Overview

This document is intended to provide complete instructions on how to use the AWS version of Qeexo AutoML application to automatically create and deploy machine learning models. It will cover:

- System Requirements
- Running the Qeexo AutoML Web App
- Working with Projects
- Data Management
- Building Machine Learning Models
- Training Results
- Notification Center

System Requirements

Requirements to run the AWS version of Qeexo AutoML are listed below.

Hardware Requirements

- For AWS version, we recommend 1 PC running Windows 10 or 1 Mac machine running macOS.
- Please refer to installation guides for a list of Qeexo's supported hardware.

Software Requirements

- Frontend application (installed on Windows 10 or macOS on the host machine)

Running the Qeexo AutoML Web App

Start a browser and navigate to https://automl.qeexo.com

Working With Projects

The basic unit of organization in the Qeexo AutoML system is a "Project". A Project represents a collection of work to solve a specific machine learning problem on a particular Target Hardware. An example of a Project might be something like "TurbinePredictiveMaintenance", where the data, models, and tests are compiled with the end goal of using machine learning for predictive maintenance on Arduino Nano 33 BLE devices attached to turbines.

For an example "Air Gesture" project, refer to Detecting Air Gestures with QeexoAutoML.
Creating and Managing Projects

As a new user, you will be taken to the "Create Project" page after logging in, where you can specify a "Project Name", "Classification Type", and the "Target Hardware".

*Project Name*: Enter a name that is reflective of the purpose of your project.

*Classification Type*: Choose between Single-class or Multi-class. Single-class is suitable for identifying whether or not a data instance belongs to the given class, while multi-class classification can be used for applications distinguishing two or more classes.

*Target Hardware*: Select the hardware that will be used in your project. If you do not plan to use any hardware, please select "Arduino Nano 33 BLE Sense".
A list of all Projects created from your account can be found on the Projects page, which can be navigated to from the drop-down menu to the right of the User Profile icon.
Additional Projects can be created from the "Create Project" button either on the Projects page or at the upper right hand corner.

You may delete the Project or edit the Project Name from the ... icon to the right of each Project. You may also switch between projects by selecting from the Projects top center drop-down menu.

**Data Management**

After a Project has been created, you will be taken to the Data page, where you can upload or collect data.

We recommend that you first collect data with the Qeexo AutoML web app to ensure that everything is working properly before trying to upload your own data.

**Collecting Data**

Navigate to the Data Collection page to collect data using the Qeexo AutoML web app. This can be done either by clicking the "Collect Data" button or the "Data Collection" tab.
Step 1: Build Environment

An Environment is a physical setting with a given set of properties (e.g. acoustics, temperature, lighting). The range of this set of properties should match the range of the environment where the final machine learning model will eventually run. For example, training the machine learning models with data in your office will likely not work very well once you test the trained models on the factory floor.

Environments also contain information about the given sensor configuration settings. All data collected for a given Environment will have the same sensor configuration.

You can either "Build an Environment" by entering a unique "Environment Name", or "Select an Environment" to add more data to a previously recorded Environment. If selecting an existing Environment, the Sensor Configuration (in Step 2) will automatically populate with the Environment's previous settings.

You should name your Environment something easily recognizable to you, with details about the specific location. For example, "OfficeCoffeeTable" or "VestasTurbineSolano".
Step 2: Configure Sensors

Click "Edit" in Step 2 to view a list of the supported sensors on the Target Hardware, selected when you created this Project. You may select any combination of the sensors listed on this menu to collect your data. After selecting the sensors, you will need to configure the corresponding sampling rate (ODR, or Output Data Rate) for each sensor, and the full scale range (FSR) when available.

The optimal ODR and FSR configuration depends on the use case. For example, using an ODR of over 1000 Hz will generally give some useful detail when detecting and analyzing turbine vibrations, while ODR of ~100 Hz or less may be more suitable for human activity detection. Having a larger FSR will allow you to see larger variations in sensor values, but the resolution will be compromised, so you will get less detail.

Finding the optimal settings may take some trial-and-error. Qeexo AutoML's data visualization support (see section on Visualizing Data) may help in finding these settings. For example, if the peaks of your signal appear to be cut off, or saturating, you may want to increase the FSR of your sensor.
Currently, a few limitations exist:

- Accelerometer and Gyroscope must share the same ODR.

After selecting the desired sensors and settings, click “Flash Data Collection App” to flash the data collection application to the Target Hardware.

Note: If this is your first data collection for a given sensor configuration, Qeexo AutoML needs to build the data collection application first. This can take a few minutes, but it will only have to be done once for each new configuration.

Note for the SensorTile.box users: If connection/flashing fails, your JTAG adapter cable may be different from the one pictured in the QeexoAutoML_Installation_Guide. Try flipping (turn it 180 degrees) the end connected to the Sensortile.box and retest.

**Step 3: Collect Data**

Qeexo AutoML currently supports a variety of supervised classification algorithms for machine learning. For each of these algorithms, all data used for training must have an associated Class Label.

For the multi-class case, at least two unique classes must be defined. For most problems, we recommend that at least one of the classes be a "baseline" class that represents the typical environmental noise or behavior.

Whether or not baseline data is necessary depends on the use case and data selected. In general, the classes collected for multi-class classification should represent the full set of possible states for the given Environment. For example, if you want to build a multi-class model which can distinguish between various types of machine vibrations (e.g. slow, medium, fast), you should collect data which represents all possible different types of machine vibrations. In this case, if the model output will also
be used in the "baseline" case where the machine is off, this data should be collected as well.

**Baseline Data:**

- Baseline data can be collected by setting the data type to Continuous, and leaving data collection application to run while the environment is in a steady state of rest or typical operating behavior.
- Some machine learning problems require collecting baseline data to differentiate events of interest from normal environmental conditions.
- Baseline data is usually associated with each Environment (since different Environments will often have different baseline data characteristics).
- For example, baseline data might be "NoGesture" in gesture recognition, "None" in kitchen appliance detection, or "AtRest" in logistics monitoring.

**Data Collection Type**

Qeexo AutoML currently supports 2 types of data collection: "Continuous" and "Event". Each Dataset must have a type.

- **Continuous**
  - Continuous collections will collect data for n consecutive seconds, where n is the Number of Seconds defined.
  - Continuous type data collection should be used to collect data over a fixed time interval where the Class Label does not change.
  - For example, Continuous data can be "Normal" in Predictive Maintenance, "Running" in activity recognition, or "Occupied" in occupancy detection.

- **Event**
  - Event data collection is provided as a method for quickly and automatically labelling Classes that typically last for short periods of time.
  - Event collections will collect n Events, where n is the Number of Instances defined.
  - All Events must be less than 10 seconds because they must fit entirely within the Recording Window (defined below).
  - For example, an Event can be a "Knock" in surface gesture recognition, a "Fall" in human fall detection, or an "Impact" in logistics monitoring.

**Class Label**

A Class Label is a machine learning concept, normally a word or phrase to label the event or condition of interest. For example, "Normal", "WornBearings", and "WindingFailure" can be classes in our Turbine Predictive Maintenance problem.

For Continuous data, the Class Label applies to all of the data collected. For Event data, the Class Label applies only to the collected instances -- the rest of the collection will likely represent the "baseline" class and is not used for training.

You must define one Class Label at a time when collecting data by entering a text string in the given field.
**Number of Instances / Number of Seconds**

This sets the duration of the data collection.

For both data collection types, a continuous stream of data will be recorded from the Data Recording page. If Continuous type data is selected, the recording will stop after \( n \) seconds, where \( n \) is the number of seconds you entered. If Event type data is selected, the recording will stop after \( n \) Events, where \( n \) is the number of instances you entered.

For Event data collection, AutoML uses the recorded signal to detect and segment the Events in real-time. Only the portions of the signal which are highlighted as Events will be associated with the given Class Label.

More data generally leads to higher performance. Depending on the complexity of the use case, the number of classes, the quality of the data, and many other factors, the optimal and minimum number of instances or seconds to collect can vary greatly. We recommend starting with at least 30 Instances or Seconds for each Class Label, but much more data may be required if the classes are highly variable or if the problem is sufficiently complex.

**Recording Window**

This sets the duration in time which the data collection app will watch for an instance of the given Event class.

For Event data collection, AutoML automatically runs a localization algorithm which will try to find the segment of interest within the given Event instance. This is provided as a convenient method for quickly and accurately labelling classes which typically last for only a few seconds or less.

To configure this parameter, you will need to have an idea of the typical length for the Event class which you are trying to collect. The Recording Window should be set to a value greater than this typical length, so that AutoML will have some period of "baseline" data on either side of the Event for the purposes of accurate localization.

**Recording Continuous Data**

After completing the previous steps, the "Record" button should now become click-able. (If it is not, check previous steps.)
After clicking "Record", you will be directed to the Data Recording page:

When you are ready to start data collection, click "Start" to begin. The text in the center circle will change from "Ready" to "Initialize" while the data collection software is starting up.
After a few seconds, data collection will start when you see the circle turn green and display "Go". Data is now being collected.

For Continuous data, the counter will continuously count up by 1 every second until it reaches the desired Instances/Seconds supplied. Take care to begin your interested action (e.g. "Running") before the circle turns green to ensure that no undesirable "blank" data gets recorded as Class Label data.

Once the specified number of Seconds have been collected, the labelled data will be uploaded to the database, and user will be redirected to the Data Collection page.
You can collect more data of the same or different Class Label from the Data Collection page. Note that, for a multi-class classification Project, you will need at least 2 distinct classes (2 different Class Labels) to be able to train a machine learning model.

**Recording Event Data**

After completing the previous steps, the "Record" button should now become click-able. (If it is not, check previous steps.)
For Event data, the counter will increment by 1 whenever an Event is collected. Note that all events will be labeled as the "Class Label" supplied earlier.

Press START to begin recording the event. You have the n number of seconds to record the actual event.

After recording an event, the sensor data is visualized. Repeat until the number of events reaches n (the given Number of Instances).
Note the blue highlighted portion of the event collection shown above. This is the "localized" event, which is the segment of the collection which is labelled with the given Class Label and which is used for model building. The rest of the recording represents a different, unknown Class Label and is therefore not used for training.

**Re-recording Data**

If you believe a mistake has been made when recording data, and the data has been contaminated, you can re-record the data from the bottom of the Data Collection page. You can click "Re-Record" to overwrite the existing data. Alternatively, you can click on the "Trash" icon to delete the Dataset and start over.

![Re-recording Data](image)

**Uploading Dataset**

From the Data page, you may upload previously-collected datasets to AutoML directly. These uploaded datasets can be used to train machine learning models, and can be combined with additional data that has been collected through the Qeexo AutoML platform.

![Uploading Dataset](image)

Click "Upload Dataset" to upload a single .csv file. Each .csv should contain one or more data collections following the Qeexo-defined data format below. All data contained in the .csv file must come from the same sensor configuration, which you will enter after uploading the .csv file. If you have more than 70 MB of data, you will need to split it into multiple .csv files.
Refer to the file sample datasets if you would like some example data for upload.

All rows with the same class label will be treated as a single, continuous collection. AutoML’s upload data functionality will produce the best results if each file contains data from only a single collection period on one device. If there were gaps in the collection period, or if the data was taken from multiple devices, it is recommended to split the separate collections into multiple files for upload.

Note: all uploaded data will be treated as "Continuous" type data. "Event" type data is currently only supported from our Data Collection page. Data with the same Class Label must be of the same data type (Event/Continuous).

Data Format Specification

Qeexo AutoML can accept two different CSV formats:

- V1 is easier to understand and prepare.
- V2 can handle multiple data points in a single line; no upsampling is required thus minimizing the chance of data duplication.
Note that V1 and V2 are for human readability - there is no need to indicate which format because Qeexo AutoML can detect the formatting automatically.

Qeexo AutoML will match the closest ODR of the selected hardware against that of the uploaded data.

**V1 CSV format**

The data file consists of 3 parts, in the following order: `timestamp`, "sensor data", and `label`.

1. **timestamp** (exactly 1 column):
   - Type: **float** (milliseconds)
   - We up-sample lower-sampling-rate sensor data to match maximum sampling rate.
   - Timestamps should indicate the maximum sampling rate.
   - Incorrect timestamps may cause data check failure (see Data Check section below).

2. **sensor data** (1 or more columns from list):
   - Type: **integer**
   - Sensor data column names must match the exact strings that Qeexo uses. For example, `accel_x` instead of `accelerometer_x` (see below).
   - Column names not matching pre-defined strings may cause data upload to fail.
   - Refer to the following table for sensors are supported in the hardware you chose:
<table>
<thead>
<tr>
<th>Sensor</th>
<th>CSV column headers</th>
<th>Arduino Nano 33 BLE Sense</th>
<th>Arduino Nano 33 IoT</th>
<th>RA6M3</th>
<th>SensorTile.box</th>
<th>STWINKT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>accel_x, accel_y, accel_z</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>gyro_x, gyro_y, gyro_z</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Magnometer</td>
<td>magno_x, magno_y, magno_z</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Temperature</td>
<td>temperature</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Humidity</td>
<td>humidity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Pressure</td>
<td>pressure</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Microphone</td>
<td>microphone</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Analog microphone</td>
<td>microphone Analog</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Light (single channel)</td>
<td>light</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Ambient light (CRGB)</td>
<td>ambient_c, ambient_r, ambient_g, ambient_b,</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RCDA: Clean Dry Air Resistance</td>
<td>rcda</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ETOH : Ethanol</td>
<td>etoh</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TVOC : Total volatile organic compounds</td>
<td>tvoc</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IAQ : Indoor air quality</td>
<td>iaq</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ECO2 : Estimated Carbon Dioxide</td>
<td>eco2</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RMOX : Metal Oxide Resistance</td>
<td>rmox</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Low power accelerometer</td>
<td>accel_lowpower_x, accel_lowpower_y, accel_lowpower_z</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>High sensitivity accelerometer</td>
<td>accel_highsensitive_x, accel_highsensitive_y, accel_highsensitive_z</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

3. **label** (exactly 1 column):
   - **Type**: string
   - Label column contains the verified Class Label (decision) for each row of sensor data.
   - We recommend that each row in `.csv` file is sorted with timestamp and grouped by class label.

4. **data_type**:
   - **Type**: string ("CONTI" or "EVENT")
   - "CONTI" for continuous data, "EVENT" for event data
   - **Ignore this column while importing data**

5. **recording_id**:
   - **Type**: int (empty or 1,2,3,...)
   - Empty if data is continuous data or not belong to any recording
6. **event_id**:
   - Type: int (empty or 1,2,3,..)
   - Empty if data is continuous data or not belong to any event

---

**V2 CSV format**

The format for uploaded data consists of 3 parts, in the following order: a timestamp column, sensor data column(s), and a label column.

1. **timestamp** (exactly 1 column):
   - Type: **integer** (milliseconds)
   - Timestamp column contains the time associated with the sample(s) in a given row.
   - Note that each line increases by 50 ms. If the data is sampled at 100Hz, then the number of samples in each line should be 50/(1000/100) or 5 samples.
   - Timestamp column should be in **ascending sorted order**.
   - In the case of a row containing multiple samples, the timestamp should be the time associated with the most recent sample in the row (i.e. the time at the end of the given sampling period).
   - Incorrect timestamps may cause a data check warning (see Data Check section below).

2. Sensor data (1 or more columns):
   - Type: **list of integers** (or list of lists of integers)
   - Each sensor data column must represent all channels of a supported sensor type on one of the AutoML-supported hardware platforms. See below table for a list of currently-supported column names.
   - Each cell in the sensor data column contains all of the sensor's samples associated with the given row timestamp. A cell can contain no samples (e.g. ",[]"), a single sample (e.g. ",[100],"), or multiple samples bracketed in a list (e.g. ",[100,99,101],").
   - For sensors with multiple channels, the format is a list of lists of integers, grouped by time. The expected channel ordering for each inner list is x, y, z (for accel, gyro, magno) and c, r, g, b (for ambient light sensor).
   - Incorrect column names will cause data upload to fail.
   - Incorrect sensor configurations will cause a data check warning (see Data Check section below).
<table>
<thead>
<tr>
<th>Sensor</th>
<th>CSV column header</th>
<th>Arduino Nano 33 BLE Sense</th>
<th>Arduino Nano 33 IoT</th>
<th>RA6M3</th>
<th>SensorTile.box</th>
<th>STWINKT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer (XYZ)</td>
<td>accel</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Gyroscope (XYZ)</td>
<td>gyro</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Magnometer (XYZ)</td>
<td>magno</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Temperature</td>
<td>temperature</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Humidity</td>
<td>humidity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Pressure</td>
<td>pressure</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Microphone</td>
<td>microphone</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Analog microphone</td>
<td>microphone_analog</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Light (single channel)</td>
<td>light</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Ambient light (RGBC)</td>
<td>ambient</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCDA: Clean Dry Air Resistance</td>
<td>rcda</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ETOH : Ethanol</td>
<td>etoh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>TVOC : Total volatile organic</td>
<td>tvoc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>compounds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IAQ : Indoor air quality</td>
<td>iaq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ECO2 : Estimated Carbon Dioxide</td>
<td>eco2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RMOX : Metal Oxide Resistance</td>
<td>rmox</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Low power accelerometer</td>
<td>accel_lowpower</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>High sensitivity accelerometer</td>
<td>accel_highsensitive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

3. **label** (exactly 1 column):
   - Type: **string**
   - Label column contains the class label for each row of sensor data.

4. **data_type**:
   - Type: string ("CONTI" or "EVENT")
   - "CONTI" for continuous data, "EVENT" for event data
   - Ignore this column while importing data

5. **recording_id**:
   - Type: int (empty or 1,2,3,...)
   - Empty if data is continuous data or not belong to any recording
   - Ignore this column while importing data

6. **event_id**:
   - Type: int (empty or 1,2,3,...)
   - Empty if data is continuous data or not belong to any event
   - Ignore this column while importing data
As shown in the image above, all sensor samples are formatted as lists of integers or lists of lists of integers. For the lowest ODR sensor (humidity), there were only 1-2 samples per sampling period (50 ms for this collection), while the highest ODR sensor (microphone) has hundreds of samples per row. For the multi-channel sensors, the x, y, z and c, r, g, b channels are grouped together in-time and recorded inside their own lists.

**Upload Data and Confirm Sensor Configurations**

Qeexo AutoML will detect the most appropriate sensor configurations for your uploaded .csv file. You will have the opportunity to accept or select a different values of sensor type, ODR, and FSR. We recommend accepting the auto-detected values. Incorrect sensor configurations may break the library-building process or generate sub-optimal results.
If your data is collected at an ODR that is too far off from the supported ODR of the selected hardware, you may need to consider up-sampling or down-sampling the data before uploading them to Qeexo AutoML.

**Data Check**

Data check verifies the quality of the data, whether uploaded or collected. A failure in data check will not prevent you from using the data to train machine learning models. However, poor data quality may result in poor model performance.

Qeexo AutoML currently looks for the following data issues:
- collected data does not match the selected sensors in the Sensor Configuration step
- collected data does not match the selected sampling rate in the Sensor Configuration step
- collected data contain duplicate or missing timestamps
- collected data has duplicate values or constant values
- collected data contains invalid values including NaN or inf
- collected data is saturated

Here is an example of a data check with warnings:

<table>
<thead>
<tr>
<th>Data Issues Found</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor(s) mismatch</td>
<td>PASS</td>
</tr>
<tr>
<td>Sampling rate mismatch</td>
<td>PASS</td>
</tr>
<tr>
<td>Duplicated/missing/bad timestamps</td>
<td>PASS</td>
</tr>
<tr>
<td>Duplicated/Constant values</td>
<td>WARNING</td>
</tr>
<tr>
<td>Invalid values (NaN/Inf)</td>
<td>PASS</td>
</tr>
<tr>
<td>Data saturated</td>
<td>PASS</td>
</tr>
</tbody>
</table>

A green PASS icon indicates that data check has passed; a yellow WARNING icon indicates that the data contains one or more issues from the list above; a red ERROR icon indicates that something went wrong during data collection or during data check (connection error or device error), the data may not be usable if it remains ERROR after refresh.
Training data and Test data

Training data

A set of data instances you plan to train and build models with.
To make sure you are collecting or uploading training data, navigate to Data page and make sure Training is selected in the dropdown menu before you collect or update data.

Test data

A set of data instances used to evaluate the performance of a trained model.
To make sure you are collecting or uploading test data, navigate to Data page and make sure Test is selected in the dropdown menu before you collect or update data.
Linking test data with training data

A test dataset can be associated with at most one training dataset and both must have the same sensor configuration AND data type (Continuous/Event).

When a test dataset is linked to a training dataset, it will be used to evaluate the performance of the final model created using said training data.

Note: If there are multiple datasets with the same label, you will only have to link the test datasets to one of the training datasets.

Consider the following example:
For a multi-class classification project applied to activity tracking, you want to evaluate the model performance against actual users. You first collect/upload test data from users, link the test label to the corresponding training label you want to evaluate against (ex: running-test-age-name to running-training and walking-test-age-name to walking-training). You will see the overall test data accuracy and other matrix after the model is done training. Later, let's say you want to find out how the model perform for users with age 65 and up, you can unlink the test datasets that are not from users over the age of 65 from training datasets and train again, the result will give you an idea of what performance you can expect for users in the age group of 65+ using this particular machine learning model.

There are several ways that you can link test data with training data:

**Collect test data**

After clicking on collect test data, a pop up will appear and ask which training data you would like to associate with.
You will then enter data collection page with the sensor configuration pre-set to the same as the training data you had picked.

**Link from training data**

Manage test data for each specific training dataset by clicking on the linking icon to see and manage which test data are linked and available to be linked.
One training data can link multiple test data as long as they share the same sensor configuration AND data type (Continuous/Event).

**Link from test data**

Manage test data by looking at a list of test datasets and see what each test data is linked to by clicking on the linking icon.
A test dataset can be associated with at most one training dataset and both must have the same sensor configuration AND data type (Continuous/Event).

Training settings (Step 2)

This step is an optional step to show an overview of test data management and whether any test data is linked to the selected training datasets. It gives you a chance to do any last minute adjustments on test data linkage before training starts.
View Test Data Result

If test datasets are included in the build, you will see an additional TEST PERFORMANCE column in model result.

Performance Summary will now also include matrix and graphs with test data.

Viewing and Managing Project Data

All of the Datasets associated with the current Project can be viewed and managed from the Data page. You can review the Dataset Information including its Sensor Configurations and Data Check results, as well as visualize and delete them.

From this page, select Datasets containing more than one Class Label to begin training machine learning models. (Also see section below on Building Machine Learning Models.)
Visualizing Data

On the Training page, you will see all the Datasets collected and uploaded in the current Project.

At the bottom of the Data Collection page, you will see all the Datasets collected and uploaded in the current Environment.
Click on the "Graph" icon on either page to go to the Data Visualization page to view and analyze the data for the selected sensor.

Click drop-down list in the upper left of the screen can select different sensor data.

The first 100 seconds (continuous) or instances (events) of each Dataset will be shown in visualization. If there are over 100, the extra will be paginated.

**Building Machine Learning Models**

Navigate to the Data page to build machine learning models with collected and/or uploaded training data.

Select which training datasets to use to build machine learning models by clicking the checkbox at the left of each Dataset.
Note that the selected Datasets should ideally be from the same Environment, but Qeexo AutoML will allow you to train Datasets from different Environments as long as the selected sensors and Sensor Configuration are identical.

Once the desired Datasets are selected, click "Start New Training" button to configure Training Settings. Note that the "Start New Training" button is only clickable when Datasets containing 2 or more Class Labels are selected in Multi-class classification. However, for One-class classification, the button becomes clickable as soon as the first collection has been selected.

There is a minimum amount of data Qeexo AutoML needs for each Class Label in order to train machine learning models. For Event data classes, this minimum is 10 events. For Continuous data classes, this minimum is 640 samples from the highest ODR sensor. For example, if your sensor configuration is 104 Hz accelerometer & 25 Hz humidity, you need to collect data for at least 7 seconds (because 104 Hz * 6 seconds < 640 samples).

Note that satisfying this minimum requirement does not guarantee performance/accuracy; it is just the minimum amount of data our platform will work with. For good performance, you should likely collect much more than this minimum amount of data.

Training Settings

Step 1. Group Labels

This step is an optional step in case you have many class labels that you want to group together as a single class before model training. Consider the following example:

For a one-class classification project applied to anomaly detection, you may have machinery data that is labelled based on two different types of motion: vertical rotation and horizontal rotation. Since both of these classes are expected behavior, it is convenient to group these labels as a "Normal" group to feed into single-class classification.
Note that you can only group datasets with the same data type. (There are 2 data types: Event and Continuous.)

This is an optional step that can be bypassed by pressing the SKIP button.
Step 2. Linked test data

This step is an optional step to show an overview of test data management and whether any test data is linked to the selected training datasets. It gives you a chance to do any last minute adjustments on test data linkage before training starts.

Note: One training data can link multiple test data as long as they share the same sensor configuration AND data type (Continuous/Event).

---

Step 3. Sensor Selection

Next, there is an option to have Qeexo AutoML automatically select sensors and feature groups for optimal model performance. If you have collected data from multiple sensors, but you are not sure which may be helpful for the given problem, it is recommended to enable this feature.

Qeexo AutoML computes hundreds of statistical and signal-processing-based features from raw sensor data. Turning on Automatic Sensor and Feature Group Selection will automatically select a subset of sensor inputs as well as a subset of features relevant to the current use case. Using a smaller number of sensors and/or features will reduce classification latency and model size. However, this operation may increase the training time due to the additional computation required to select features. This increase in time largely depends on the amount of data and the machine learning algorithm used.

Note that this automatic selection applies to both sensor and feature groups. If you know what sensors should be used for the problem, but you still want to enable feature selection, that is possible on the following page.
If Manual Sensor Selection is enabled, you will be able to select a subset of sensors to feed into the ML pipeline. Note that:

- All sensors included in Project Creation are available for selection in this stage
- Individual axes from multi-channel sensors can be selected independently. These include:
  - X-, Y-, and Z-axis selection for Accelerometer, Gyroscope, and Magnetometer sensors
  - Red, Green, Blue and Clear Light channel selection for Ambient Light sensor
Sensor Selection

- Manual Sensor Selection

- ACCELEROMETER
  - X-Axis
  - Y-Axis
  - Z-Axis

- GYROSCOPE
  - X-Axis
  - Y-Axis
  - Z-Axis

- MAGNETOMETER
  - X-Axis
  - Y-Axis
  - Z-Axis

- TEMPERATURE

- HUMIDITY

- PRESSURE

- PROXIMITY

NEXT  CANCEL
Under manual selection, you can visualize data collected from sensors using a general-purpose dimensionality reduction algorithms called UMAP (Uniform Manifold Approximation and Projection) and PCA (Principal Component Analysis). UMAP is a novel manifold learning technique for dimension reduction. For further details about UMAP algorithm, please refer to "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction" available at https://arxiv.org/abs/1802.03426. PCA linearly transforms input features into a lower-dimensional space where the variance of the
features is maximally preserved. PCA is very well established statistical technique for dimensionality reduction while preserving the variance. Similar to UMAP plot, we project features into two-dimensional space for the PCA visualization as well. These visualizations can often determine which sensors might be useful for classification. Examining these UMAP and PCA plots before making a final manual sensor selection is encouraged.

Step 4. Feature group configuration

Note that this step is automatically skipped when "automatic sensor and feature group selection" is chosen in the previous screen.

When you chose manual selection in the previous sensor screen, you will be presented an option to select between automatic and manual feature selection:
You may want to refer to the following description of features during manual selection:
<table>
<thead>
<tr>
<th><strong>Feature group</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw Statistics</td>
<td>Computes the statistics based on the raw sensor data such as range, standard deviation, skewness, etc.</td>
</tr>
<tr>
<td>Auto-Correlation</td>
<td>Compute the auto-correlations of raw sensor data, measuring how much the signal correlates with itself through time</td>
</tr>
<tr>
<td>Raw Peak</td>
<td>Includes two parts, linear fit and zero crossing rate (ZCR). Linear fit computes the statistics (mean, standard deviation) on coefficients when fitting a linear regression line to the signal peaks; ZCR Computes the zero-crossing rate on signal peaks. Zero crossing rate is the number of times signal crosses zero.</td>
</tr>
<tr>
<td>FFT Power Simple</td>
<td>Computes the statistics based on the power of Fast Fourier Transform (FFT) performed on raw sensor data. Includes quantities such as range, mean, standard deviation, skewness, root mean square, etc.</td>
</tr>
<tr>
<td>FFT Power Advanced</td>
<td>Computes several other quantities based on the power of FFT. Includes the spectral centroid, bandwidth, rolloff point, decrease, flatness, linear slope, etc.</td>
</tr>
<tr>
<td>FFT Power Binned</td>
<td>Computes the power spectrum of FFT coefficients binned linearly. FFT Power Octave: Computes the power spectrum of FFT coefficients bucketed into octaves.</td>
</tr>
<tr>
<td>FFT Power Thirds</td>
<td>Computes the power spectrum binned into music thirds where numbers are binned in 3 logarithmically spaced steps.</td>
</tr>
<tr>
<td>MFCC</td>
<td>Computes the Mel-frequency cepstral coefficients (MFCC) based on the power of FFT coefficients. MFCCs are the discrete cosine transform of the Mel-filterbank coefficients where Mel-filterbank coefficients are the log of power at 16 distinct Mel-frequencies.</td>
</tr>
<tr>
<td>MFCC Delta</td>
<td>Computes the difference between the current coefficients and the previous coefficients in MFCC features.</td>
</tr>
<tr>
<td>MFCC Delta Delta</td>
<td>Computes the difference between the current and the previous delta value in MFCC delta features.</td>
</tr>
</tbody>
</table>

If the automatic option is chosen, Qeexo AutoML will select the optimal set of features based on final model accuracy as input into model training.

Note that features are grouped under each sensor. For each of the selected sensors, at least one feature group is required under the manual mode.

Similar to sensor selection screen, users can visualize data collected from sensors as well as feature groups using the UMAP visualization tool described in the previous section.
Step 5. Inference Setting

Inference Settings

See below sections for more details about the Instance Length and Classification Interval.

Here are the recommended values for a couple of sample use cases:

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Instance Length (ms)</th>
<th>Classification Interval (ms)</th>
<th>Max Sensor ODR (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air Gesture Recognition</td>
<td>2000</td>
<td>250</td>
<td>417</td>
</tr>
<tr>
<td>Machine Vibration</td>
<td>300</td>
<td>100</td>
<td>6660</td>
</tr>
<tr>
<td>Human Presence Detection</td>
<td>5000</td>
<td>2000</td>
<td>100</td>
</tr>
</tbody>
</table>

**Instance Length**

This is sets the length of the sensor data which the model will use to make classifications. The maximum allowable value for this property is set based on the highest sensor ODR in the given sensor configuration.

The optimal value depends on the real-world time duration of the events corresponding to the Class Labels that you want to capture/detect with the sensors, as well as the selected ODRs used during data collection.

For example, in machine vibration problems, the most important requirement for the signal data is high ODR, because higher ODRs will increase the amount of high frequency information available to the model. For these type of problems, we should often use the highest available sampling rate, and then use the maximum allowable instance length for that ODR (~300 ms @ 6.6kHz).

However, in human activity detection and monitoring, 300 ms would likely not be a sufficient instance length for good performance. We need to make sure the window is long enough to capture some human motion and interaction with the environment, which may require several seconds of data in total. For this case, we should pick an instance length which we want to achieve (e.g. 5000 ms), and then we should use the highest possible ODR which still allows us to achieve this instance length.

**Classification Interval**

This sets the number of milliseconds between requests for classification.

For example, if Classification Interval is set to 200 ms, Qeexo AutoML will produce a classifier that classifies incoming data at a rate of 5 Hz.
This value should be set relative to the average time between class changes in the real world -- if the classes may change quickly or last for a very short period of time, Classification Interval needs to be set low so that classifications are run near-constantly in order to capture the events of interest.

For problems like human presence detection, where based on the nature of the problem the classes cannot change often, we may select a higher value (e.g. 2000 ms) in order to save on power consumption and network bandwidth.

**Step 6. Model Settings**

This page determines which model types are trained. For multi-class classification, it also contains switches for enabling a few different features related to model building.

**Single-class Classification**

**Algorithm Selection**

For single-class classification, Qeexo AutoML supports the following machine learning algorithms:

- Isolation Forest (IF)
- Local Outlier Factor (LOF)
- One Class Random Forest (ORF)
- One Class SVM (OCSVM)

**Multi-class Classification**

**Hyperparameter Tuning**

Hyperparameters are a set of adjustable parameters of machine learning models. These parameters affect the accuracy, runtime, and size of machine learning models. Different models have different parameters depending on the model architecture. AutoML provides built-in option for tuning these hyperparameters. There is a simply switch users need to flip if hyperparameter optimization is desired. If this option is enabled, AutoML tunes hyperparameters using a collection of optimization techniques tailored to TinyML applications. It maximizes accuracy while it ensures that all resource usages are under constraints (e.g., firmware binary size and memory usage). This option will often improve final model accuracy at the expense of additional runtime for model-building.

**Generate Learning Curve(s)**

If enabled, this option will produce learning curves for the given data set. Learning curves visualize how your model is improving as more data is added. These curves can be extrapolated, which can be useful for determining if the model may benefit from additional data collection.

As shown in the example below, the "Circle" and "Punch" gestures are still improving with additional data. It is likely that they would continue to improve if more data is collected.
Note: If the dataset that is used for training is very small, the learning curves may not be accurate. The model may be very good at classifying the limited data it's seen, but might not generalize to new cases. In that case, even if the learning curve does not show it, it is safe to assume that final model performance will improve with additional data collection.

Algorithm Selection

For multi-class classification, Qeexo AutoML supports the following machine learning algorithms:

**Ensemble Methods:**
- Gradient Boosting Machine (GBM)
- Random Forest (RF)
- XGBoost (XGB)

**Neural Networks:**
- Artificial Neural Network (ANN)
- Convolutional Neural Network (CNN)
- Convolutional Recurrent Neural Network (CRNN)
- Recurrent Neural Network (RNN)

**Support Vector Machines:**
- Polynomial Support Vector Machine (POLYSVM)
- RBF Support Vector Machine (RBFSVM)
- Support Vector Machine (SVM)

**Others:**
- Decision Tree (DT)
- Gaussian Naive Bayes (GNB)
- Logistic Regression (LR)
Support for additional algorithms will be added in the future.

Note that Neural Networks models may take longer to train, due to the significant computation required for the training process.

Algorithm Selection

<table>
<thead>
<tr>
<th>Method</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensemble Methods</td>
<td>GBM, RF, XGB</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>ANN, CNN, CRNN, RNN</td>
</tr>
<tr>
<td>Support Vector Machines</td>
<td>POLYSVM, RBFSVM, SVM</td>
</tr>
<tr>
<td>Others</td>
<td>DT, GNB, LR</td>
</tr>
</tbody>
</table>

CONFFIGURE

Pressing CONFFIGURE (available for some models) will yield the following configuration screen:

<table>
<thead>
<tr>
<th>Method</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensemble Methods</td>
<td>GBM</td>
</tr>
<tr>
<td>Gradient Boosting Machine (GBM)</td>
<td>CONFIGURE</td>
</tr>
</tbody>
</table>
Quantization denotes an option to conduct quantization-aware training so as to achieve model size reduction.

There are additional configurable options to fine tune the ANN model:
Artificial Neural Network (ANN)

Configurable Parameters

- **LEARNING RATE**: 0.01
  Value from 0.00001 to 0.5

- **LAYER 1 UNITS**: 5
  Value from 2 to 10

- **LAYER 2 UNITS**: 10
  Value from 2 to 10

- **LAYER 3 UNITS**: 10
  Value from 2 to 10

- **DROPOUT RATE**: 0.1
  Value from 0.1 to 0.9

- **EPOCHS**: 50
  Value from 1 to 1000

- **BATCH SIZE**: 20
  Value from 10 to 200

- **ACTIVATION**: ReLU

- **BATCH NORMALIZATION**: False
<table>
<thead>
<tr>
<th>Configurable Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>Scaling parameter which sets the step size at each iteration in optimization of the cost function</td>
</tr>
<tr>
<td>Layer 1 units</td>
<td>Number of nodes in layer 1</td>
</tr>
<tr>
<td>Layer 2 units</td>
<td>Number of nodes in layer 2</td>
</tr>
<tr>
<td>Layer 3 units</td>
<td>Number of nodes in layer 3</td>
</tr>
<tr>
<td>Dropout rate</td>
<td>Fraction of units to drop during each training round, applied to all network layers</td>
</tr>
<tr>
<td>Epochs</td>
<td>Number of passes through the complete training dataset; one epoch means the network will use each training instance exactly once</td>
</tr>
<tr>
<td>Batch size</td>
<td>Number of training examples in one training round; higher batch sizes may have faster runtimes, but are more likely to get stuck in local optima</td>
</tr>
<tr>
<td>Activation</td>
<td>Function applied to the outputs of the neurons</td>
</tr>
<tr>
<td>Batch normalization</td>
<td>If true, apply normalization process to the output of each layer, typically helpful for improving the convergence and stability of the training process</td>
</tr>
</tbody>
</table>

Note: many of these parameters interact with each other in unique and non-intuitive ways. Unless you have significant experience tuning deep learning models, you may want to consider using the automatic hyperparameter optimization tool.

Similarly there are configurable options to fine tune the CNN model:

**Convolutional Neural Network (CNN)**
*Deep learning model - may take longer to train*
<table>
<thead>
<tr>
<th>Configurable Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensor Length limit</td>
<td>Threshold length that determines whether to stop adding convolution layers (reducing the length limit will lead to more convolution layers)</td>
</tr>
<tr>
<td>Learning rate</td>
<td>Scaling parameter which sets the step size at each iteration in optimization of the cost function</td>
</tr>
<tr>
<td>Batch size</td>
<td>Number of training examples in one training round; higher batch sizes may have faster runtimes, but are more likely to get stuck in local optima</td>
</tr>
<tr>
<td>Dense layer units</td>
<td>Number of nodes in the final network layer</td>
</tr>
<tr>
<td>Drop out rates</td>
<td>Fraction of units to drop during each training round, applied to all network layers</td>
</tr>
<tr>
<td>Epochs</td>
<td>Number of passes through the complete training dataset; one epoch means the network will use each training instance exactly once</td>
</tr>
<tr>
<td>Input layer filters</td>
<td>Number of filters in the first convolution layer</td>
</tr>
<tr>
<td>Intermediate layers filters</td>
<td>Number of filters in all the intermediate convolution layers</td>
</tr>
<tr>
<td>Input layer strides</td>
<td>Number of samples to move at each step along one direction for the first convolution layer</td>
</tr>
<tr>
<td>Intermediate layers strides</td>
<td>Number of samples to move at each step along one direction for all intermediate convolution layers</td>
</tr>
<tr>
<td>Augment</td>
<td>If true, apply data augmentation technique to prevent overfitting; will lead to higher training time due to larger amount of data</td>
</tr>
<tr>
<td>Batch normalization</td>
<td>If true, apply normalization process to the output of each layer, typically helpful for improving the convergence and stability of the training process</td>
</tr>
<tr>
<td>Activation</td>
<td>Function applied to the outputs of the neurons</td>
</tr>
<tr>
<td>Input layer kernel size</td>
<td>Filter kernel size in the first convolution layer</td>
</tr>
<tr>
<td>Intermediate kernel size</td>
<td>Filter kernel size in the all intermediate convolution layers</td>
</tr>
</tbody>
</table>

Configuration sub-menu for other algorithms will be added in the future.
Training Process

Click "Start Training" with one or more selected machine learning algorithms to begin the training process. Selecting more than one type of algorithm is recommended, so that results could be compared.

Real-Time Training Progress pops up after training begins. The top row shows the progress of common tasks (e.g. featurization, data cropping, etc.) shared between different algorithms, followed by the build progress of each of the selected models.

Multi-class classification:
Single-class classification:
At the end of the training process, Qeexo AutoML will flash, in sequence, each of the built models to the hardware device to test and measure the average latency for performing classifications.
Training Result

Click "Training Result" to navigate to the Models page (also reachable from the top navigation bar), where all of the previous trainings will be listed, with the most recent one on top. The current training will be expanded to show relevant information, including the type of machine learning model, cross validation accuracy, latency, size, and additional details. It also allows you to save each model to your computer or push a selected model to Target Hardware for Live Testing.

ML Model

Each entry is differentiated by the algorithm with which each model had been built. We also call these machine learning "packages" because they include supporting code such as sensor drivers in addition to the machine learning models built by Qeexo AutoML.
**Cross Validation**
This is the average classification accuracy for 8 different models, each trained and tested on different, mutually-exclusive subsets of the given data. This is always a value between 0 and 1, with 0 being the worst accuracy and 1 being perfect accuracy.

**F1-Score**
F1-score also lies within the unit interval; the best score is 1, and it approaches 0 as performance gets worse. F1-score factors false positives, false negatives, and true positives. F1-score thus is an important model performance metric. Accuracy only can obscure some important aspects of model performance if a large proportion of a dataset belongs to one class. In contrast, F1 score is more tolerant to this type of class imbalance problems. Operating the model at the peak of the F1-score means the rate of True Positives and False positives are optimized. To the either side of this point, either True positives or False positives dominates.

**Latency**
Latency is the average time (in milliseconds) required for the machine learning model to compute the prediction of a single instance. It includes time spent on featurization of sensor data and running inference with the model. We calculate this average empirically by first flashing each model to the Target Hardware, running 10 inferences, then taking the average.

**Size**
This is the combined file size of the model parameters and the model interpreter (in kilobytes). Model interpreter scans the respective model parameters and computes the classification from sensor readings.

**Details**
Press "Click for details" to bring up a pop-up window with additional information about each of the machine learning models. Note that the amount of model details depends on whether the project is for single-class or multi-class classification.

**Multi-class classification**

**UMAP and PCA Plots**
In model details we are showing the dimensionality reduction UMAP and PCA plots as visual indications of how the training datasets are "clustered" in the given model.
Confusion Matrix
Matrix representation of True Labels and Predicted Labels.
Diagonal (upper left to lower right) elements indicates instances correctly classified. Off-diagonal elements indicate instances mis-classified. Summing instances over each row should sum to total instances for the respective class.

Cross Validation: By-fold Accuracies vs Classes
Visual representation of the spread of classification accuracies across the CV folds. This representation is done by-class. If the by-fold points are all shown close to the mean line, this shows that the average by-class accuracy is a precise measurement of how well the model should perform.
for the given class. More variance in the by-fold points suggests that the model may perform much better or much worse than expected.

**Learning Curves**
Learning Curves illustrate the performance for each class at different number of instances of data collected/uploaded. Each point on the Learning Curve is the cross-validation accuracy at the respective data size. This gives an understanding of whether adding more data will help to improve the classification performance for each class and whether similar performance can be achieved with fewer instances of data.

**RoC Curves**
RoC Curves plot the False Positive Rate (FPR, x-axis) vs. True Positive Rate (TPR, y-axis) for each class in the classification problem. The dotted line indicates flip-of-the coin performance where the model has no discriminative capacity to distinguish between 2 classes. The greater the area under the curve (AUC), the better the model.

**Matthew's Correlation Coefficient**
The Matthew's Correlation Coefficient (MCC) is a measure of discriminative power for binary classifiers. In the multi-class classification case, it can help show you which combinations of classes are the least well understood by your model. The values can range between -1 and 1, although most often in AutoML the values will be between 0 and 1. A value of 0 means that your model is not able to distinguish between the given pair of classes at all, and a value of 1 means that your model can perfectly make this distinction.

There will be one MCC value for every pair of class labels in the datasets (order does not matter). For example, there will be 3 coefficients for each combination of the 3 class labels, and 6 coefficients for 4 class labels.

**Single-class classification**

**Confusion Matrix**
For the single-class classification case, we only have data from the one given class. A perfect confusion matrix for single-class models has all of the cases concentrated in the top-left corner, meaning that none of the given class data was classified as not coming from that class.

**Cross Validation: By-fold Accuracies vs Classes**
Similar to the confusion matrix case, the most important information in the single-class classification by-fold results are the left-most case. This will show us how varied our single-class accuracies were for each fold of our cross validation.

**Matthew's Correlation Coefficient**
For single-class classification, there is only one Matthew's Correlation Coefficient, which measures the quality of the classification between the given class and things that do not belong to the given class. The values can range between -1 and 1, although most often in AutoML the values will be between 0 and 1. A value of 0 means that your model is not able to recognize the given class at all, and a value of 1 means that your model can perfectly make this distinction.
Sensitivity Analysis
Trade off model accuracy between classes to find a balanced performance that is right for your use case.

Save
Download the model as a binary image to your machine.

Flash to Hardware
Flashes the model to the Target Hardware. Target Hardware must be connected.

Live Test
Once the model has been flashed to Target Hardware, "Test" becomes clickable, and will take you to Live Classification screen.

Delete
When a model is no longer required, you can delete it. A confirmation dialog box will be presented.

Live Classification
Here you can perform live-testing. The screen will display the current class that is predicted by the model that was flashed to the Target Hardware, based on the signals from the enabled sensors for this Project.

Classification Methods

Continuous Classification
Continuous classification is selected on the live test page as default.
In continuous classification, the live test screen will display the classification result one after another in a rate you set previously in classification interval. Continuous classification is recommended when the training sets are all continuous data.

Event Classification (Start/Stop)
In event classification, the model will only read the sensor input within the event window, and give one prediction after each event.
Event classification is recommended when the training set contains event data.

Using GUI button: Press START and begin performa the event. The prediction result will display on the screen after the event ends by either pressing STOP or the n second of event length is up.
Using hardware button:

AutoML also supports event classification using hardware button for some target hardware. Press the button* on the device to begin the event. The prediction result will display on the screen after the event ends either by releasing the button or the n second of event length is up.

*Current target hardware that supports hardware event classification:

ST SensorTile.Box:
Arduino BLE sense and Arduino IoT sense using Qeexo custom case:
Contact us and get your own Arduino battery case.

**Connection Methods**

**Choose between USB or Bluetooth**

If your hardware supports more than one connection method, select the option which works for your use case:
AutoML supports both USB and Bluetooth for most Target Hardware.

Hardware Connection

When you press CONNECT for Bluetooth, a screen similar to the following will be presented:
There may be more than one device ready to connect. If so, please choose the one starting with "QX-Nano" to pair with your device.

When you press CONNECT for USB, Live Classification will be conducted via the connected USB cable and no further connection action is required.

For Single-class classification, USB is the only connection method. This is due to limitations for the Manual Calibration option (described below).

**Reading Inference from Hardware**

**Multi-class Classification Project**

The machine learning model during live classification outputs two pieces of information: the current predicted class label and the current model output values (called "Probability" in the webapp) for each class.
The current predicted class label is based on the class with the highest model output value. Some weighting may also be applied for certain problems, so there may appear to be a minor delay between the maximum class "Probability" and the predicted class label.

UPDOWN

<table>
<thead>
<tr>
<th>CLASS LABEL</th>
<th>PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDOWN</td>
<td>0.46</td>
</tr>
<tr>
<td>WAVE</td>
<td>0.32</td>
</tr>
<tr>
<td>IDLE</td>
<td>0.22</td>
</tr>
</tbody>
</table>

The probability table of all class labels are presented as reference.

**Single-class Classification Project**

For single-class classification, the machine learning model outputs the same two pieces of information. In this case, there will always only be two classes displayed: the given class and the anomaly (i.e. "NON_*") class.

NON_IDLE

<table>
<thead>
<tr>
<th>CLASS LABEL</th>
<th>PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON_IDLE</td>
<td>0.63</td>
</tr>
<tr>
<td>IDLE</td>
<td>0.37</td>
</tr>
</tbody>
</table>

After your single-class model is trained, it may be beneficial to calibrate the threshold of your model to tailor-fit your use case and application scenario. Begin the calibration process by clicking "MANUAL CALIBRATION".
Scores for single-class classifiers are in the range $(0, 1]$, i.e., $0 < \text{Score} \leq 1$. Whether the given instance belongs to the given class or the anomaly class is determined by thresholding the Score, as shown below:

```python
if \text{Score} < \text{threshold}:
    \text{Signal is Normal}
else:
    \text{Signal is Abnormal (i.e. Anomaly)}
```

In the Manual Calibration menu, you can see scores (the blue line) generated by the single-class model plotted over time. Qeexo AutoML recommends an initial threshold based on an analysis of the training data, which is shown by the dotted red line. Users can change the threshold and flash the new, manually-selected threshold to the embedded target. The most recent user-selected threshold is shown in the plot by the dotted green line.

**Manual Calibration**

General rules of thumb for threshold tuning is the following:

- If you want to be very sensitive to potential anomalies, you should set the threshold to be lower. That means that even small variations in the sensor data away from the collected operating condition would trigger an anomaly. However, please beware that setting the threshold too close to zero may result in too many false positives.
- Likewise, if you only want to detect obviously anomalous data, you should keep the threshold to be higher. That means that minor variations in the sensor data away from the collected operating condition may not trigger anomalies.
Please find right balance by doing live classification experiments through live classification after flashing a range of thresholds to the embedded device.

Live Classification Analysis

Sensitivity Analysis

For multi-class classification, the Sensitivity Analysis tool allows you to trade off accuracy between classes, depending on your specific use-case. You can re-weight the classes in your model and see how the cross-validation accuracies and confusion matrix is affected.

The selected sensitivities are normalized and are used to scale the model output probabilities. Higher values for a given class will make the model more likely to ultimately make a classification of that type.

The easiest way to understand how the Sensitivity Analysis page works is to train a multi-class model and then try a few different values. The accuracy plots and confusion matrix will update in real-time along with your changes to the sensitivities. Notice how the plots change when the sensitivity for the "Punch" class is increased from 1 to 100:
Once you find sensitivity values that seem best for your use-case, press "Save" on the new sensitivity values. This will generate a new binary with your selected values. Click "Select" on the newly-compiled binary, and this updated binary will be the one that is flashed to your device when you go to test live classification.

Note that there is no sensitivity analysis for single-class projects.

**Live-Data Collection and Analysis**

Live-data collection allows users to collect the live data for specific duration, class-by-class and then the subsequent analysis section shows a confusion matrix, ROC curves, Matthews correlation coefficient, and F1 score. Moreover, AutoML estimates the distribution of prediction scores using kernel density estimation (KDE).
KDE plots provide detailed insights into error analysis. The example below is a KDE plot for a gesture recognition problem. All instances are collected as "Gesture2". Ideally, the scores for the other classes should be distributed around zero. However, the mode of "Gesture1" distribution (the blue line) is 0.3, and its tail extends beyond 0.5. These signify potential issues with the live-data or the model used for the analysis. We want to see the distribution of "Gesture1" and "Gesture2" as separate as possible with almost no to little overlap for both the classes. While we can observe that the "Stationary" class is quite well separated from "Gesture1" and "Gesture2". "Stationary" class has a peak around zero and also very narrow compared to "Gesture1" and "Gesture2" very well isolates it.

![KDE plot example]

**Notification Center**

If you wish to be more aware of the current status of your projects, you can go to the Notification Center in which AutoML can provide detailed information of your projects, data, models and training runs.

By simply clicking the bell icon located at the rightmost of the sub-navigation-bar, you will be redirected to the Notification Center.
1. Choose only one microphone but not both.