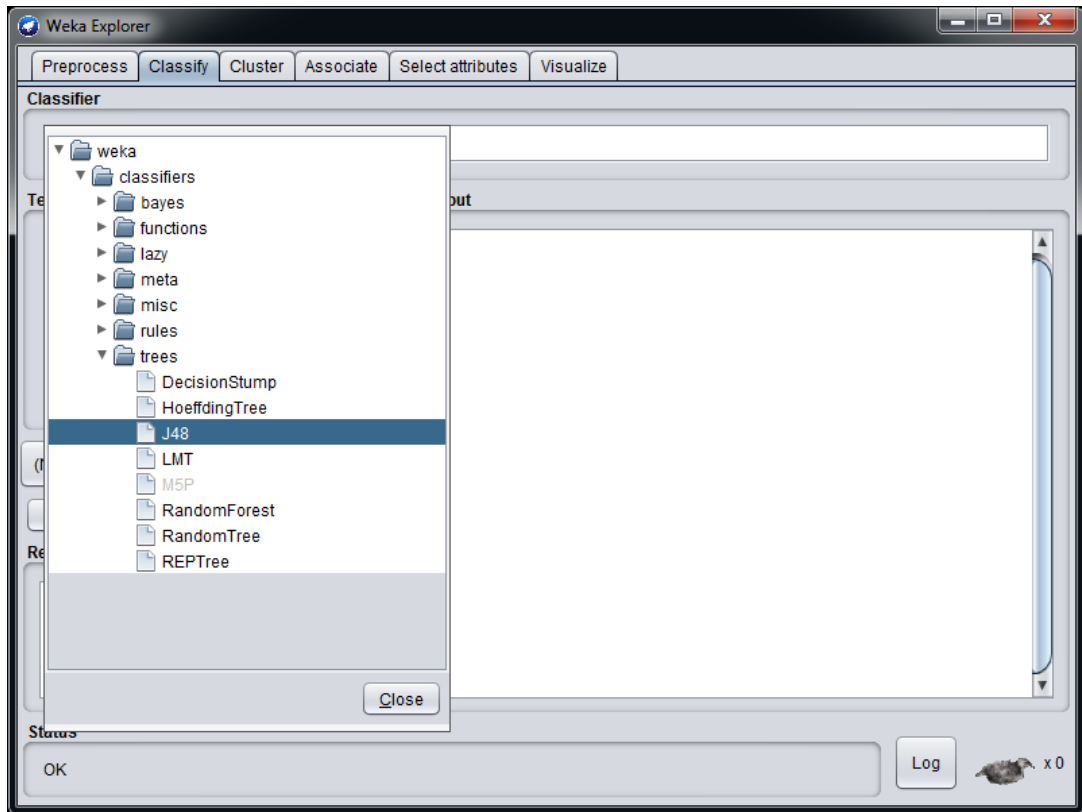
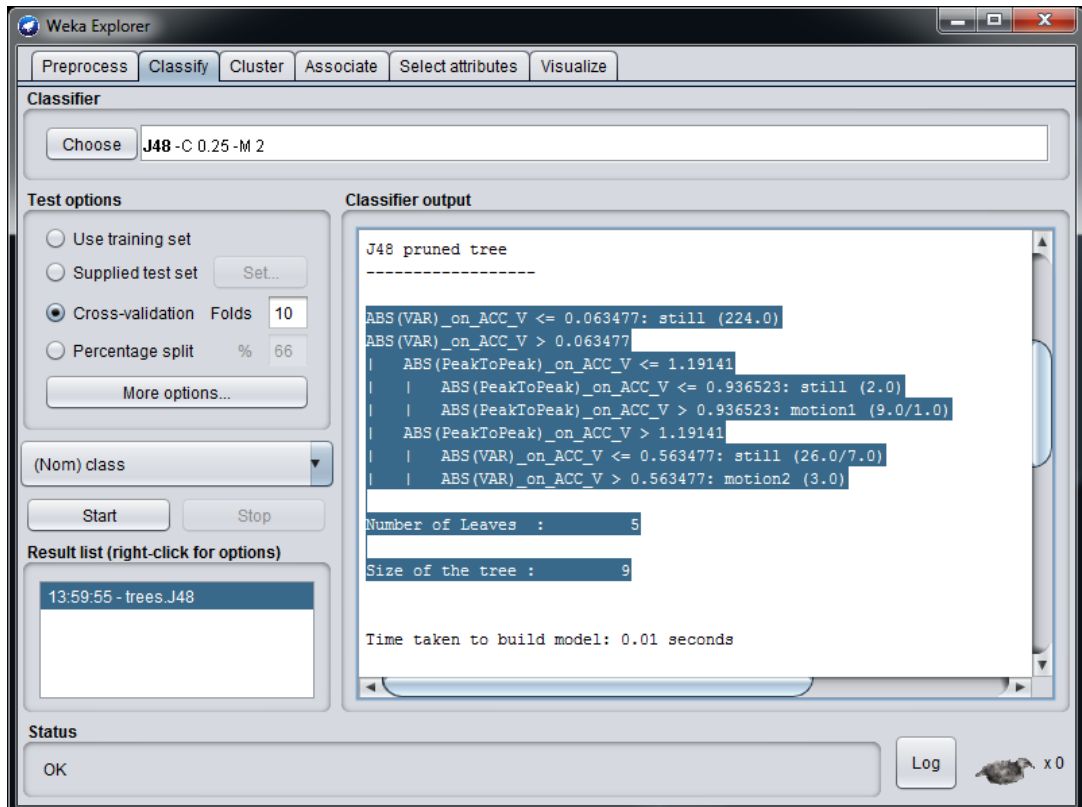


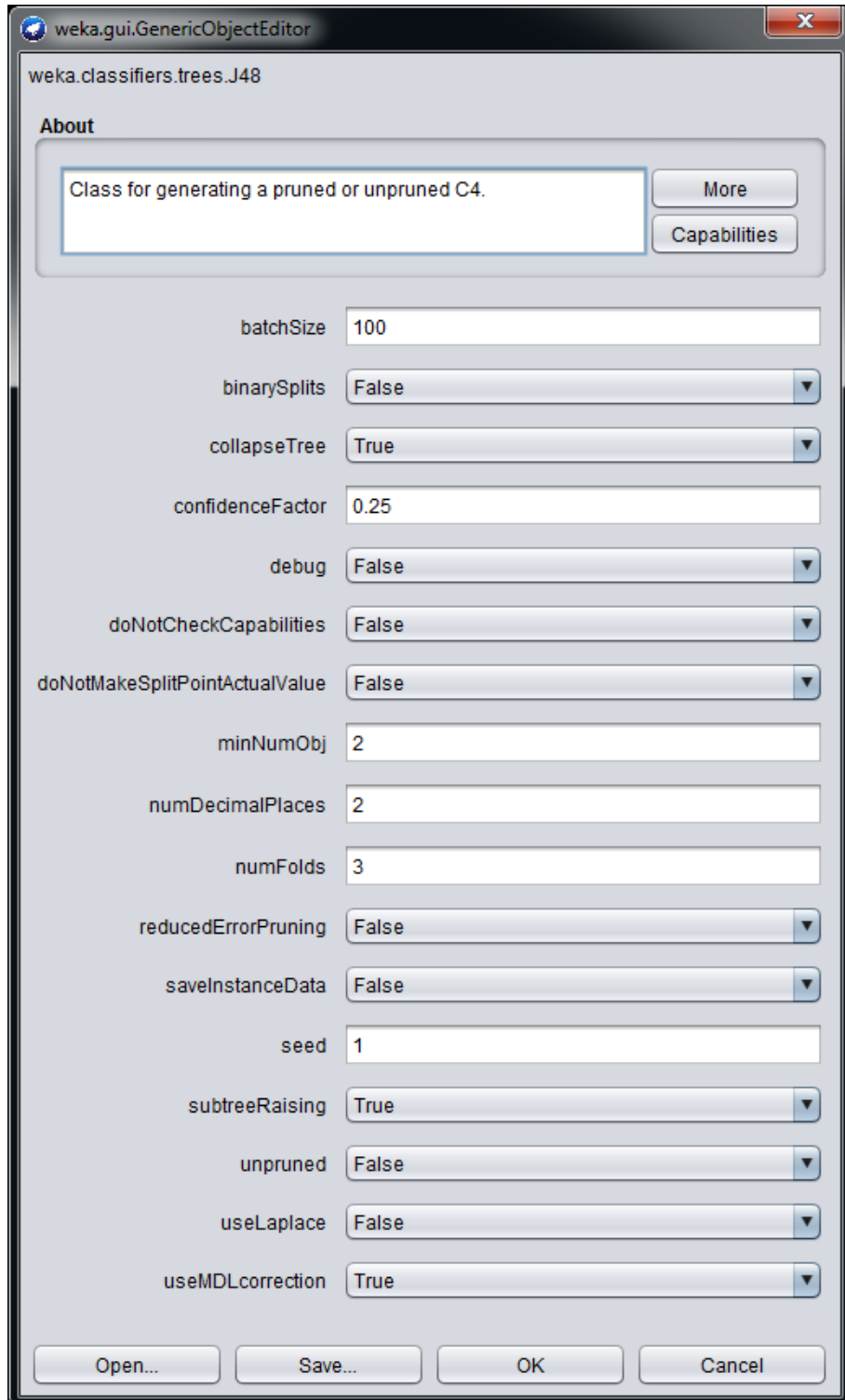
Figure 35. Weka Classify J48



Many parameters can be changed in the classifier section (Figure 36), and different decision trees can be generated by clicking the “Start” button.

All the decision trees generated can be compared in terms of a correctly classified instance and confusion matrix (parameters calculated for every decision tree generated).

Figure 36. Weka J48 classifier parameters



Revision history

Table 12. Document revision history

| Date | Version | Changes |
|-------------|---------|-----------------|
| 28-Jan-2019 | 1 | Initial release |

Contents

| | | |
|----------|--|-----------|
| 1 | Machine Learning Core in the LSM6DSOX | 2 |
| 1.1 | Inputs | 4 |
| 1.2 | Filters | 5 |
| 1.2.1 | Filter coefficients | 6 |
| 1.3 | Features | 7 |
| 1.3.1 | Mean | 8 |
| 1.3.2 | Variance | 8 |
| 1.3.3 | Energy | 8 |
| 1.3.4 | Peak-to-peak | 8 |
| 1.3.5 | Zero-crossing | 8 |
| 1.3.6 | Positive zero-crossing | 8 |
| 1.3.7 | Negative zero-crossing | 8 |
| 1.3.8 | Peak detector | 8 |
| 1.3.9 | Positive peak detector | 9 |
| 1.3.10 | Negative peak detector | 9 |
| 1.3.11 | Minimum | 9 |
| 1.3.12 | Maximum | 9 |
| 1.4 | Decision tree | 9 |
| 1.4.1 | Decision tree limitations in the LSM6DSOX | 10 |
| 1.5 | Meta-classifier | 11 |
| 1.5.1 | Meta-classifier limitations in the LSM6DSOX | 11 |
| 1.6 | Finite State Machine interface | 11 |
| 2 | Machine Learning Core tools | 12 |
| 2.1 | Unico GUI | 12 |
| 2.2 | Decision tree generation | 14 |
| 2.3 | Configuration procedure | 16 |
| 3 | Decision tree examples | 23 |
| 3.1 | Vibration monitoring | 23 |
| 3.2 | Motion intensity | 24 |
| 3.3 | 6D position recognition | 24 |

| | | |
|----------|--|-----------|
| 3.4 | Activity recognition for smartphone applications | 26 |
| 3.5 | Gym activity recognition | 28 |
| A | Appendix | 31 |
| A.1 | WEKA | 31 |
| | Revision history | 36 |
| | Contents | 37 |
| | List of tables | 39 |
| | List of figures | 40 |

List of tables

| | | |
|------------------|--|----|
| Table 1. | Machine Learning Core output data rates | 2 |
| Table 2. | Filter coefficients | 6 |
| Table 3. | Examples of filter coefficients | 6 |
| Table 4. | Features | 8 |
| Table 5. | Decision tree results. | 10 |
| Table 6. | Decision tree interrupts. | 10 |
| Table 7. | Decision tree limitations in the LSM6DSOX | 10 |
| Table 8. | Meta-classifier example | 11 |
| Table 9. | Meta-classifier limitations in the LSM6DSOX | 11 |
| Table 10. | Activity recognition for smartphone configuration | 26 |
| Table 11. | Configuration for gym activity recognition | 28 |
| Table 12. | Document revision history | 36 |

List of figures

| | | |
|------------|---|----|
| Figure 1. | Machine Learning Core in the LSM6DSOX | 2 |
| Figure 2. | Machine Learning Core blocks | 3 |
| Figure 3. | Filter basic element | 5 |
| Figure 4. | Decision tree node | 9 |
| Figure 5. | Unico Load/Save tab | 13 |
| Figure 6. | Machine Learning Core tool - Data Patterns | 13 |
| Figure 7. | Machine Learning Core tool - Configuration | 14 |
| Figure 8. | Weka preprocess | 15 |
| Figure 9. | Weka classify | 15 |
| Figure 10. | Decision tree format | 16 |
| Figure 11. | Configuration procedure | 17 |
| Figure 12. | Assigning a result to a data pattern | 18 |
| Figure 13. | Configuration of Machine Learning Core | 18 |
| Figure 14. | Configuration of filters | 19 |
| Figure 15. | Configuration of features | 19 |
| Figure 16. | ARFF generation | 19 |
| Figure 17. | ARFF file | 20 |
| Figure 18. | Configuration of results and decision tree | 20 |
| Figure 19. | Meta classifier and device configuration | 21 |
| Figure 20. | Unico load configuration | 21 |
| Figure 21. | Unico Data window | 22 |
| Figure 22. | Unico - Machine Learning Core source registers | 22 |
| Figure 23. | Vibration monitoring decision tree | 23 |
| Figure 24. | Motion intensity decision tree | 24 |
| Figure 25. | 6D positions | 25 |
| Figure 26. | 6D decision tree | 25 |
| Figure 27. | Activity recognition for smartphone decision tree | 27 |
| Figure 28. | Weka cross-validation | 28 |
| Figure 29. | Gym activity recognition decision tree | 30 |
| Figure 30. | ARFF example | 31 |
| Figure 31. | Weka GUI Chooser | 31 |
| Figure 32. | Weka Explorer | 32 |
| Figure 33. | Weka Explorer - Attributes | 33 |
| Figure 34. | Weka Classify | 34 |
| Figure 35. | Weka Classify J48 | 34 |
| Figure 36. | Weka J48 classifier parameters | 35 |

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved