



# Non-Intrusive Load Monitoring in Smart Grids

Project number: 18-1-1-1618

Final Report

Execution:

308121961 Yuval Mandelbaum

208663328 Rinat Feldhuhn

Advisor:

Dr. Yuval Beck

Tel Aviv University

# Table of Contents

# Table of Figures

# 1. **Abstract**

In today's world, the information given to the consumer from the power grid is limited to the total power that he uses at home. The ability to identify and monitor consumption data of appliances in each household, is not yet available to the average user. It is expected that such information will encourage energy saving behavior, enable improved fault detection, encourage improved energy efficiency incentives, and more.

For instance, researchers in [1] have experimentally found that consumers are able to reduce their energy consumption level by 10-15% within one month if they are provided with a frequent feedback about their energy consumption.

In our project, we will be developing a completely unsupervised Non-Intrusive-Load-Monitoring algorithm, using a smart meter by SATEC (located in the Energy Conversion lab in Tel Aviv University), and multiple data science tools.

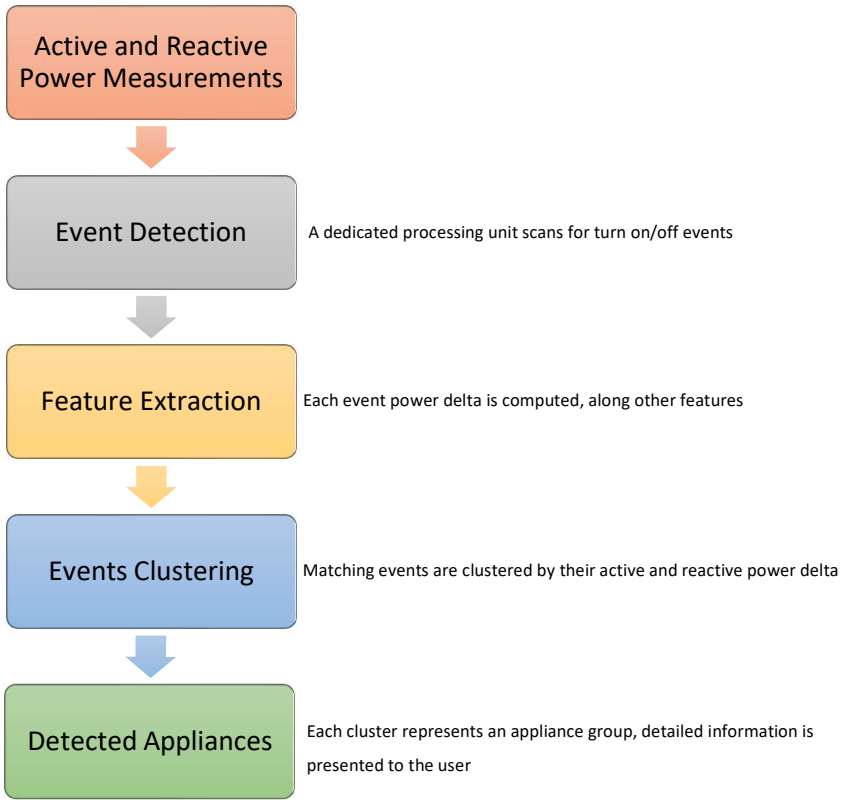


*Figure 1: A block diagram of the proposed unsupervised NILM system. All stages utilize only unsupervised techniques in order to disaggregate the individual appliances from the given real and reactive power signals.*

# 2. Introduction

Electrical loads can be monitored either intrusively or non-intrusively. In the intrusive approach, electrical signals are gathered from each appliance separately.
The Non-Intrusive-Load-Monitoring (NILM), gathers signals of simultaneously operating appliances. The signals are then disaggregated to infer information about individual appliances. With NILM, we save money by reducing the hardware needed to monitor our power consumption, and we eliminate the need for multiple metering points.

The initial concept of data decomposition for general power consumption for different loads was proposed by George Hart In 1992. He proposed the idea that each electrical device produces an energy "signature" that derives from the measurements (voltage, current, and active/reactive power). According to these measurements, he showed that the signatures allow us to characterize the different appliances. Over the years, substantial research in load identification and power disaggregation have been carried out following Hart’s model

The various NILM techniques can be classified as either low frequency or high frequency. With the latter we can analyze the network with very high resolution, thus identifying with relative ease the signature of the components connected to the network. However, the cost of installing high sample rate meters might be very expensive, and in order to analyze such large amounts of data, additional high-end equipment will be required as well. Therefore, in most cases low frequency sampling is more desirable and practical.

Beside the distinction made on the frequency of data sampling, we can also classify NILM methods into supervised or unsupervised learning algorithms.

With unsupervised learning, the model is trained using only the aggregated data, and no prior training with labeled data is required. In comparison, supervised learning techniques require the individual appliance consumption data, to be used as a training set.

Our project goal is to develop an unsupervised intelligent algorithm that can disaggregate total facility's energy data down to individual appliances via non-intrusive load monitoring. That way we can eliminate the need to attach a dedicated device to monitor each and every electrical component in said household.

# 3. **Theoretical background**

In our project we are going to use an algorithm in order to group incoming data samples to identify which appliances are connected to our grid. This method is called clustering, and we will base our project upon this idea.

In computer science, data stream clustering is widely used to cluster data that arrives continuously such as telephone records, multimedia data, financial transactions, search queries, etc. The objective is, given a sequence of points, to construct a good clustering of the stream, using a small amount of memory and time.

There are many clustering algorithms to choose from to achieve our goal, and the algorithm we chose is BIRCH.

BIRCH (balanced iterative reducing and clustering using hierarchies) is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets.

It constructs a tree data structure with the cluster centroids being read off the leaf.

The tree data structure consists of nodes with each node comprised of a number of subclusters. The maximum number of subclusters in a node is determined by the branching factor (a parameter of our choosing).

Each sub-cluster maintains a linear sum, squared sum and the number of samples in that subcluster. In addition, each subcluster can also have a node as its child, if the subcluster is not a member of a leaf node.

Every new sample is inserted into the root of the Clustering Feature Tree. It is then clubbed together with the subcluster that has the centroid closest to the new sample. The linear sum, squared sum and the number of samples of that subcluster are then updated. This process is done recursively until the properties of the leaf node are updated.

Calculations with the Cluster Features:

Given $CF = [N, \overrightarrow{LS}, \overrightarrow{SS}]$ only the the same measures can be calculated without the knowledge of the underlying actual values.

- Centroid: $\overrightarrow{C} = \frac{\sum_{i=1}^{N} \overrightarrow{X_i}}{N} = \frac{\overrightarrow{LS}}{N}$
- Radius: $R = \sqrt{\frac{\sum_{i=1}^{N} (\overrightarrow{X_i} - \overrightarrow{C})^2}{N}} = \sqrt{\frac{N \cdot \overrightarrow{C}^2 + \overrightarrow{SS} - 2 \cdot \overrightarrow{C} \cdot \overrightarrow{LS}}{N}} = \sqrt{\frac{\overrightarrow{SS}}{N} - (\frac{\overrightarrow{LS}}{N})^2}$
- Average Linkage Distance between clusters

  $CF_1 = [N_1, \overrightarrow{LS_1}, \overrightarrow{SS_1}]$ and $CF_2 = [N_2, \overrightarrow{LS_2}, \overrightarrow{SS_2}]$:

$$D_2 = \sqrt{\frac{\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} (\overrightarrow{X_i} - \overrightarrow{Y_j})^2}{N_1 \cdot N_2}} = \sqrt{\frac{N_1 \cdot \overrightarrow{SS_2} + N_2 \cdot \overrightarrow{SS_1} - 2 \cdot \overrightarrow{LS_1} \cdot \overrightarrow{LS_2}}{N_1 \cdot N_2}}$$

In multidimensional cases the square root should be replaced with a suitable norm.

# 4. **Preplanning**

## 4.1 **Thought Process:**

Our project goal has always been to identify which devices are connected to the grid and present an accurate summary of the energy usage to the users.

The smart meter we are using is able to sample the total complex power of the grid, and we base our identification algorithm on the basis of interpreting the changes in complex power.

Different appliances have different complex power signatures. Complex Power is comprised of Active and Reactive components.

We express complex power as: $\boldsymbol{S = P + JQ}$ Where $P$ and $Q$ denote active and reactive power, and $|S|$ is the magnitude of the complex power, and is named apparent power.

We can plot the samples of $S$ on the cartesian $PQ$ plane at the given sampling rate in order to see trends in power usage. At first, we assumed that analyzing the steady state of the complex power will be enough to identify appliances that are connected to the grid, but in most cases electrical appliances work simultaneously and therefore we cannot infer from the total complex power which device is connected at any given moment.

After realizing the above, the project shifted focus into analyzing the transition between steady states, which we called “events”. The shift in focus allowed us to gather more information about the different appliances that are connected to the grid and also to identify which type of device is connected in real time.

When we have enough events shown on the PQ plane, we will have to use a clustering algorithm in order to classify events to their matching appliance group. There are many different clustering algorithms to choose from, and we tested some of them to see which one is most suitable.

### 4.2 Comparison with other algorithms:

The first clustering algorithm we tested was K-Means. The K-Means algorithm is one of the most popular clustering algorithms in current use as it is relatively fast yet simple to understand and deploy in practice. Nevertheless, its use entails certain restrictive assumptions about the data we will be processing. Mainly, we need to provide the number of clusters (K) the algorithm will converge to. Since one of the guidelines to our project is to be non-supervised, and because we do not know the number of appliances connected at any given moment, that presented a real issue.

Additionally, K-means works well when the shape of the clusters is circular. If the natural clusters occurring in the dataset are non-circular then K-Means will not work effectively. Another disadvantage is that K-means starts with a random choice of cluster centers, therefore it may yield different clustering results on different runs of the algorithm. Thus, the results may not be repeatable and lack consistency. However, with hierarchical clustering, we will definitely achieve the same results independent of the number of runs.

The second algorithm we tested was Mean-Shift-Clustering. This algorithm, unlike K-Means, does not impose any pre-assumptions on the received dataset. That means it is suitable for non-supervised data clustering. However, Mean-Shift is less effective over very large datasets, and does not adequately consider the case wherein a dataset was too large to fit in memory. Furthermore, Mean-Shift, like many other clustering algorithms inspects all data points(or all currently existing clusters) equally for each “clustering decision”, and does not perform heuristic weighting based on the distance between these data points, that makes it very computationally intensive. To make matters worse, it also repeats the same computations for every run, and so it does not suit our pre-requisite of real-time online clustering.

The third and final algorithm we tested was BIRCH clustering. An advantage of BIRCH over the previous algorithms is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points to produce the best quality clustering for a given set of resources (memory and time constraints). In most cases, BIRCH only requires a single scan of the database. It is also completely unsupervised.

We have not seen any preceding work in regards to incorporating BIRCH with NILM. We assume that the reasons are:

- BIRCH is quite novel, it was invented in 1996 and was not implemented in popular data science tools until very recently. For example, Scikit-Learn, which is the most popular and extensive library for machine learning in python, implemented BIRCH only in 2014.
- BIRCH has many parameters, which makes it harder to deploy and also complicates scalability since it is harder to tweak.

### 4.3 **Guidelines:**

We followed the guidelines and assumptions below in order to achieve our project goal:

- The algorithm's input will be based solely on active and reactive power. (although it remains expandable, to be discussed in Section 7:Summary & Conclusions)
- We will be feeding information from only one of three phases in our grid.
- The learning method should be non-supervised.
- The identification process will present results in real time.
- The algorithm should be stable, non-memory-intensive, and will be able to handle a large scale of appliances.
- The algorithm in its current form should identify appliances with constant, stable, and non-variant energy signatures.
- The sampling rate we will be working with is 1Hz.
- No two turn on/off events will occur in an interval smaller than the sampling time. Therefore, we will not toggle two devices in less than one second.

# 5. Implementation:

## 5.1 Hardware Implementation

Our algorithm requires accurate measurements of the total active and reactive power of the facility we will be analyzing. In our project we used a smart meter by SATEC that measures a vast number of electrical parameters digitally.

The SATEC smart meter we have been working with in our Tel-Aviv University energy lab has two digital samplers working simultaneously. The first is a high-frequency sampler, that provides a high-resolution waveform of the power, voltage and current signals. The second is a low-frequency (1 Hz) sampler that measures total active and reactive power amongst many other attributes. This sampler will be our primary instrument to feed data to our algorithm.


*Figure 2: The SATEC PM135EH smart meter. The primary measuring device in our lab.*

We must note that the SATEC smart meter is not required for our algorithm to perform properly, and there are much cheaper and easier to implement low-frequency smart meters to choose from.

The measurements taken from the meter were processed by a windows-based computer and fed to the python script we wrote in order to achieve results.

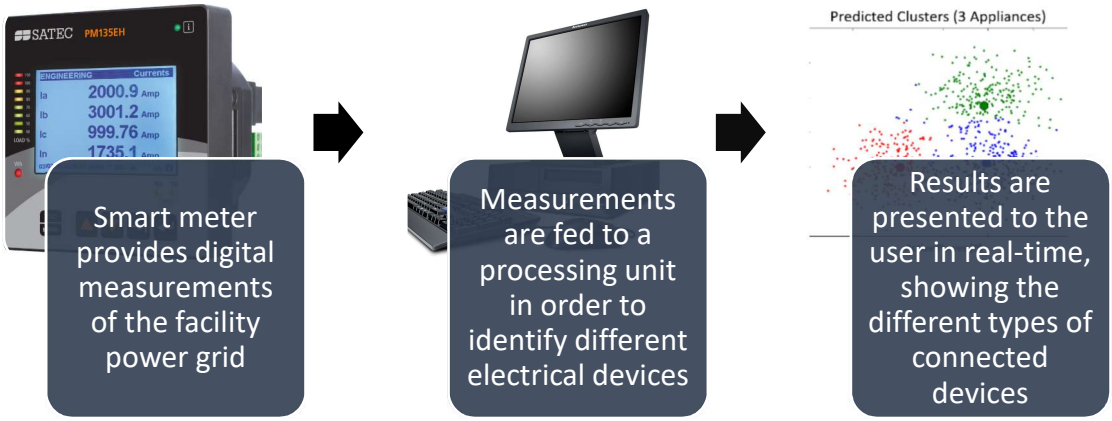


*Figure 3: A block diagram of the hardware we used in order to implement our algorithm.*

## 5.2 Software Implementation

The algorithm is comprised of four different stages:

Stage 1: Event Detection:

As mentioned in the previous section, events are time-based objects that involve a transition between one steady state to another in the active power signal. They can be attributed to an appliance turning on or off.

In event detection, the power signal is segmented into transient sections. Since features are going to be extracted from the transient sections, two constraints on the event detection algorithm have to be satisfied.

First, it has to be a change interval detector. Thus detecting the interval of transition between steady states and not the changepoint.

Second, it has to preserve all characteristics in the transient sections. Therefore, we will not use any filtering technique on our power signal before injecting it into our algorithm (We also assume the signal to be relatively stable, since it was sampled from a low frequency smart meter).

We analyze the incoming power signal by examining a group of samples in each iteration. The size of the group is free for us to determine, and will affect heavily on the performance of the identification process. If we choose a large group, the precision of classification will increase with more samples to analyze. Unfortunately, latency between an event occurrence to a user-prompt will increase as well.

If we choose a group of samples which is too small, the latency will reduce, but we will not be able to classify events with only a few samples. Consequently, we will have to compromise.

After determining the group size, we use numerical differentiation on that group of samples to scan for changes in active power. We will now have to choose a threshold in order to differentiate between a change in power due to an appliance turning on or off, to a change caused by a normal fluctuation in an already active appliance. After choosing the threshold, we filter out the anomalies and receive a vector that consists of changepoints, denoted $t_s$.

Those will be the corner-stones of the events to be constructed. The change-intervals will be decided in the next phase.

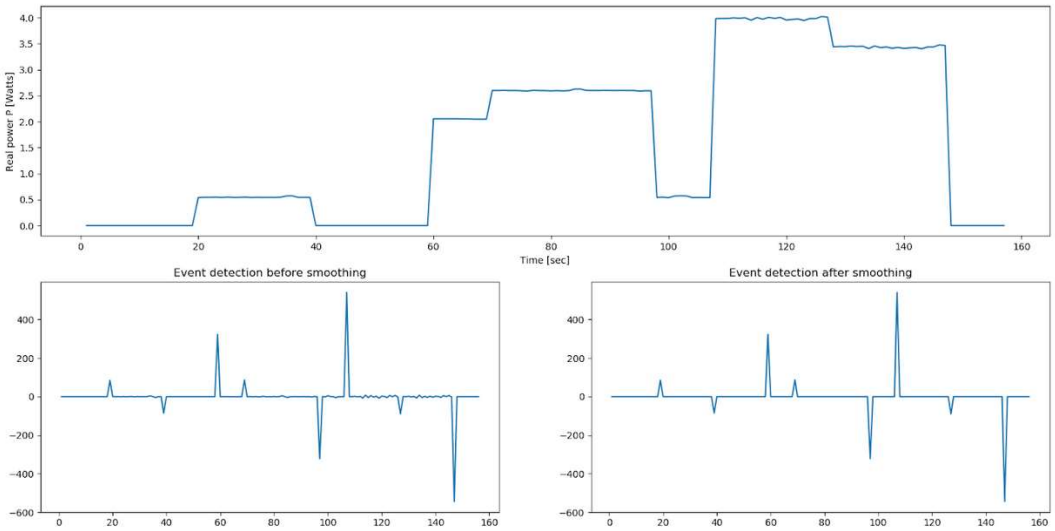


*Figure 4: Graphs showing the event detection process. At the top we see the electrical power signal that is fed to our system. Bottom Left: The resulting power signal after numerical differentiation. Bottom Right: the result has been smoothen in order to filter out anomalies or noise.*

Stage 2: Feature Extraction:

After detecting an event we will want to analyze the unique features that will help us classify this event to its appliance group. The main feature that will differentiate between appliances is the power change $\Delta\psi$. The change is computed using both the real $P$ and reactive $Q$ power signals. To calculate the power delta we will also want to choose an interval before and after the power spike to extract the samples and view them on the $PQ$ plane.

The time interval will be computed as follows: $\Delta T_i = \min \begin{cases} \left|t_s^i - t_s^{i-1}\right| \\ \left|t_s^i - t_s^{i+1}\right| \\ Const \end{cases}$

$t_s^i$ denotes the time of occurrence of the current processed event. And the constant is chosen to limit the number of samples we will collect. In our project we will choose 50 as the limit, as we found out that intervals that are longer may cause memory overload and are superfluous. $\Delta\psi$ will be computed as follows:

$$\Delta\psi_i[k] = \begin{cases} \Delta P_i[k] = P_i\left[t_s^i + k\right] - P_i\left[t_s^i - \Delta T_i + k\right] \\ \Delta Q_i[k] = Q_i\left[t_s^i + k\right] - Q_i\left[t_s^i - \Delta T_i + k\right] \end{cases} for\ k = 0,1,2,3\ \ldots$$

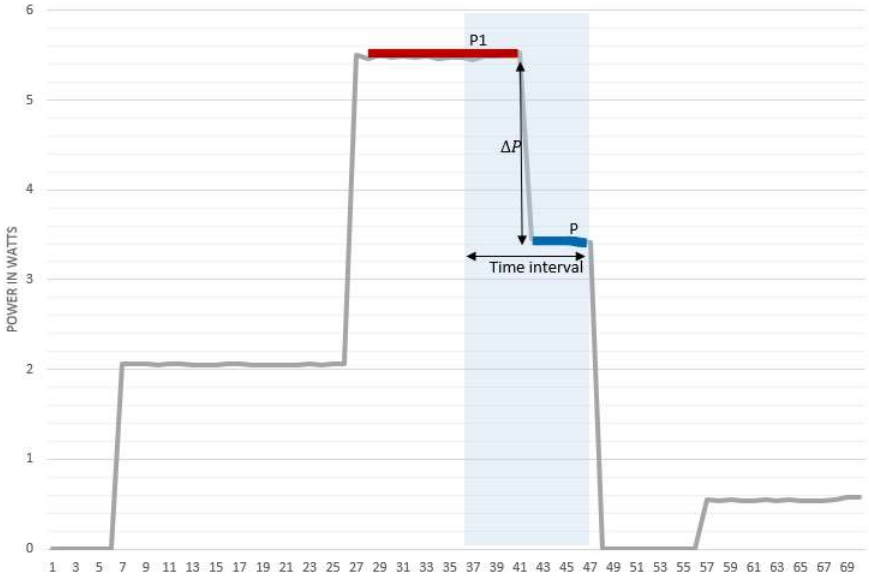


*Figure 5: Power amplitude delta calculation of the active signal. We can observe that the steady state marked with blue was chosen as the minimum interval.*

We will determine whether the event is linked to an appliance being turned on or off depending on the sign of $\Delta P$. If an event increased the total active power of the grid then it has been turned on.

Similarly, reducing the total active power means an appliance has been turned off. We can not determine the toggle mode by looking at the sign of $\Delta Q$ since capacitive loads will reduce the total reactive power while turned on.

We will save the time of occurrence $t_s$ as a property of the event, for future analytics e.g. average active time, or toggle frequency. (to be further discussed in Stage 4: Result Analysis)

We will now present the $\Delta\psi$ of multiple events on the PQ plane, and cluster them together in order to classify each event to its matching appliance group.

## Stage 3: Clustering Events:

In the clustering stage, events are grouped in separate clusters according to their extracted features. Since the number of the underlying appliances is not known in advance, the clustering algorithm has to be non-parametric. This limits the usage of all clustering algorithms that are based on function optimization since they assume that the model order is known in advance or require an estimation.

The utilized clustering algorithm is BIRCH clustering. In the initial run, when separate clusters are not yet formed, the algorithm will look for new events in order to "learn" the grid. When a new cluster is formed, the algorithm will prompt the user in order to label that cluster, from now on each future event that belongs to that cluster represent one type of appliance turning on or off.

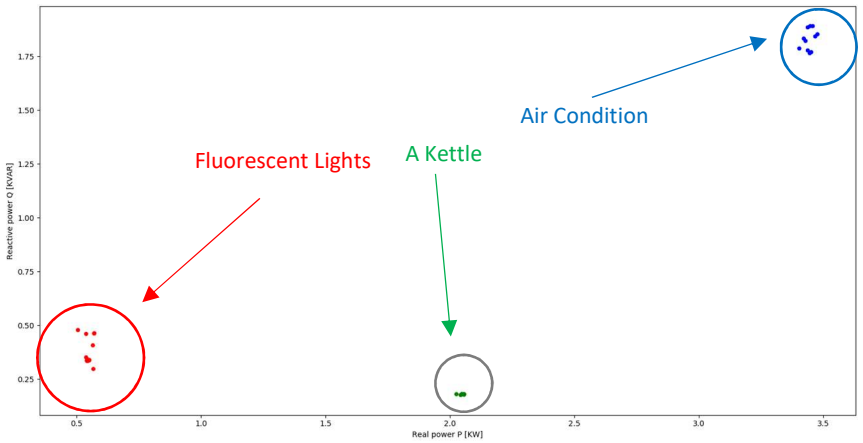


*Figure 6: The clustering algorithm in action. We can see three distinct groups of appliances.*

Stage 4: Result Analysis:

After the algorithm has been deployed for a long enough time, we will have a complete picture of all the appliances connected to the grid in real-time. We can now analyze all of the events in memory in order to calculate electricity costs, figure out which appliances have been used frequently, and notify the user.

In order to calculate cost per appliance, the algorithm looks for matching on and off events from that particular cluster to calculate the total time the appliance has been turned on: $T_{on} = \sum_{i=0}^{n} t_s^i - t_s^{i+1}$ , this will also be able to tell us how frequent we use each appliance in our household.

To calculate cost, we will need to have the average power consumption of said appliance. We will calculate it from the collection of $\Delta P$ samples in each event. We will then multiply $T_{on} \cdot P_{avg}$ in order to achieve $W$, total energy consumed per appliance. Finally, we will multiple $W$ by the cost of electricity for households to get the total cost for a specific appliance.

# 6. Simulation

## 6.1 Working Environment:

We began evaluating our algorithm by fetching measurement data from the SATEC smart meter in Tel-Aviv University energy lab. We simulated a normal functioning household by turning on and off everyday appliances such as fluorescent ceiling lights, a kettle, an air condition unit, a vacuum and an iron.

To simulate a real-time data stream, our algorithm read incoming information with pre-determined time intervals of 20 seconds.
For each time interval the algorithm found the different events and also computed the power delta for each detected event in said interval.
These deltas were processed by the BIRCH algorithm, that clustered them in real-time.

## 6.2 Performing Measurements:

The meter provided us with measurements of total active and reactive power of the grid. As we can see in the following graphs:

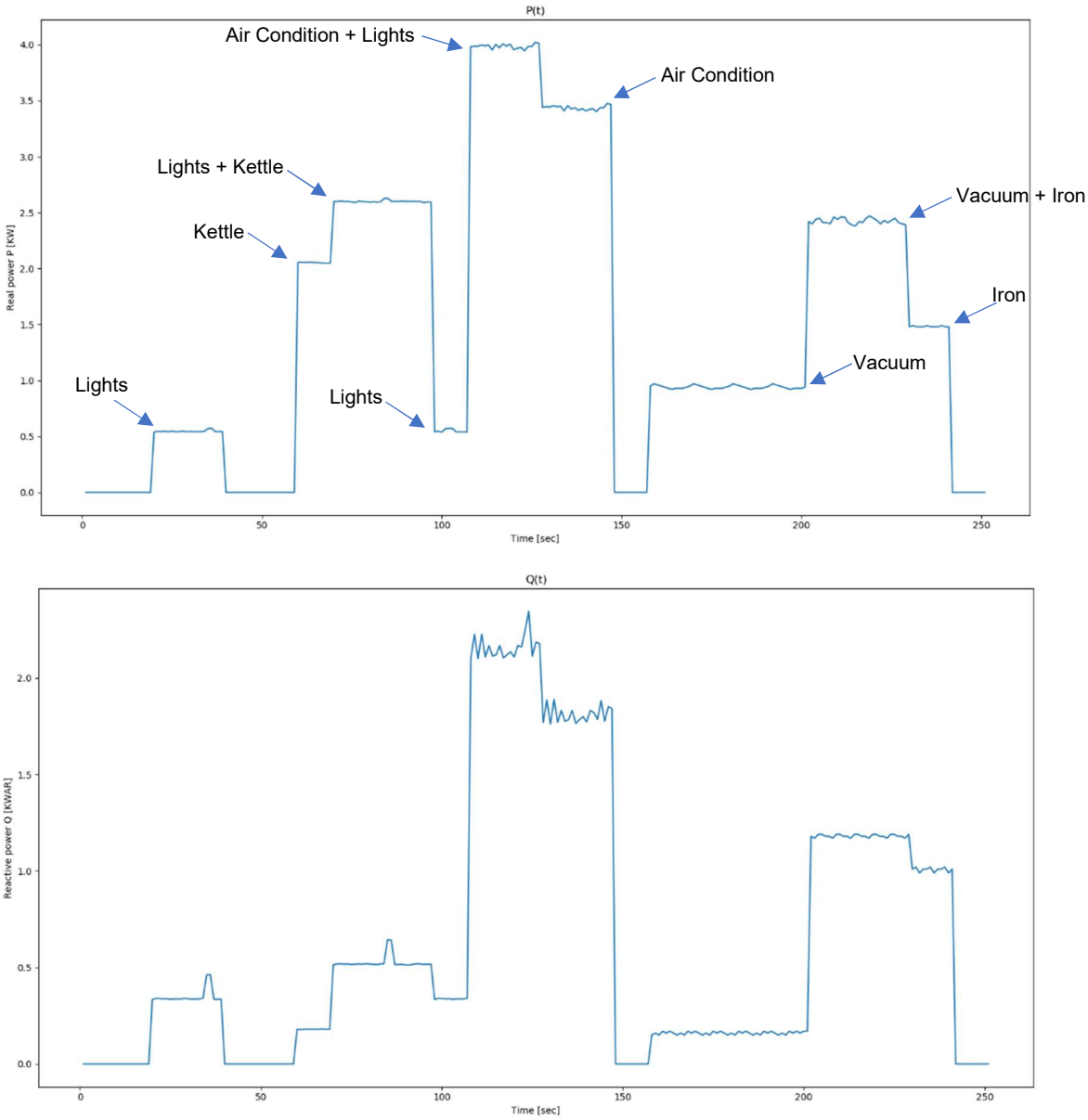


*Figure 7: A graph of the active and reactive power signals that were fed to our algorithm during simulation. Each steady state is a measurement of an individual appliance or a superposition of multiple appliances.*

### 6.3 Event Detection and Clustering

After acquiring the measurements from the meter and feeding them to our algorithm, we successfully extracted the turn on and off events, and matched the events to their appropriate cluster as we can see from the console output:

```
NEW DEVICE DETECTED at 20: Please type name:Lights
The 1 event is a On-Type event that occurred at 20.It belongs to cluster #Lights
The 2 event is a Off-Type event that occurred at 40.It belongs to cluster #Lights
NEW DEVICE DETECTED at 60: Please type name:Kettle
The 3 event is a On-Type event that occurred at 60.It belongs to cluster #Kettle
The 4 event is a On-Type event that occurred at 70.It belongs to cluster #Lights
The 5 event is a Off-Type event that occurred at 98.It belongs to cluster #Kettle
NEW DEVICE DETECTED at 108: Please type name:Air Condition
The 6 event is a On-Type event that occurred at 108.It belongs to cluster #Air Condition
The 7 event is a Off-Type event that occurred at 128.It belongs to cluster #Lights
The 8 event is a Off-Type event that occurred at 148.It belongs to cluster #Air Condition
NEW DEVICE DETECTED at 158: Please type name:Vacuum
The 9 event is a On-Type event that occurred at 158.It belongs to cluster #Vacuum
NEW DEVICE DETECTED at 202: Please type name:Iron
The 10 event is a On-Type event that occurred at 202.It belongs to cluster #Iron
The 11 event is a Off-Type event that occurred at 225.It belongs to cluster #Vacuum
The 12 event is a Off-Type event that occurred at 237.It belongs to cluster #Iron
```

*Figure 8: Console output during simulation, describing the different appliances found. In green: Appliance labeling during a "NEW DEVICE DETECTED" prompt.*

### 6.4 Clustering Incoming Measurements

The algorithm will continue scanning for incoming turn on/off events. For each new detected event, the algorithm will try to predict its matching cluster. If a new cluster is formed – the system will prompt the user in order to label the new device. As time goes by the system will learn all devices connected to the grid and we will be provided with a clearer picture of the collection of devices at the analyzed facility.

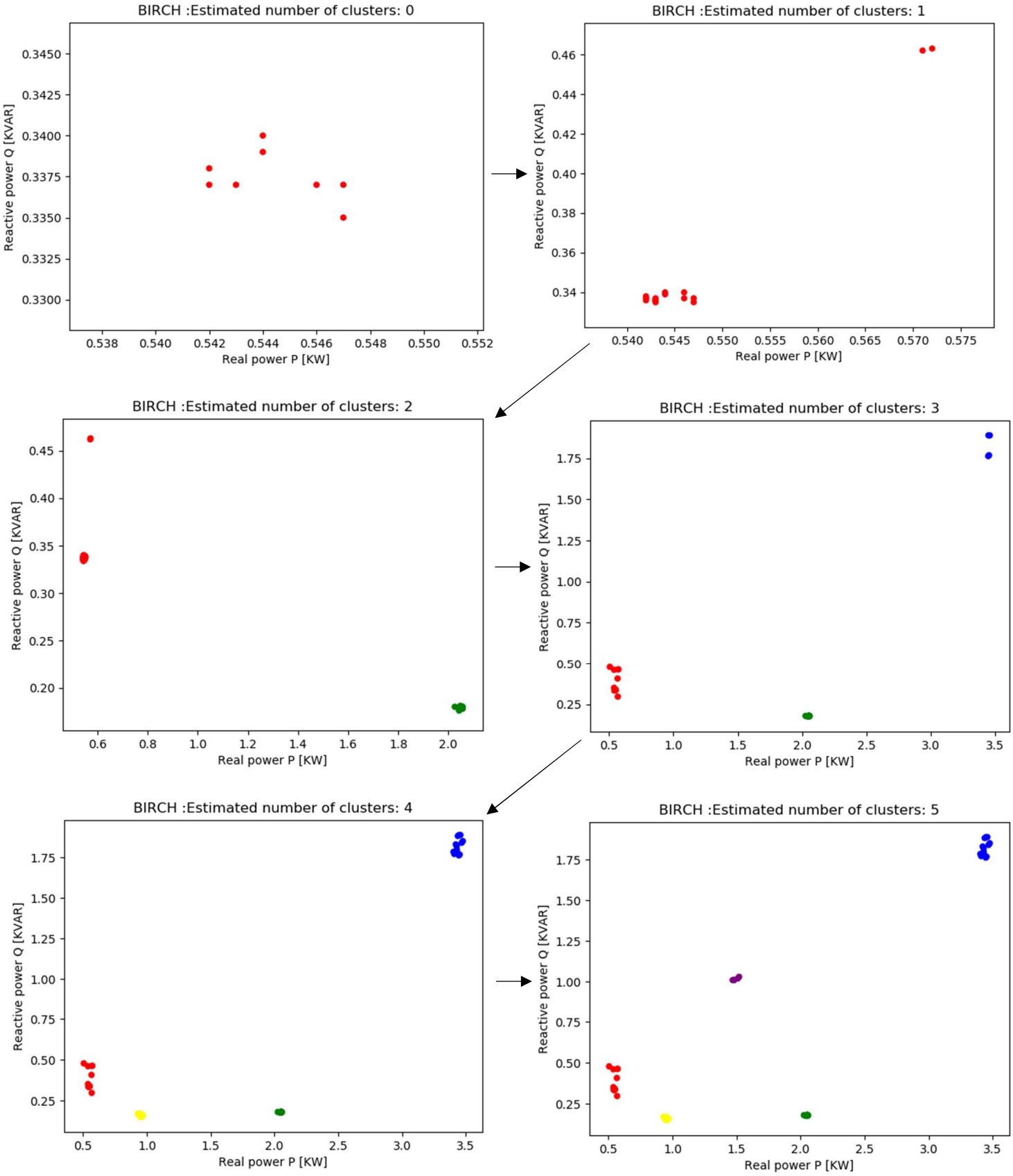


*Figure 9: The clustering process. We can observe how the BIRCH algorithm learns the grid with each iteration over the processed data stream.*

**Notice both axis changing as we add more datapoints.*

## 6.5 Result Analysis:

While the algorithm is running we can query the database for consumer data and statistics, such as:

- Total cost and total energy usage:

```
What would you like to do next?
1)Check total cost and total energy usage
2)Check specific appliance usage
3)Check state of all appliance
4) exit
1
Total usage until 251 sec is 0.1043KWh. the total cost is 0.0492
```

- Usage statistics for a specified appliance group:

```
What would you like to do next?
1)Check total cost and total energy usage
2)Check specific appliance usage
3)Check state of all appliance
4) exit
2
statistic on which device would you like to see?
Lights
Lights used 0.01182KWh (11.33% of total power usage
```

- Real-time state of each appliance connected to our grid:

```
What would you like to do next?
1)Check total cost and total energy usage
2)Check specific appliance usage
3)Check state of all appliance
4) exit
3
Lights - Off
Kettle - Off
Air Condition - Off
Vacuum - Off
Iron - Off
```

*The screenshots above were taken from the user interface, built-in our python script.

**Results shown simulate an active grid for a period of 250 seconds total. We should assume very low usage.

# 7. **Summary & Conclusions**

At the beginning of the project our goal was to develop an unsupervised intelligent algorithm that can disaggregate total facility's energy data down to individual appliances via non-intrusive load monitoring.

The algorithm at this moment is able to detect, process and analyze data of mostly stable appliances. For best results, the algorithm needs a "learning period" of at least one day with all appliances on. Afterwards, appliance detection will be nearly instant. We also created a user interface at which the user can get useful information and learn about his/her power consumption habits.

While writing the code we emphasized the importance of runtime. We understood that it is important for the algorithm to run in real time in order for the code to be applicable. For this purpose, we insisted on optimizing the runtime of the algorithm by choosing the appropriate clustering technique. Another challenge we faced was choosing the scanned time intervals. Long time intervals will increase runtime and latency, on the other hand, short intervals affect the accuracy of the algorithm negatively.
Lastly, we tried to optimize BIRCH by tweaking its parameters in order to get the best results with minimum runtime.

We believe that most of our goals have been achieved, although there is still more work to be done before the software can be fully integrated into the industry.

In the future, we believe our project can serve as a basis for energy monitoring applications, and there are many aspects for the software to be improved:

- First, the clustering algorithm (BIRCH) is flexible, it can expand to multiple dimensions thus cluster the samples more precisely. For example, we can classify the appliances by their harmonic features which can vary vastly from one device to another. With expandability to multiple dimensions, the algorithm will be able to differentiate between two appliances with very similar energy signatures.

- With machine learning algorithms, the platform will be able to also disaggregate appliances with continuously variable energy signatures. In addition, as time continues the system will learn the power consumption patterns, thus achieve more accurate and reliable results.

.

# 8. **Sources**


[1] Barsim, K.S., Streubel, R. and Yang, B., Unsupervised Non-Intrusive Load Monitoring of Residential Appliances.

[2] R. A. Winett, M. S. Neale, and H. Cannon Grier, “Effects of self-monitoring and feedback on residential electricity consumption,” J. Applied Behavior Analysis, 1979.

[3] *Zhang, T.; Ramakrishnan, R.; Livny, M. (1996). "BIRCH: an efficient data clustering method for very large databases". Proceedings of the 1996 ACM SIGMOD international conference on Management of data - SIGMOD '96. pp. 103–114.*

[4] Modified Cross-Entropy Method for Classification of Events in NILM Systems, Ram Machlev, Student Member, Yoash Levron, *Member*, and Yuval Beck, *Member*.

[5] Chinthaka Dinesh, Pramuditha Perera 2, Roshan Indika Godaliyadda 1,Mervyn Parakrama B. Ekanayake and Janaka Ekanayake. "Non-intrusive load monitoring based on low frequency active power measurements", Aims Energy, Volume 4, Issue 3, 414–443.

[6] J. Z. Kolter and T. Jaakkola, “Approximate inference in additive factorial 671 HMMs with application to energy disaggregation,” J. Mach Learn. Res., vol. 22, pp. 1472–1482,Mar 2012

[7] https://en.wikipedia.org/wiki/BIRCH

[8] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html

[9] https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html

# 9. **Annex – Python Code**

```
import numpy as np
from sklearn.cluster import Birch
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import math
plt.rcParams.update({'figure.max_open_warning': 0})

# ////////////////////////// Classes and functions
//////////////////////////////////////////////////////////


class Event:  # each event describes the transition between two steady states.
    #Initialize Event
    def __init__(self, active, reactive,ts, interval, time_counter):
        self.active = active
        self.reactive = reactive
        self.ts = ts
        self.toggle = True if active[ts]> active[ts-1] else False
        self.toggle_time = time_counter + ts
        self.deltap = np.zeros(interval)
        self.deltaq = np.zeros(interval)
        self.calc_active_and_reactive_power(active,reactive,ts,interval)
        self.cluster = -1


    def calc_active_and_reactive_power(self,active,reactive,ts,interval):
        # input: the active and reactive power, the time of the change , and the size of the
interval
        # this function calculates the difference is the active and reactive power between the two
steady states

        if interval == 0 | interval == 1 | interval == 2: # if the size of the interval is too
short we fix it by adding more samples to the deltap/q vector
            self.deltap[0] = abs(active[ts] - active[ts-1])
            self.deltaq[0] = abs(reactive[ts] - reactive[ts - 1])
            self.deltap[1] = abs(active[ts] - active[ts-1])
            self.deltaq[1] = abs(reactive[ts] - reactive[ts - 1])
            self.deltap[2] = abs(active[ts] - active[ts - 1])
            self.deltaq[2] = abs(reactive[ts] - reactive[ts - 1])
            self.deltap[3] = abs(active[ts] - active[ts - 1])
            self.deltaq[3] = abs(reactive[ts] - reactive[ts - 1])
        else:
            for i in range(interval):
                self.deltap[i] = abs(active[ts + i] - active[ts - interval + i])
                self.deltaq[i] = abs(reactive[ts + i] - reactive[ts - interval + i])

class Device: # For each device we save: its name and a list of it's on/off events that were
found.
    def __init__(self,name,event):
        self.events = []
        self.name = name
        self.events.append(event)
    def update_events_list(self,event):
        self.events.append(event)

def EventDetection(p_t):
    # This function gets the signal p_t and finds the time of change between each steady state
    # This function returns a vector of all the changes in power
    x, y = np.linspace(0, 1, p_t.size), p_t
    dx, dy = np.diff(x), np.diff(y)
    dir = dy / dx

    for i in range(dir.size): # In order to get better and more accurate results we want to smooth
the signal and get rid of noise
        if abs(dir[i]) < 10:
            dir[i] = 0
    lst = []
    for i in range(dir.size):
        if abs(dir[i]) > 10:
            lst.append(i+1)
    return lst

def ExtractFeatures(p_t,q_t,ts_vec, time_counter):
    # This function gets the signals p and q, the time of the events in the time interval and the
general time counter.
    # This function builds and returns the list of events, each event with it's characteristics.
    events = list()
    for i in range(len(ts_vec)):
        ts = ts_vec[i]
        if i == 0:
            tsprev = 0
        else:
            tsprev = ts_vec[i-1]
```

```
        if i == len(ts_vec) - 1:
            tsnext = len(p_t) - 1
        else:
            tsnext = ts_vec[i+1]
        interval = min(abs(ts - tsprev), abs(ts - tsnext), 50)
        events.append(Event(p_t, q_t, ts, interval,time_counter))

    return events

def new_data_stream_classification(p_t_stream,q_t_stream, ts_vec, X, labels, num_of_clusters,
time_counter):
    # this function analyze the new data stream, creates its list of events and re-fit the new
data to the BIRCH algorithm
    num_clusters = num_of_clusters
    if len(ts_vec) > 0:
        curr_events = ExtractFeatures(p_t_stream, q_t_stream, ts_vec,time_counter)
        deltaP_vec = []
        deltaQ_vec = []
        for event in curr_events:
            deltaP_vec.extend(np.absolute(event.deltap))
            deltaQ_vec.extend(np.absolute(event.deltaq))
        X_new = (deltaP_vec[1:], deltaQ_vec[1:])
        X_new = np.transpose(X_new)
        X = np.concatenate((X, X_new), axis=0) #Add to X Vector the X-Samples found in Data Stream
        brc.fit(X) #Re-Fit the new X Vector to Birch Clustering
        labels = brc.labels_
        num_clusters = len(np.unique(labels))
    else:
        curr_events = []
    return X, labels, num_clusters, curr_events

def CalcCostPerDevice(device):
    # // This function calculates and returns the power usage and cost of a curtain device

    power_usage_in_KWh= 0
    cost_per_KWh =0.4715
    i = 0
    while i < len(device.events) - 1:
        if device.events[i].toggle and (not(device.events[i+1].toggle)):
            time_in_hours = (device.events[i+1].toggle_time - device.events[i].toggle_time)/3600
            power_usage_in_KWh += round(time_in_hours * device.events[i].deltap[1], 5)
            i += 2
    cost = round(power_usage_in_KWh *cost_per_KWh, 4)
    return power_usage_in_KWh, cost

def CalclTotalUsageAndCost(devices):
    #  this function receives the on and off list of appliances.
    #  this function calculates and returns the estimated cost

    total_power_usage_in_KWh = 0
    total_cost = 0
    for device in devices:
        power_usage_in_KWh, cost = CalcCostPerDevice(device)
        total_power_usage_in_KWh += power_usage_in_KWh
        total_cost += cost
    return round(total_power_usage_in_KWh, 4), round(total_cost, 4)


# ////////////////////////// Generating Data
//////////////////////////////////////////////////////////
p_t = np.genfromtxt("KW.csv", delimiter=',', dtype=float)
q_t = np.genfromtxt("KVAR.csv", delimiter=',', dtype=float)

# ////////////////////////////Clustering whole chunk of data - Without
labeling//////////////////////////////////
'''                                          ---OPTIONAL---


events = list()
ts_vec = EventDetection(p_t)
events = ExtractFeatures(p_t, q_t, ts_vec)
deltaP_vec = []
deltaQ_vec = []
for event in events:
    deltaP_vec.extend(np.absolute(event.deltap))
    deltaQ_vec.extend(np.absolute(event.deltaq))
X = (deltaP_vec[1:], deltaQ_vec[1:])
X = np.transpose(X)
brc = Birch(branching_factor=50, compute_labels=True, copy=True, n_clusters=None, threshold=0.5)
brc.fit(X)
color_theme = np.array(['red', 'green', 'blue', 'yellow, black'])
labels = brc.labels_
num_clusters = len(np.unique(labels))
plt.scatter(X[:, 0], X[:, 1], c=color_theme[labels], s=20)
plt.title('BIRCH :Estimated number of clusters: {}'.format(num_clusters))
plt.show()
'''
```

```
# ////////////////////////////// Clustering Data Stream
//////////////////////////////////////////////////////////////
k = 0 #First Plot
time_counter = 0
event_counter =0
devices = []
clusters = []  # Initialize Cluster Array
num_clusters = 0   # Initialize Num of Clusters
startval = 30
jump_val = 21
tskip = 5
events = list()  # Initialize Event List
ts_vec = EventDetection(p_t[0:startval])  # find the on/off events in the time interval
events = ExtractFeatures(p_t[0:startval], q_t[0:startval], ts_vec, time_counter)  # Extract
Features for Events in 30 Samples
deltaP_vec = []
deltaQ_vec = []
for event in events:  # we build a list of all on/off events for furthermore calculations
    deltaP_vec.extend(np.absolute(event.deltap))
    deltaQ_vec.extend(np.absolute(event.deltaq))
X = (deltaP_vec[1:], deltaQ_vec[1:])
X = np.transpose(X)
brc = Birch(branching_factor=50, compute_labels=True, copy=True, n_clusters=None, threshold=0.1)
# Initialize Birch Settings
brc.fit(X)                                  # Fit first 30 samples to Birch Clustering
labels = brc.labels_                       # we save the BIRCH labels for later use ( with printing
the output on the PQ plane)
n_clusters_ = len(np.unique(labels))
color_theme = np.array(['red', 'green', 'blue', 'yellow', 'purple'])
if (len(events) > 0) & (events != []):
    for event in events:
        event_counter += 1
        predict_arr = np.array([event.deltap[1:], event.deltaq[1:]])
        predict_arr = np.transpose(predict_arr)
        predict_vector = brc.predict(predict_arr)    # predict tells us which cluster the said
vector matches up with .
        if predict_vector[0] > (len(devices) - 1):  # if we detected a new device, a message will
be sent to the user to identify it.
            new_device_name = input("NEW DEVICE DETECTED at {}: Please type
name:".format(event.toggle_time))
            clusters.append(new_device_name)
            devices.append(Device(new_device_name, event))
        else:
            devices[predict_vector[0]].update_events_list(event)
        event.cluster = clusters[predict_vector[0]]
        if event.toggle:
            res = "On"
        else:
            res = "Off"
        print("The {} event is a {}-Type event that occurred at {}.It belongs to cluster
#{}".format(event_counter, res,

event.toggle_time,

event.cluster))
time_counter +=startval
# ////////////////////////////////// Start Data Stream Clustering
//////////////////////////////////////////
i = startval
while (i < len(p_t)):
    if i+jump_val > len(p_t):
        jump_val = len(p_t) - i - 1
    plt.figure(k)
    k += 1
    plt.scatter(X[:, 0], X[:, 1], c=color_theme[labels], s=20)  # print all the points on the PQ
plane and paint them according to their cluster
    plt.title('BIRCH :Estimated number of clusters: {}'.format(num_clusters))
    plt.xlabel('Real power P [KW]')
    plt.ylabel('Reactive power Q [KVAR]')
    ts_vec = EventDetection(p_t[i:i + jump_val])  # find the on/off events in the time interval
    if len(ts_vec) == 0:  # if no events were found- continue to the next time interval
        i += jump_val
        time_counter += jump_val
        continue
    flag = 0
    for j in range(len(ts_vec)):
        if (ts_vec[j] < tskip):  #  if an event was found in the very start of an interval or in
the very end of one- adjust the time frame for a more accurate delta vector
            time_counter -= tskip
            ts_vec = EventDetection(p_t[i-tskip:i - tskip + jump_val])

        if (ts_vec[j] > jump_val - tskip):  #  if an event was found in the very start of an
interval or in the very end of one- adjust the time frame for a more accurate delta vector
            jump_val += tskip
            ts_vec = EventDetection(p_t[i:i + jump_val])

    X, labels, num_clusters, temp_event = new_data_stream_classification(p_t[i:i + jump_val],
q_t[i:i + jump_val],ts_vec,X, labels,num_clusters, time_counter)  # collecting and analyzing the
```

```
new data stream

    if (len(temp_event) > 0) & (temp_event != []):
        for event in temp_event:
            event_counter +=1
            predict_arr = np.array([event.deltap[1:], event.deltaq[1:]])
            predict_arr = np.transpose(predict_arr)
            predict_vector = brc.predict(predict_arr)   # predict tells us which cluster the said
vector matches up with .
            if predict_vector[0] > (len(clusters)-1):  # if we detected a new device, a message
will be sent to the user to identify it.
                new_device_name = input("NEW DEVICE DETECTED at {}: Please type
name:".format(event.toggle_time))
                clusters.append(new_device_name)
                devices.append(Device(new_device_name, event))
            else:
                devices[predict_vector[0]].update_events_list(event)
            event.cluster = clusters[predict_vector[0]]
            if event.toggle:
                res = "On"
            else:
                res = "Off"
            print("The {} event is a {}-Type event that occurred at {}.It belongs to cluster
#{}".format(event_counter, res, event.toggle_time, event.cluster))
        events.extend(temp_event)
    time_counter += jump_val
    i += jump_val

print('')
while True:  # after all calculations were done, the user can get useful information about
his/hers consumption habits.
    print('What would you like to do next?')
    print('1)Check total cost and total energy usage')
    print('2)Check specific appliance usage')
    print('3)Check state of all appliance')
    print('4) exit')
    usr_inpt = input()
    if usr_inpt == '1':
        total_usage, total_cost = CalclTotalUsageAndCost(devices)
        print('Total usage until {} sec is {}KWh. the total cost is {}'.format(time_counter,
total_usage, total_cost))
        print('')
    if usr_inpt == '2':
        for i in range(5):
            print('statistic on which device would you like to see?')
            flag = False
            usr_inpt2 = input()
            for device in devices:
                if device.name == usr_inpt2:
                    usage, temp = CalcCostPerDevice(device)
                    flag = True
                    break
            if flag == True:
                print('{} used {}KWh ({}% of total power usage'.format(device.name,
usage,round((usage * 100) / total_usage, 2)))
                print('')
                break
            else:
                print('No match found')
                print('')
                continue
    if usr_inpt == '3':
        for device in devices:
            if device.events[len(device.events)-1].toggle:
                res = "On"
            else:
                res = "Off"
            print('{} - {}'.format(device.name, res))
        print('')
    if usr_inpt == '4':
        break

# ////////////////////////////////////// Plots
//////////////////////////////////////////////////////////////////////////////

# Plot the event detection
fig = plt.figure(k)
k = k+1

gs = gridspec.GridSpec(2, 2, figure=fig)
ax = fig.add_subplot(gs[0, :])
ax.plot(p_t)
ax.set_xlabel('Time [sec]')
ax.set_ylabel('Real power P [KW]')

# Plot the event detection before smoothing
x, y = np.linspace(0, 1, p_t.size), p_t
dx, dy = np.diff(x), np.diff(y)
```

```
dir = dy / dx
ax2 = fig.add_subplot(gs[1, 0])
ax2.plot(dir)
plt.title('Event detection before smoothing')

# Plot the event detection after smoothing
ax3 = fig.add_subplot(gs[1, 1], sharex=ax2, sharey=ax2)
for i in range(dir.size):
    if abs(dir[i]) < 10:
        dir[i] = 0
ax3.plot(dir)
plt.title('Event detection after smoothing')

plt.figure(k)
plt.title("P(t)")
plt.xlabel('Time [sec]')
plt.ylabel('Real power P [KW]')
plt.plot(p_t)

plt.figure(k+1)
plt.title("Q(t)")
plt.xlabel('Time [sec]')
plt.ylabel('Reactive power Q [KWAR]')
plt.plot(q_t)

plt.show()
#
////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////
```