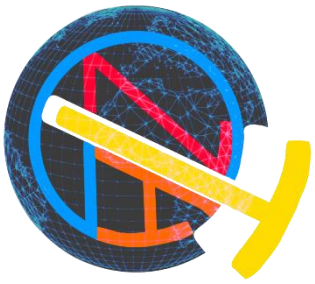
TeamCognito

Proposed Approach for a Smart Meter Network for monitoring power consumption in Smart Cities



Part A General Overview

- TeamCognito will develop an Anomaly Detection and Location Algorithm based on the electric power reading from smart meters installed in residential homes in well-demarcated localities and regions.

- The plan is to establish a network of such smart-meters, with a tree-like hierarchical structure that can be effectively monitored and addressed. The levels of this network hierarchy include the residential home, the locality, and the district in the city region.

- Such a hierarchical network structure can help in localizing deployment of the software systems at the smallest level, in the residential homes, and thereby grant full grant of privacy of electric power consumption data to the residents.

- The algorithm will work in two “protocols”: it will analyse and look for unusual power consumption by collecting and storing residential power consumption data (self-history) and also collaborate with other meters in the same locality to better its’ own detection prowess.

- After the meter hardware, the network and the anomaly detection software is developed, there can be added plans of a centralized district “dashboard” similar to the ones for traffic redirection and control.

- AI can be possible when there is an availability of large repository of multi-dimensional data records. Power consumption data is one-dimensional i.e. they are numbers recorded at time intervals. AI can be employed at a successful and advanced stage of the project deployment when other IOT devices facilitate to collect data and related meta-data (data regarding data) from various power consuming sources in localities and districts.

Part B1 TeamCognito's approach for a well-rounded solution

- Anomaly detection problems for a time series (for example, power consumption data collected at regular timestamps) are usually formulated as finding outlier data points relative to the standard "latent" pattern of a signal hidden with the recorded time series signal. We focus on finding this latent pattern with a business perspective, such as monitor for unexpected spikes, drops, trend changes and level shifts.
- Unexpected spikes are called Additive Outliers. Unexpected decrease or drop to critical zero for a period of time is called temporal drop or change. Series signals can show a change in its total value (amplitude) for a period of time without any significant change in its trend line or pattern: such a change is called a level shift or seasonal level shift.
- An anomaly detection algorithm should either classify each new data point as a probable anomaly if it is, or, forecast a signal pattern for a future data point and then test if the real observed data point deviates too much from the forecasted signal pattern. A confidence interval can help understand and control thresholds for detecting outliers/anomaly data points.
- Exploratory Data Analysis is a pre-requisite step to understand the nature of the data gathered, and can provide us with valuable insights when analysed by a domain expert in Digital Signal Processing, and Electrical Systems.
- Multi-dimensional data can be visualized in lower dimensional spaces using clustering algorithms that perform dimensionality reduction like the T-SNE algorithm.

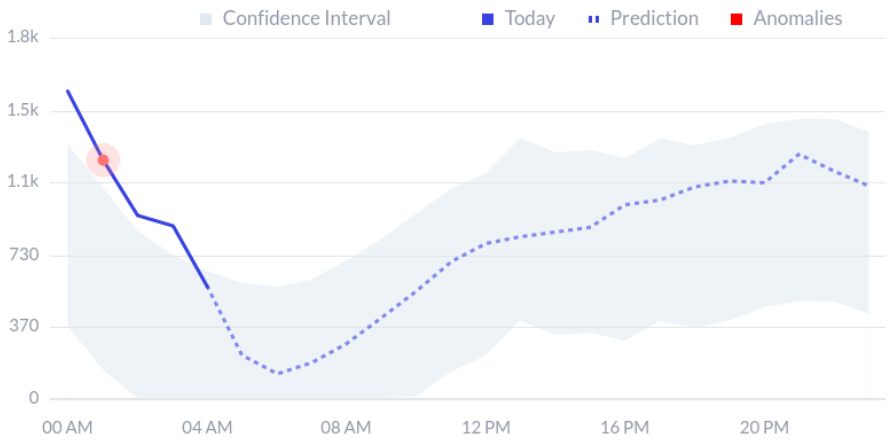
TeamCognito will employ a well-researched and rounded strategy for detecting anomalies in the power consumption, and make sure to minimize the number of false positive or false negative alarms. Some of the well-researched and practically reliable algorithms for anomaly detection are as follows:

i. **Seasonal Trend Decomposition** – STL
ii. **Classification and Regression Trees and the Generalized Extreme Studentized Deviate Test** (GESD Test)
iii. **Auto-regressive Integrated Moving Average** (ARIMA) **Class Model**
iv. **Exponential Smoothing or ARIMA(0,1) Model:** Holt-winters seasonal method
v. **Long Short Term Memory Networks (**LSTM**) for multi-dimensional data**

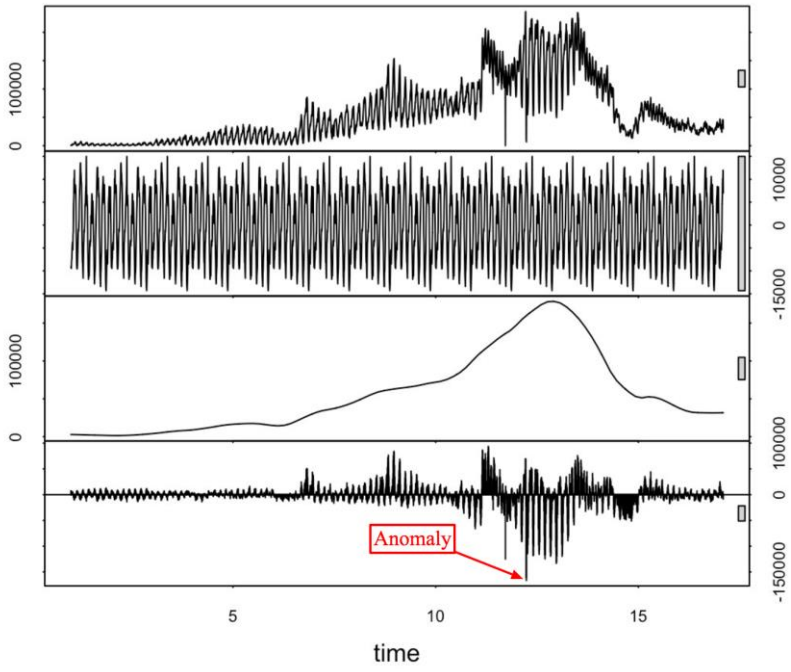
You can find pictorial explanations of these five algorithms and strategies in the next page.

Two prototype and simple algorithms have already been developed and analysed at our R&D department. We believe to always start with the simplest and computationally inexpensive model first, that can give the best results. The process can be complicated and added with more sophistications, if required and if sufficient data and resources can be provided to the R&D Department.

Part B2 TeamCognito's approach for a well-rounded solution

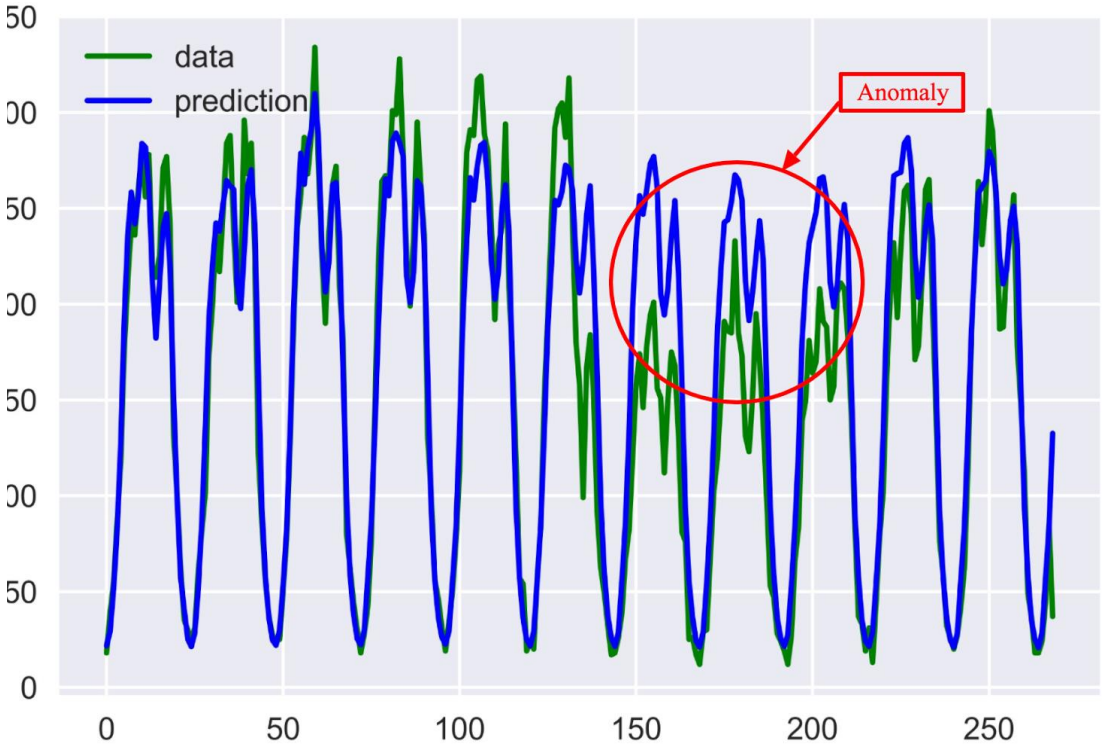


Actual time series, predicted time series and confidence interval help understand why anomaly occurs.

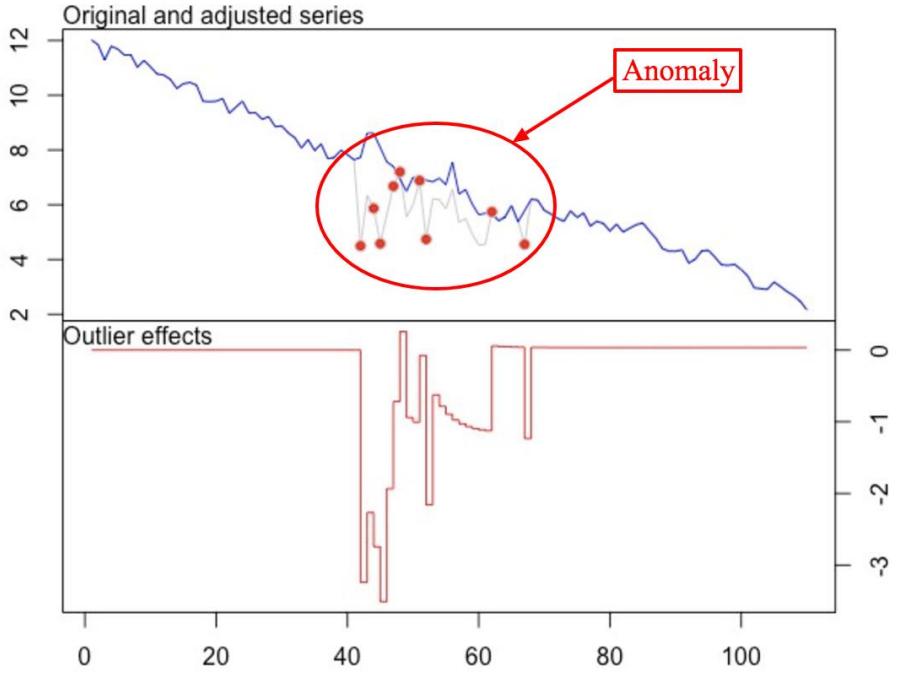


From top to bottom: original time series, seasonal, trend and residue parts retrieved using STL decomposition.
If you analyse deviation of residue and introduce some threshold for it, you'll get an anomaly detection algorithm.

Part B3 TeamCognito's approach for a well-rounded solution



Actual time series (Green), predicted time series made using CART model (Blue), and anomalies detected as deviation from forecasted time series.



Two time series built using original ARIMA model and adjusted for outliers ARIMA model. Each time you work with a new signal you should build a new ARIMA model.

Part C TeamCognito's initial simplest algorithm for anomaly detection

Part C1 *Detection and Location algorithm for metres with saved history*

Let $M = \{M0, M1, M2, \ldots, Mn\}$ be an ordered set of existing smart metres to be inspected.

Let M_i be a smart metre from M.

For metre M_i, we have a history for its readings as a set of floating point values from a minimum (MIN) and maximum (MAX) value. Let the history values for M_i be an ordered set M_i^H

Let μ_i be mean and σ_i be the standard variation of M_i^H

A. Perform moving average operation on M_i^H
   Let $(M_i^H)'$ be the resultant ordered set.
B. Find ordered set of residuals $R_i = M_i^H \sim (M_i^H)'$
C. Let σ be the standard variation for ordered set R_i
D. Let ε_1 and ε_2 be two floating real numbers such that if there <u>are</u> ε_2 values in ordered set R_i all of which are more than $\varepsilon_2\sigma$, then we have a possible scenario of an anomaly.
E. The instance/time of possibly anomalous reading(s) can be determined by the indices of the value of R_i:
   a. Let l be the number of values in M_i^H
   b. Find absolute differences between adjacent values in M_i^H
      Let this ordered set of differences be named r_i
   c. Find values greater than ε_3
      For a value v_j in r_i greater than ε_3 where j is the index of the value v_j in r_i, the metre reading value positions in ordered set R_i are probably anomalous readings are j and j+1

ε_1, ε_2, and ε_3 are three hyper-parameters for this protocol that needs to be adjusted accordingly and fine-tweaked to work and produce reliable results, and will be dependent on the spread and qualitative features of the history data series M_i^H

Part C2 *Detection and Location algorithm for metres in the same locality/district*

Let $M = \{M0, M1, M2, \ldots, Mn\}$ be an ordered set of existing smart metres to be inspected.

A simple unsupervised KMeans Clustering algorithm with k = number of localities/districts considered can easily locate and find the anomalous meters.

Part D TeamCognito's initial algorithm for anomaly detection implemented as a Python Class/Module

(Source code)

```
import numpy as np


class AnomalyDetector:

    def __init__(self, meter_readings, meter_history_readings):
        self.meter_readings = meter_readings
        self.meter_history_readings = meter_history_readings

    def moving_average(signal, window=1):
        window = np.ones(int(window)) / float(window)
        return np.convolve(signal, window, "same")

    def find_anomaly(self, readings=None, E1=2, E2=1):
        anomaly_indices = []

        if readings is None:
            return None
        elif type(readings) is list:
            readings = np.asarray(readings)

        averaged = self.moving_average(signal=readings, window=1)
        residuals = np.absolute(readings - averaged)

        std = np.std(residuals)

        right_lim = averaged + (E1 * std)
        left_lim = averaged - (E1 * std)

        filtered_reads = np.where((readings > right_lim) | (readings < left_lim))

        index_value = list(enumerate(filtered_reads))
        anomaly_indices.extend(index_value)

        # for index, x, averaged in zip(count(), readings, averaged):
        #     if x>averaged+(E1*std) or x<averaged-(E1*std):
        #         anomaly_indices.append((index,x))

        return (len(anomaly_indices) >= E2), anomaly_indices

    def detect_from_meters(self, E1=1.5, E2=1):

        _, anomaly_indices = self.find_anomaly(readings=self.meter_readings, E1=E1, E2=E2)

        meter_indices = [index for index, _ in anomaly_indices]
        return meter_indices

    def detect_from_meter_history(self, meter_index, E1, E2):
        readings = self.meter_history_readings

        if meter_index < 0 or meter_index >= len(readings):
            return None

        _, anomaly_indices = self.find_anomaly(readings=readings[meter_index], E1=E1, E2=E2)

        time_indices = [index for index, _ in anomaly_indices]
        return time_indices
```