Ilkogretim Online - Elementary Education Online, 2020; Vol 19 (Issue 4): pp. 7865-7885
http://ilkogretim-online.org
doi: 10.17051/ilkonline.2020.04.765184

Evolution Of Multi-Stream Data In A Smart City's Vanet

Arindam Pal Assistant Professor, Mahadevananda Mahavidyalaya, Kolkata-700120
arinpal2020@gmail.com

Abstract:

In smart cities, vehicular ad-hoc networks (VANETs) will play an important role with the development of security and smart vehicle video surveillance services. With the smart components, multimedia communication drivers over VANETs allow passengers to capture live scenes, share and access on-wheel multimedia services. In this study, I present an application that provides multiple services in a smart city VANET. I suggest Since, the main concern of video streaming transmission in such a highly dynamic environment is to increase the quality of experience (QoE). The massive growth of Big Data and the evolution of Internet of Things (IoT) technologies enable cities to gain valuable intelligence from large amounts of real-time generated data. In a smart city, various IoT devices sequentially generate streams of data that need to be analyzed in a short period of time; Some use big data techniques. Distributed Stream Processing Framework (DSPF) has the ability to handle real-time data processing for smart cities. According to our results, selecting a proper structure in the data analytics layer of a smart city requires sufficient knowledge about the characteristics of the target applications. The Internet of Vehicles(IoV) enabled vehicles to exchange information with roadside units (RSU) with minimum or zero human intervention. This autonomous nature of IoV requires strong authentication for each entity to recognize each other, as well as these entities, should ensure the integrity of the exchanged information. Otherwise, this autonomy will attract malicious users and malicious activity. Due to the dynamic nature of the IoV network, it is almost impossible to solve the Cyber Security issue with the centralized authentication system throughput.

Key Words: smart cities, VANET, quality of experience (QoE), Internet of Things (IoT), Distributed Stream Processing Framework (DSPF)

Introduction

Nowadays, data streams occur in many real-world situations. For example, they are developed for sensors, web traffic, satellites and other interesting use cases. We need to process them in a fast manner and extract as much knowledge from them as possible. Data streams have their own specific characteristics for processing and data mining. For example, they can be very fast, we cannot process the entire history of the data stream in memory, so we have to do it incrementally or in (eg sliding) windows.

In this post, I will cover a data stream clustering method that was developed by me. In R, you can do data stream clustering by stream package, but! There are methods for

clustering only one stream (not multiple streams). However, I want to show you the clustering of multiple data streams, so from multiple sources (eg sensors).

I have developed a clustering method adapted for time series streams - known as ClipStream . This approach is attribute based, so multiple attributes (features) are collected from multiple stream windows. You can read more about it from my journal paper.

I will show you the use case of smart meter data - time series of electricity consumption (consumer) from London (openly available here). The motivation for clustering such data is:

1. Find out the general pattern of costs,

2. Detecting outliers,

3. Unsupervised Classification of Consumers,

4. Create more predictable groups of time series,

5. Change and monitor behavior.

The motivation for processing them as streams is that power consumption (or production) data is measured every 5, 15 or 30 minutes.

Clipstream has the following steps (very briefly):

1. Computing data stream representation (reduced dimensionality) for each stream – eg summary,

2. Identify external (extraordinary) flows directly from the extracted representation,

3. Cluster non-outlier representation by K-medoids,

4. Assign the outlier stream to the nearest cluster (medoid),

5. Total time by cluster

On the other hand, the Internet of Things (IoT) is becoming the primary grounds for data mining and Big Data analytics. With the rapid growth of IoT and its use cases in different domains such as Smart City, Mobile e-Health and Smart Grid, streaming applications are driving a new wave of data revolutions. In most IoT applications the resulting analytics give some feedbacks to the system to improve it. Compared to the other Big Data domains, there is a low-latency cycle between system responses which makes it necessary to process events in real-time, to derive acceptable responsiveness. In all of these domains, one of the most fundamental challenge is to explore the large volumes of data and extract useful information for future actions. In particular, this real-time exploration has to be done at massive scales.

Nowadays generated data in IoT era have several characteristics that put them in the class of Big Data. Smart cities follow four steps to improve the quality of life and enable

economic growth through a network of connected IoT devices and other technologies. These steps are as follows:

1. Collection – Smart sensors gather real-time data

2. Analysis – The data is analysed to gain insights into the operation of city services and operations

3. Communication – The results of the data analysis are communicated to decision makers

4. Action – Action is taken to improve operations, manage assets and improve the quality of city life for the residents

The ICT framework brings together real time data from connected assets, objects and machines to improve decision making. However, in addition, citizens are able to engage and interact with smart city ecosystems through mobile devices and connected vehicles and buildings. By pairing devices with data and the infrastructure of the city, it is possible to cut costs, improve sustainability and streamline factors such as energy distribution and refuse collection, as well as offering reduced traffic congestion, and improve air quality.

BIG DATA PLATFORM:

A big data platform acts as an organized storage medium for large amounts of data. Big data platforms utilize a combination of data management hardware and software tools to store aggregated data sets, usually onto the cloud.

Google Cloud offers lots of big data management tools, each with its own specialty. BigQuery warehouses petabytes of data in an easily queried format.

Users can analyze data stored on Microsoft's Cloud platform, Azure, with a broad spectrum of open-source Apache technologies, including Hadoop and Spark.

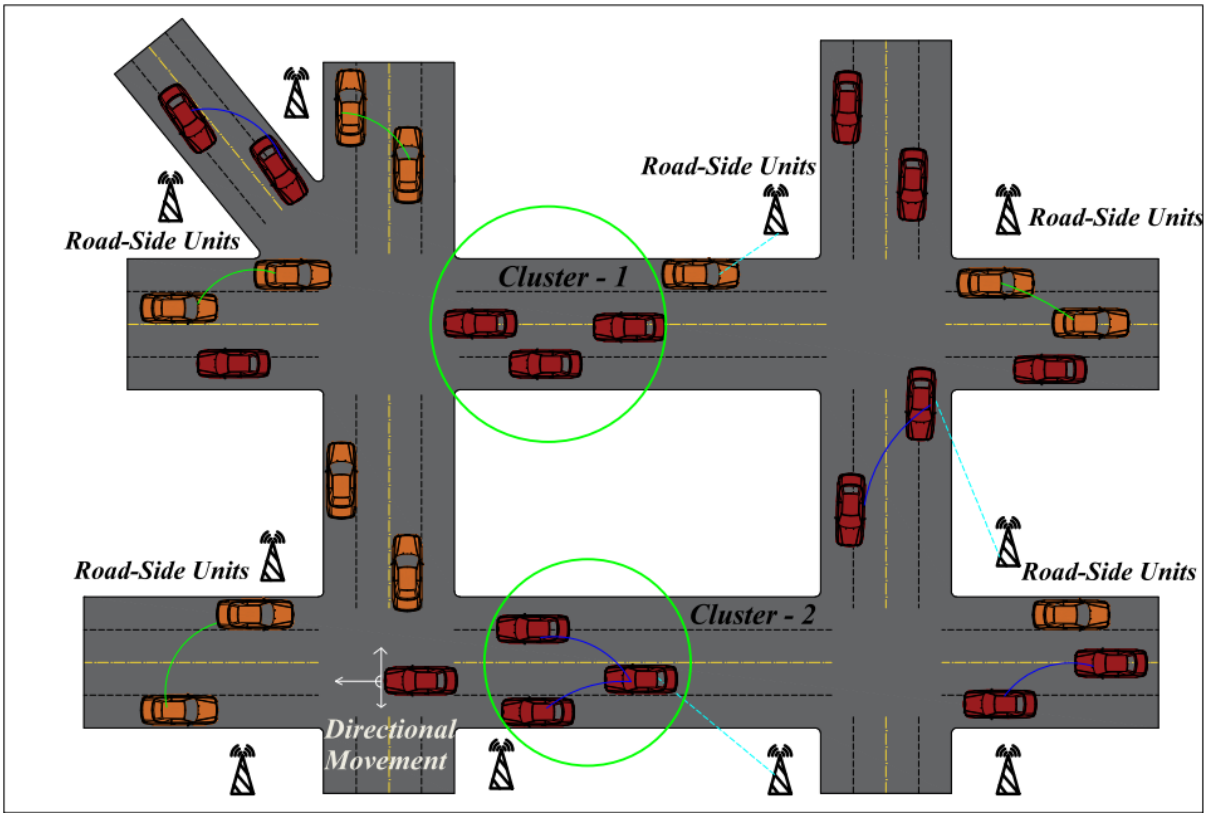
Best known as AWS, Amazon's cloud-based platform comes with analytics tools that are designed for everything from data prep and warehousing to SQL queries and data lake design. All the resources scale with your data as it grows in a secure cloud-based environment. Features include customizable encryption and the option of a virtual private cloud.

Snowflake is a data warehouse used for storage, processing and analysis. It runs completely atop the public cloud infrastructures — Amazon Web Services, Google Cloud Platform and Microsoft Azure — and combines with a new SQL query engine. Built like a SaaS product, everything about its architecture is deployed and managed on the cloud.

VANET:

VANET (Vehicular Ad-hoc Networks) has become a promising field of research and is an important component of Intelligent Transportation Systems (ITS). The dramatic increase

in the number of vehicles equipped with computing technologies and wireless communication devices have given way to new application scenarios that were not feasible before. In this era of modern technology, the Internet of Vehicles (IoV) is an emergent technology. The IoV augments ITS by magnifying the capabilities of VANETs riding on the concept of the Internet of Things (IoT). IoV enabled vehicles to exchange information with roadside units (RSU) with minimum or zero human intervention. This autonomous nature of IoV requires strong authentication for each entity to recognize each other, as well as these entities, should ensure the integrity of the exchanged information. Otherwise, this autonomy will attract malicious users and malicious activity. Due to the dynamic nature of the IoV network, it is almost impossible to solve the Cyber Security issue with the centralized authentication system.



Ad Hoc Network Features:

Infrastructureless network: Like other network, it doesn't require centralized controlling authority, routers, and physical medium for transmitting data.

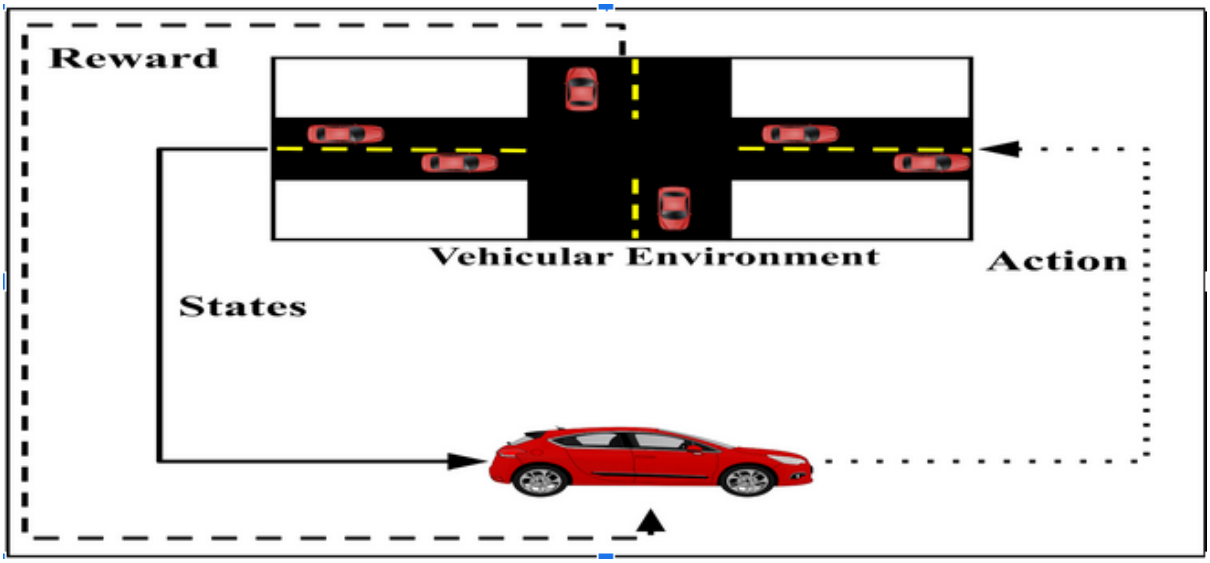
Nodes in the network can be mobile: Nodes can be stationary or nodes in the network have freedom to move in the communication range of network nodes. If all nodes are moving with a constant speed in the range of each other, connection can remain without interruption. In some cases, every node moves at a different speed, which can be connected and disconnected from the network.

Each node acts as a router and takes the decision of forwarding the packets: Nodes in the network themselves play a role of router. Nodes either send their own messages or

forward another node messages to destination nodes. Route-finding process is initiated when a particular message is sent by the source node. Nodes in the network have individual or group strategy for forwarding messages.

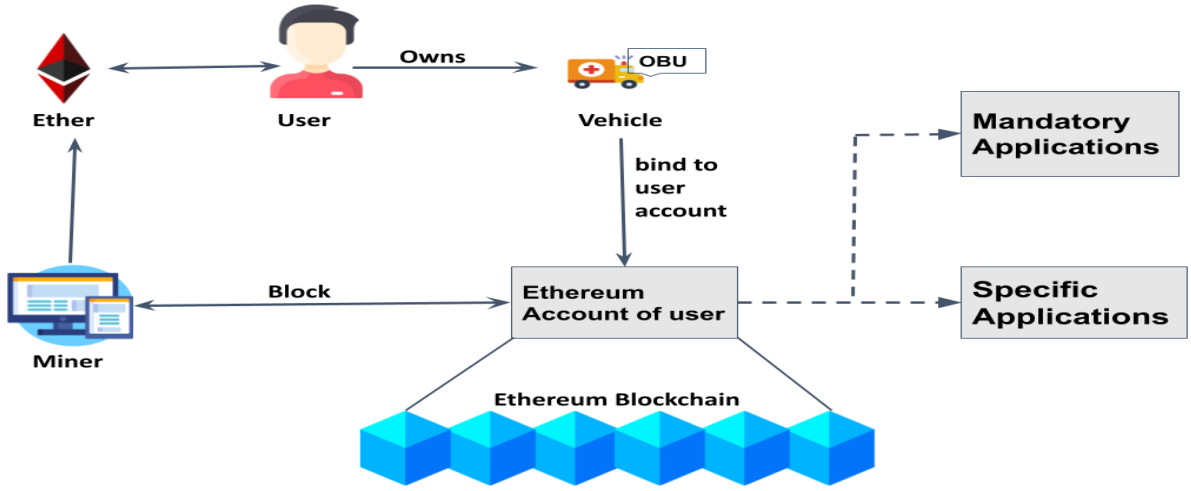
No centralized controlling authority: In other wireless network, behaviors of nodes in the network are controlled by centralized controlling authority. In the case of ad hoc network, there is no centralized controlling authority.

Flexible in nature, can establish, and can dissolve as nodes move: As nodes in the network can move, they result in formation and deformation of network from time to time.



Vehicular Ad Hoc Networks (VANETs) are characterized by high mobility of nodes and volatility, which make privacy, trust management, and security challenging issues in VANETs' design. In such networks, data can be exposed to a variety of attacks, the most dangerous is false information dissemination, which threatens the safety and efficiency of transportation systems. False emergency messages can be injected by inside attackers to announce fake incidents such as traffic accidents, resulting in a false information attack. As the data in VANET is based on events, any trust mechanism must first identify the true events. To address these security challenges, a blockchain-based authentication scheme and trust management model are proposed for VANETs. Using the authentication scheme, vehicles are enabled to send messages anonymously to the roadside units (RSUs) and the identity privacy of vehicles is protected. Besides, the proposed trust management model is designed to detect and deal with false information by evaluating the trustworthiness of vehicles and data. Using the trust model, when vehicles report an incident to the nearest RSU, the RSU is able to verify whether or not the incident took place. This mechanism ensures that RSUs send only verified event notifications. Finally, RSUs participate in updating the trust values of vehicles and store these values in the blockchain. The efficiency of the proposed authentication scheme is validated through analysis while the trust model is validated through simulations. The results obtained show that the proposed authentication scheme and the trust model provide better

performance than other state-of-the-art models where malicious vehicles can be identified efficiently and RSUs are enabled to broadcast only legitimate events.



Abbreviations

IoT:

Internet of Things

DSPF:

distributed stream processing framework

RFID:

Radio Frequency IDentification

HDFS:

Hadoop Distributed File System

API:

application programming interface

DB:

database

JSON:

JavaScript Object Notation

DAG:

directed acyclic graph

ETL:

extract, transform, load

CEP:

Complex Event Processing

ML:

machine learning

IVP:

Image and Video Processing

Evalution:

I will show you use case on smart meter data - electricity consumption time series (consumers) from London (openly available here). Motivation for clustering such a data is to:

extract typical patterns of consumption,

detect outliers,

unsupervised classification of consumers,

create more forecastable groups of time series,

monitor changes and behavior.

The motivation to process them as streams is the fact that the data of electricity consumption (or production) are measured every 5, 15 or 30 minutes.

ClipStream has following steps (very briefly):

computing data streams representation for every stream (dimensionality reduction) - i.e. synopsis,

detect outlier (anomal) streams directly from extracted representations,

cluster non-outlier representations by K-medoids,

assign outlier streams to nearest clusters (medoids),

aggregate time series by clusters,

detect changes in aggregated time series.

In this post, I will mainly focus on the first 4 parts of my approach, since there are applicable to various use cases - not only on smart meter data and its forecasting. The whole method is developed in unsupervised fashion (yey!), so representation, clustering and outlier detection of time series streams are "learned" unsupervised.

Data exploration

Firstly, read the London smart meter data, which compromise more than 4000 consumers and load all the needed packages.

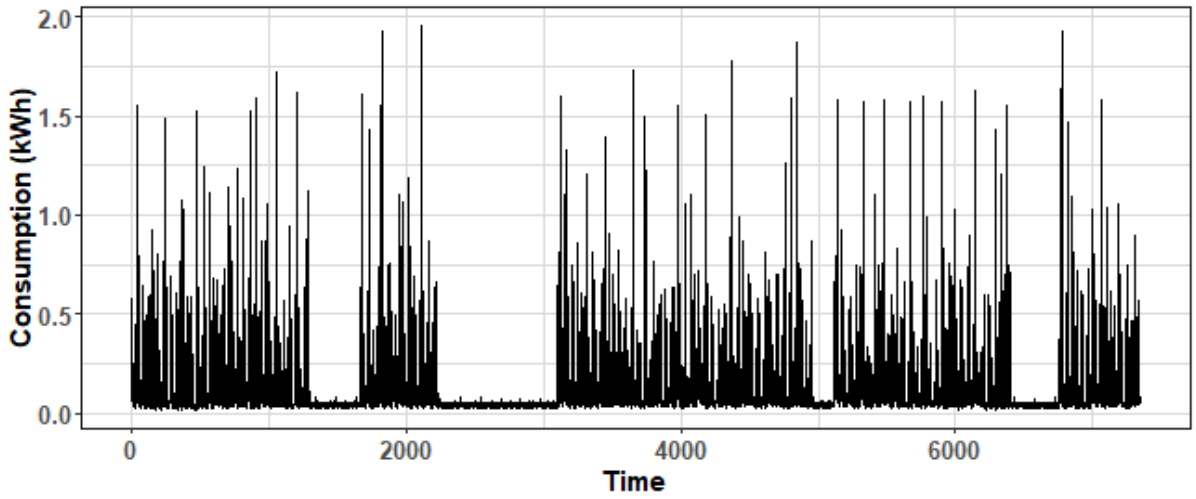
```
library(data.table)
library(parallel)
library(cluster)
library(clusterCrit)
library(TSrepr)
library(OpenML)
library(ggplot2)
library(grid)
library(animation)
library(gganimate)
library(av)
data <- OpenML::getOMLDataSet(data.id = 41060)
data <- data.matrix(data$data)
```

Let's subset first 1000 consumers for more simple operations.

```
data_cons <- data[1:1000,]
period <- 48 # frequency of time series, every day 48 measurements are gathered
```

Let's plot random consumer for better imagination.

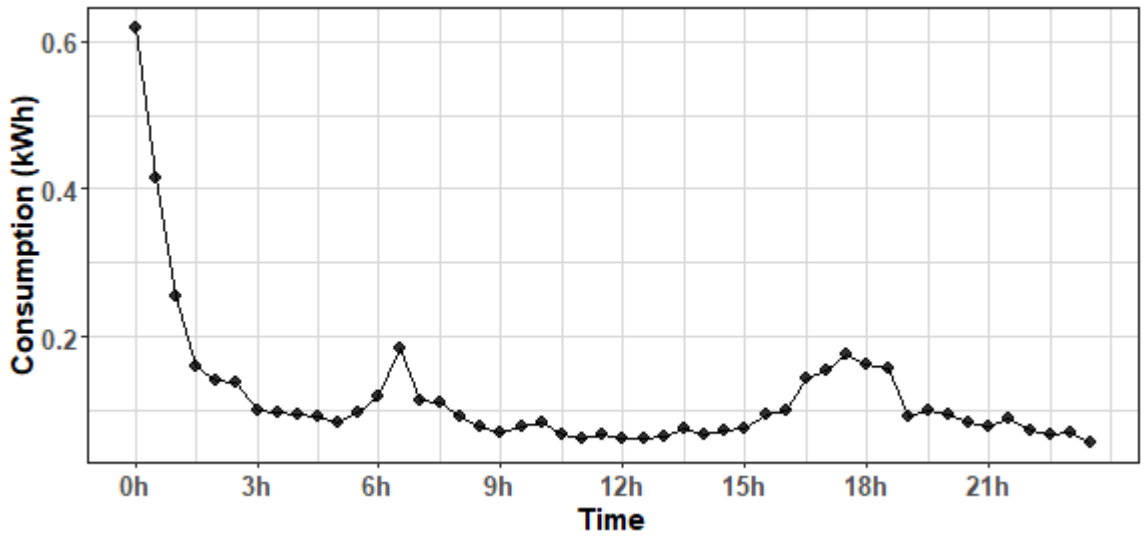
```
ggplot(data.table(Time = 1:ncol(data_cons),
                  Value = data_cons[200,])) + geom_line(aes(Time, Value)) +
 labs(y = "Consumption (kWh)") + theme_ts
```



We can see stochastic behavior, but also some consumption pattern.

We can plot, for example, its average daily or weekly profile. We will do it by TSrepr's package function repr_seas_profile.

```
ggplot(data.table(Time = 1:period,
   Value = repr_seas_profile(data_cons[200,], freq = period,func = mean))) +
 geom_line(aes(Time, Value)) + geom_point(aes(Time, Value), alpha = 0.8, size = 2.5) +
 scale_x_continuous(breaks = seq(1, 48, 6),  labels = paste(seq(0, 23, by = 3), "h", sep = "")) + labs(y = "Consumption (kWh)") + theme_ts
```

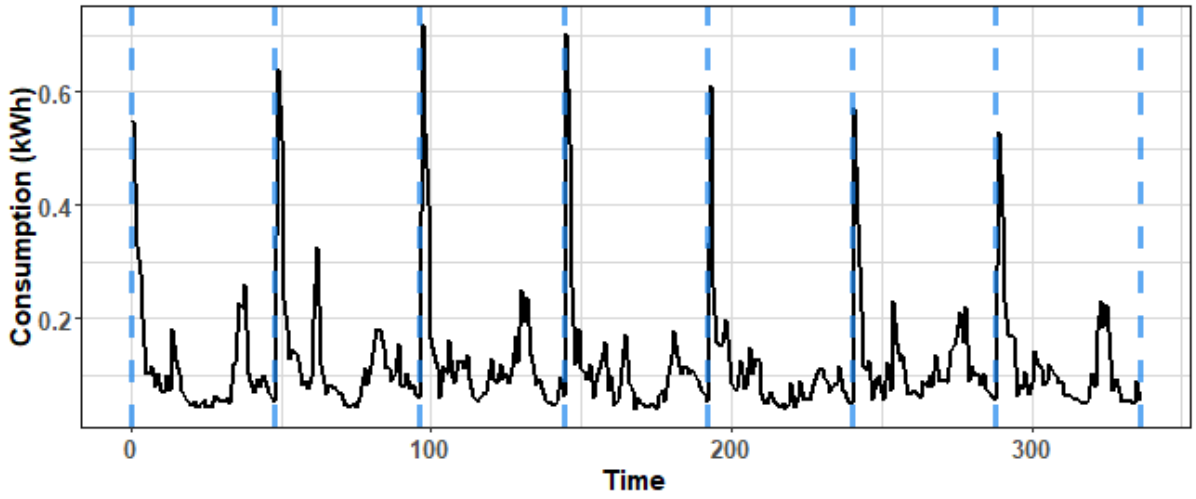


The peak at the start of a day…

Now, let's plot weekly pattern:

```
ggplot(data.table(Time = 1:(period*7), Value = repr_seas_profile(data_cons[200,], freq = period*7, func = mean)))
geom_line(aes(Time, Value), size = 0.8) +
```

```
geom_vline(xintercept = (0:7)*period, color = "dodgerblue2",
           size = 1.2, alpha = 0.7, linetype = 2) +  labs(y = "Consumption (kWh)") + theme_ts
```



There is some change during a week.

Let's explore the whole consumer base, but how when we have more than 1000 time series? We can cluster time series and just plot its daily patterns for example by created clusters. We will reduce the length of the visualized time series and also a number of time series in one plot.

First, extract average daily patterns, we will make it by repr_matrix function from TSrepr. Normalization of every consumer time series - row-wise by z-score is necessary! You can use your own normalization function (z-score and min-max methods are implemented).

```
data_ave_prof <- repr_matrix(data_cons, func = repr_seas_profile,args = list(freq = period,
                                                                              func = median),normalise = TRUE,
                                                                              func_norm = norm_z)
```

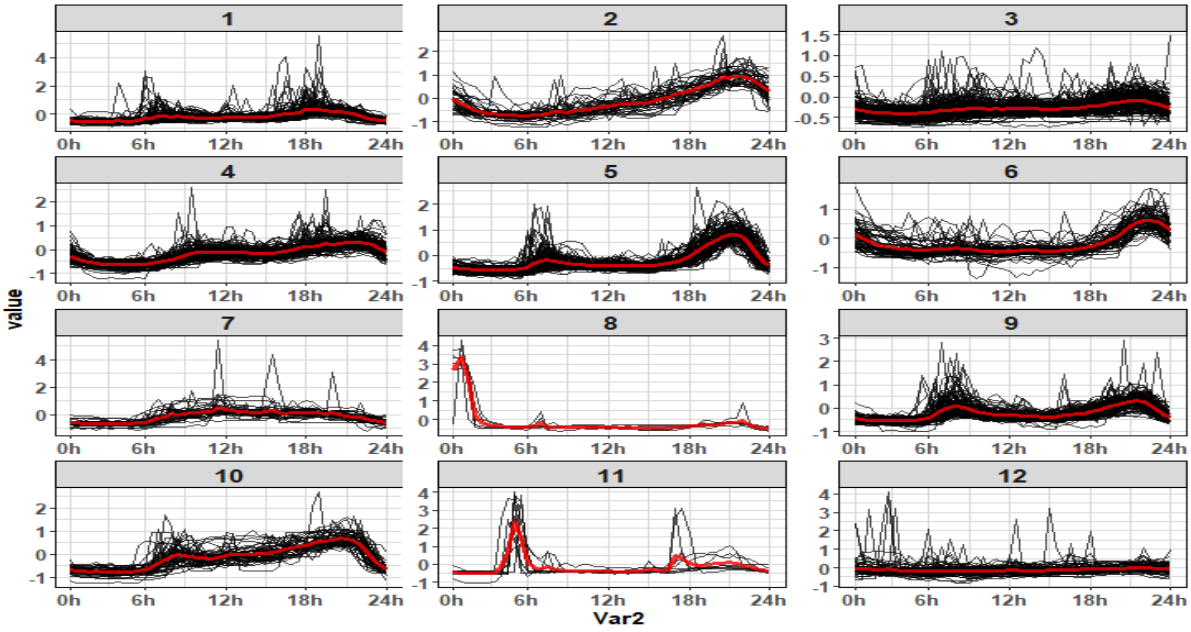
Let's cluster computed mean daily profiles for example by K-means and to 12 clusters:

```
res_clust <- kmeans(data_ave_prof, 12, nstart = 20)
```

Preprocess clustering results with data.table package and plot them:

```
data_plot <- data.table(melt(data_ave_prof))
data_plot[, Clust := rep(res_clust$cluster, ncol(data_ave_prof))]
 data_centroid <- data.table(melt(res_clust$centers))
data_centroid[, Clust := Var1]
ggplot(data_plot) +
  facet_wrap(~Clust, scales = "free", ncol = 3) +
```

```
 geom_line(aes(Var2, value, group = Var1), alpha = 0.7) +
 geom_line(data = data_centroid, aes(Var2, value, group = Var1),
  alpha = 0.8, color = "red", size = 1.2) +  scale_x_continuous(breaks = c(1,seq(12, 48, 12)),
labels = paste(seq(0, 24, by = 6), "h", sep = "")) + theme_ts
```



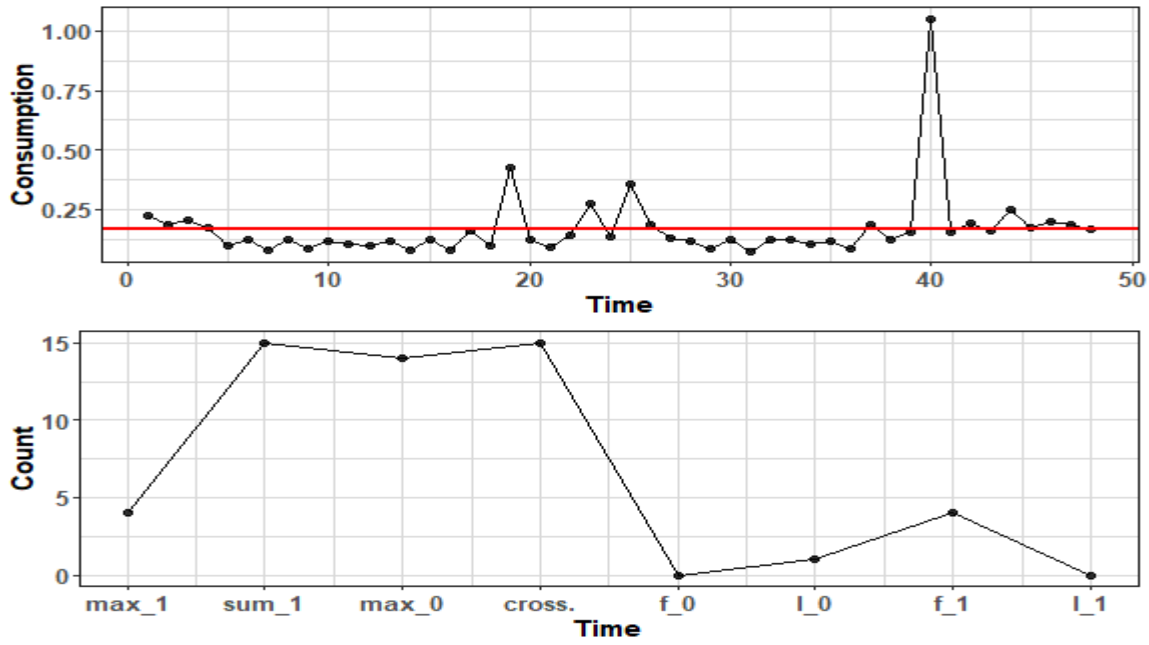
Various interesting patterns of daily consumption!

BUT (here comes motivation)! When we would like to cluster just most recent data (for example 2-3 weeks), and update it regularly every day (or even thought half-hour), automatically detect changes in time series streams and detect anomalies (outlier consumers), detect automatically the number of clusters, and make it as quickly as possible… We can not do it like in the previous case, because of computational and memory load and also accuracy. Here comes on screen more sophisticated multiple data streams clustering methods. In ClipStream that uses FeaClip time series streams representation (see my previous post about time series represetnations), a representation can be computed incrementally, clusterings are computed in data batches, outliers are detected straight from representation and etc.

FeaClip is interpretable time series representation. It extracts 8 interpretable features from clipped representation (again please see the mentioned blog post). Let's plot representation from one day consumption time series:

```
ts_feaclip <- repr_feaclip(data_cons[1, 1:48])
 define_region <- function(row, col){
  viewport(layout.pos.row = row, layout.pos.col = col)
}
```

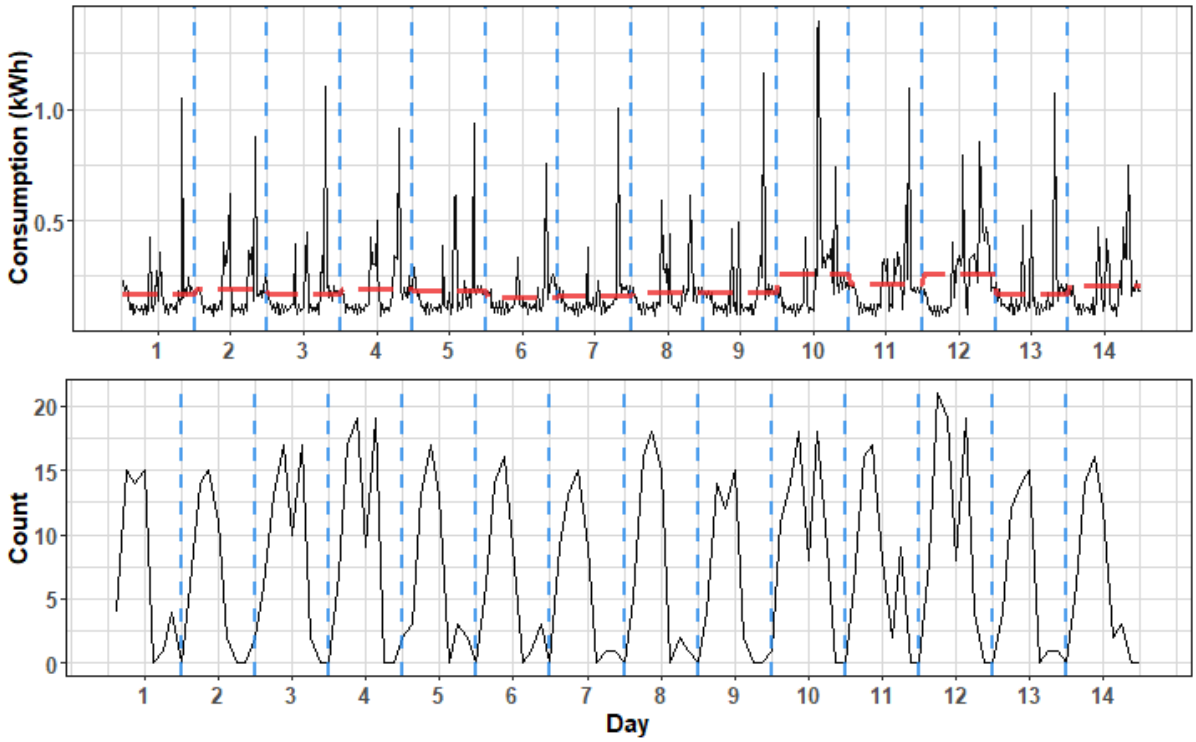
```
gg1 <- ggplot(data.table(Consumption = data_cons[1, 1:48],Time = 1:48)) +
 geom_line(aes(Time, Consumption)) +
 geom_point(aes(Time, Consumption), size = 2, alpha = 0.8) +
 geom_hline(yintercept = mean(data_cons[1, 1:48]), color = "red", size = 0.8, linetype = 7) +
 theme_ts
gg2 <- ggplot(data.table(Count = ts_feaclip, Time = 1:8)) + geom_line(aes(Time, Count)) +
 geom_point(aes(Time, Count), size = 2, alpha = 0.8) +
 scale_x_continuous(breaks = 1:8, labels = names(ts_feaclip)) + theme_ts
grid.newpage()
pushViewport(viewport(layout = grid.layout(2, 1)))
print(gg1, vp = define_region(1, 1))
print(gg2, vp = define_region(2, 1))
```



Let's compute it also for consumption of length 2 weeks - by windows.

```
win <- 14
ts_feaclip_long <- repr_windowing(data_cons[1, 1:(period*win)],
                                  func = repr_feaclip, win_size = period)
```

```
means <- sapply(0:(win-1), function(i) mean(data_cons[1, ((period*i)+1):(period*(i+1))]))
means <- data.table(Consumption = rep(means, each = period),
                    Time = 1:length(data_cons[1, 1:(period*win)]))
 gg1 <- ggplot(data.table(Consumption = data_cons[1, 1:(period*win)],
                          Time = 1:(period*win))) +
  geom_line(aes(Time, Consumption), alpha = 0.95) +
  geom_vline(xintercept = seq(from=period,to=(period*win)-period, by=period),
             alpha = 0.75, size = 1.0, col = "dodgerblue2", linetype = 2) +
  geom_line(data = means, aes(Time, Consumption), color = "firebrick2",
            alpha = 0.8, size = 1.2, linetype = 5) +
  labs(y = "Consumption (kWh)", title = NULL, x = NULL) +
  scale_x_continuous(breaks=seq(from=period/2, to=(period*win), by = period),
                     labels=c(1:win)) +theme_ts
 gg2 <- ggplot(data.table(Count = ts_feaclip_long,
                          Time = 1:length(ts_feaclip_long))) +
  geom_line(aes(Time, Count)) +
  geom_vline(xintercept = seq(from=8,to=(8*win)-8, by=8),
             alpha = 0.75, size = 1.0, col = "dodgerblue2", linetype = 2) +
  # geom_point(aes(Time, Count), size = 2, alpha = 0.8) +
  scale_x_continuous(breaks=seq(from=8/2, to=(8*win), by = 8),
                     labels=c(1:win)) +
  labs(title = NULL, x = "Day") + theme_ts
grid.newpage()
pushViewport(viewport(layout = grid.layout(2, 1)))
print(gg1, vp = define_region(1, 1))
print(gg2, vp = define_region(2, 1))
```



We can see the typical pattern during working days and some changes when consumption goes high - the representation adapts to these changes, but not drastically - because of the average of a window (normalization).

FeaClip animation

Now, let's play a little bit with gganimate and av packages and visualize the behavior of FeaClip through time (stream).

First, prepare whole one time series as a stream.

```
# Representation

data_feaclip_all <- repr_windowing(data_cons[1,],

                                   func = repr_feaclip,

                                   win_size = period)

win <- 14

win_size <- 14*8

n_win <- length(data_feaclip_all)/8 - win

data_feaclip_all_anim <- sapply(0:(n_win - 1), function(i)

  data_feaclip_all[((i*8)+1):((i*8) + win_size)])

data_feaclip_all_anim <- data.table(melt(data_feaclip_all_anim))

data_feaclip_all_anim[, Type := "Representation - FeaClip"]
```

```
# Original TS
win <- 14
win_size <- 14*period
n_win <- ncol(data_cons)/period - win
data_cons_anim <- sapply(0:(n_win - 1), function(i)
  data_cons[1, ((i*period)+1):((i*period) + win_size)])
row.names(data_cons_anim) <- NULL
data_cons_anim <- data.table(melt(data_cons_anim))
data_cons_anim[, Type := "Original TS"]
# Together
data_plot <- rbindlist(list(data_cons_anim,
                            data_feaclip_all_anim))
setnames(data_plot, "Var2", "Window")

data_vline <- data.table(Var1 = c(seq(from=period,to=(period*14)-period, by=period),
                                  seq(from=8,to=(8*14)-8, by=8)),
                         Type = rep(c("Original TS", "Representation - FeaClip"),
                                    each = 13)
                         )
```

Now, create ggplot animation object by ggplot and gganimate functions.

```
gg_anim <- ggplot(data_plot) +
  facet_wrap(~Type, scales = "free", ncol = 1) +
  geom_line(aes(Var1, value)) +
  geom_vline(data = data_vline, aes(xintercept = Var1),
             alpha = 0.75, size = 0.8, col = "dodgerblue2", linetype = 2) +
  labs(x = "Length", y = NULL,
       title = "Time series and its representation, Window: {frame_time}") +
  theme_ts +
```

```
  transition_time(Window)
```

Let's animate!

```
gganimate::animate(gg_anim, fps = 2, nframes = n_win,
                   width = 720, height = 450, res = 120,renderer = av_renderer('animation.mp4'))
```

We can see how FeaClip representation adapts on different behavior of electricity consumption - when it's low around zero, or it gets high etc. - it makes drastic noise reduction and extracts only key features from the time series.

Multiple Data Streams:

We already got a picture of one consumer time series stream. Let's again play with the whole customer base (in our playing scenario with 1000 consumers). In this part, I would like to make nice animation of outlier detection and clustering phase in one image.

First iteration

Let's get through the first iteration (window) of streams processing and clustering of ClipStream. So, first thing to do is to subset window of time series streams and compute FeaClip representations (the most recent window of length 14 days as in the previous case).

```
i = 0

data_win <- data_cons[, ((i*period)+1):((i+win)*period)]

data_clip <- repr_matrix(data_win, func = repr_feaclip,
                         windowing = T, win_size = period)
```

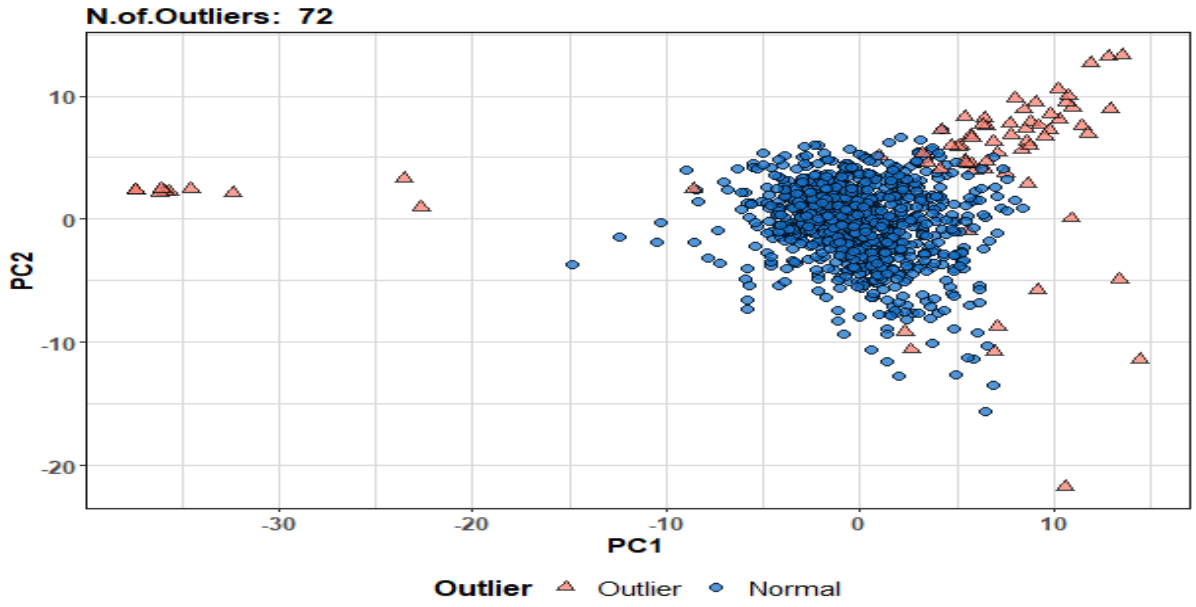
The second phase is an outlier detection from streams. This is done by "simple" IQR-quantile method from 2 key FeaClip features - Sum_1 and number of crossings. The source code for function detectOutliersIQR is in the ClipStream method repo. Let's do it.

```
outliers <- detectOutliersIQR(data_clip, treshold = 1.5)
```

Let's show outlier consumers with PCA:

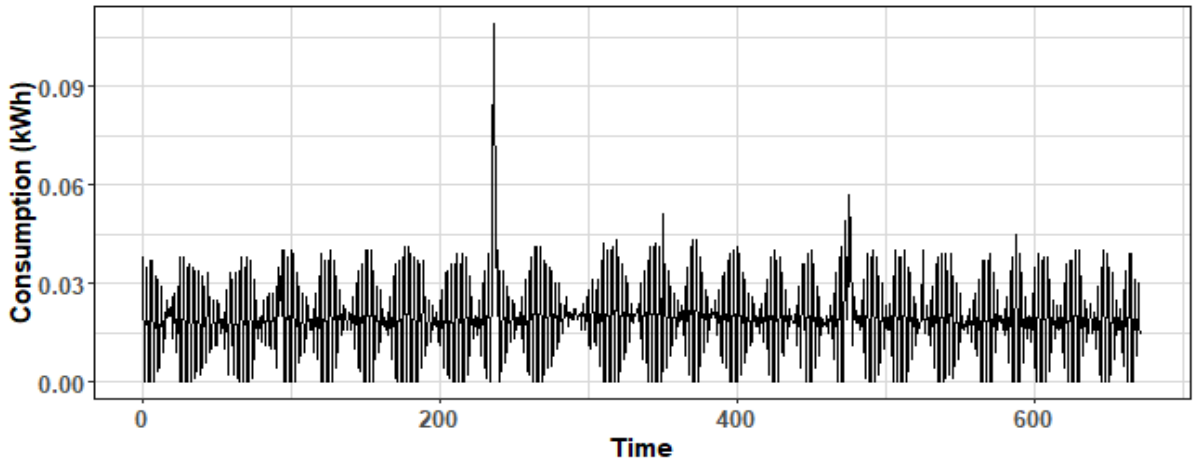
```
pc_clip <- prcomp(data_clip, center = T, scale. = T)$x[,1:2]

pc_clip <- data.table(pc_clip,
                      Outlier = factor(outliers$class))

levels(pc_clip$Outlier) <- c("Outlier", "Normal")

ggplot(pc_clip) +
  geom_point(aes(PC1, PC2, fill = Outlier, shape = Outlier),
             size = 2.8, alpha = 0.75) +
```

```
 scale_shape_manual(values = c(24, 21)) +
 scale_fill_manual(values = c("salmon", "dodgerblue3")) +
 labs(title = paste("N.of.Outliers: " , nrow(pc_clip[Outlier == "Outlier"]))) +
 theme_ts
```

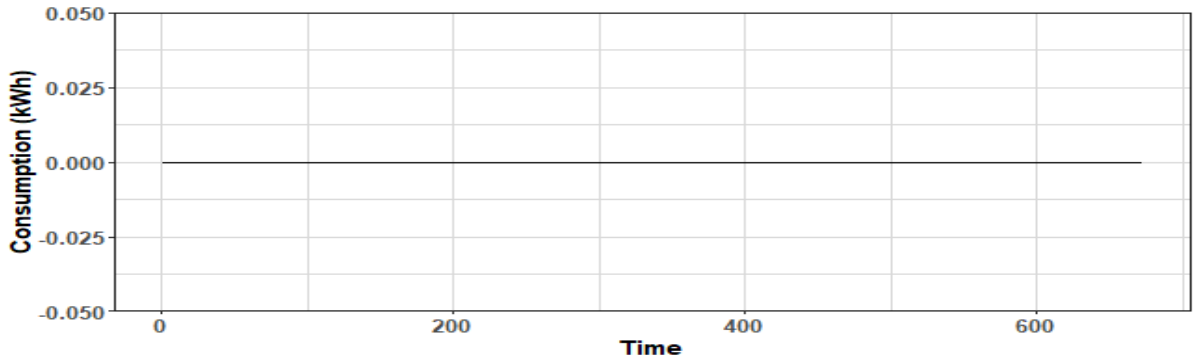


They are nicely located at the periphery of the feature space. Let's find some outlier consumers and plot them:

```
which(pc_clip$PC2 == max(pc_clip$PC2))
## [1] 597
which(pc_clip$PC1 == min(pc_clip$PC1)) # zeros
## [1] 562 908 961
ggplot(data.table(Time = 1:ncol(data_win),
                  Value = data_win[which(pc_clip$PC2 == max(pc_clip$PC2)),])) +
 geom_line(aes(Time, Value)) +
 labs(y = "Consumption (kWh)") +
 theme_ts
```




```
ggplot(data.table(Time = 1:ncol(data_win),
                  Value = data_win[which(pc_clip$PC1 == min(pc_clip$PC1))[1],])) +
 geom_line(aes(Time, Value)) + labs(y = "Consumption (kWh)") + theme_ts
```



The first scenario is a consumer with a few large moments of high consumption and lot of crossings. The second scenario is the zero consumption type of a consumer.

The clustering phase is executed by an automatic K-medoids method. The number of clusters is automatically selected by the Davies-Bouldin index from the predefined range of clusters. We will cluster only non-outlier representations, for the reduction of data and also higher stability of clustering results.

```
# n.of.clusters range
k.min <- 8
k.max <- 15
 clip_filtered <- data_clip[-outliers$outliers,]
clus_res <- clusterOptimKmedoidsDB(clip_filtered, k.min, k.max, 4,
                                   criterium = "Davies_Bouldin")
 clustering <- outliers$class
```

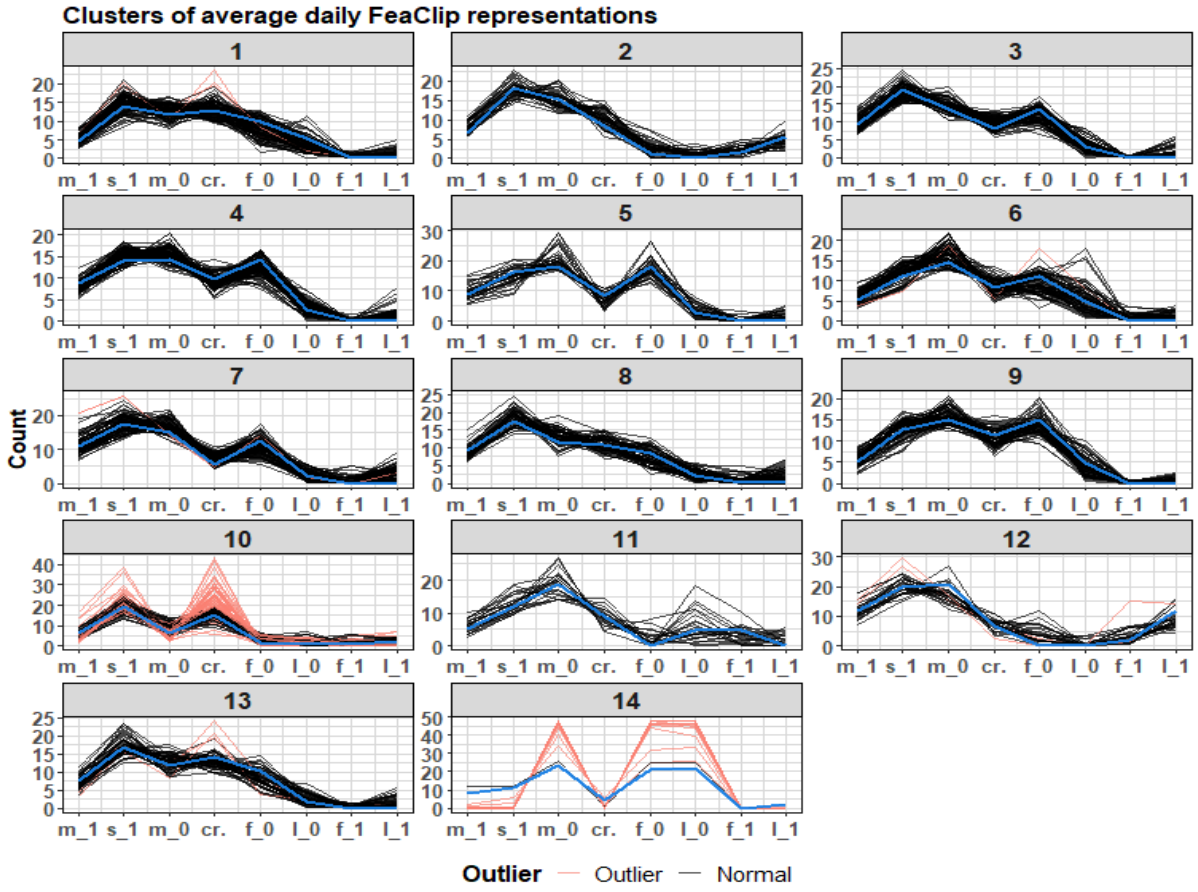
```
clustering[which(clustering == 1)] <- clus_res$clustering
```

Let's assign outliers to nearest clusters (i.e. to nearest medoids).

```
out_clust <- sapply(seq_along(outliers$outliers),
                    function(z)
                      my_knn(clus_res$medoids,
                           as.numeric(data_clip[outliers$outliers[z],])))
 clustering[which(clustering == 0)] <- out_clust
```

In the plot, average daily FeaClip representations will be plotted, because of the longer length of original FeaClip representation.

```
ave_clip <- repr_matrix(data_clip, func = repr_seas_profile,
                        args = list(freq = 8, func = mean))
data_plot <- data.table(melt(ave_clip))
data_plot[, Clust := rep(clustering, ncol(ave_clip))]
pc_clip[,Var1 := 1:.N]
data_plot[pc_clip[, .(Var1, Outlier)], Outlier := i.Outlier, on = .(Var1)]
medoids <- data.table(melt(repr_matrix(clus_res$medoids,
                                       func = repr_seas_profile,
                                       args = list(freq = 8, func = mean))))
medoids[, Clust := Var1]
 ggplot(data_plot) + facet_wrap(~Clust, scales = "free", ncol = 3) +
  geom_line(aes(Var2, value, group = Var1, color = Outlier), alpha = 0.7) +
  geom_line(data = medoids, aes(Var2, value, group = Var1), alpha = 0.9, color = "dodgerblue2", size = 1) + labs(title = "Clusters of average daily FeaClip representations",
 y = "Count", x = NULL) +
  scale_x_continuous(breaks = 1:8, labels = c("m_1", "s_1", "m_0", "cr.",
  names(ts_feaclip)[5:8])) +  scale_color_manual(values = c("salmon", "black")) + theme_ts
```



We can see that the number of outliers and clusters dynamically changes through time (daily windows). You can build your own data streams clustering algorithm with a different streams representation.

Conclusion

In this post, I showed you how to process and cluster multiple seasonal time series streams with the ClipStream method. I showed you also detected outliers that were automatically found from the FeaClip representation. The results of clustering can be also used for improving forecasting accuracy of aggregated consumption.

References

Agarwal S. 2016 state of fast data and streaming applications survey. https://www.opsclarity.com/2016-state-fast-data-streaming-applications-survey/. Accessed 12 Oct 2017.

Díaz M, Martín C, Rubio B. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. J Netw Comput Appl. 2016;67:99–117.

Article Google Scholar

Zhu C, Zhou H, Leung VC, Wang K, Zhang Y, Yang LT. Toward big data in green city. IEEE Commun Mag. 2017;55(11):14–8.

Article Google Scholar

Chen F, Deng P, Wan J, Zhang D, Vasilakos AV, Rong X. Data mining for the internet of things: literature review and challenges. Int J Distrib Sens Netw. 2015;11(8):431047.

Article Google Scholar

Guo Y, Rao J, Jiang C, Zhou X. Moving hadoop into the cloud with flexible slot management and speculative execution. IEEE Trans Parallel Distrib Syst. 2017;3:798–812.

Article Google Scholar

Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.

Article Google Scholar

Goudarzi M. Heterogeneous architectures for big data batch processing in mapreduce paradigm. IEEE Trans Big Data. 2017. https://doi.org/10.1109/TBDATA.2017.2736557.

Article Google Scholar

Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, et al. Storm@twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. New York: ACM; 2014. p. 147–56.

Zaharia M, Das T, Li H, Shenker S, Stoica I. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. HotCloud. 2012;12:10.

Google Scholar

Katsifodimos A, Schelter S. Apache flink: stream analytics at scale. In: 2016 IEEE international conference on cloud engineering workshop (IC2EW). New York: IEEE; 2016. p. 193.