Adaptive Clustering for Dynamic IoT Data Streams

Daniel Puschmann, Payam Barnaghi, *Senior Member, IEEE,* and Rahim Tafazolli, *Senior Member, IEEE*

***Abstract*—The emergence of the Internet of Things (IoT) has led to the production of huge volumes of real-world streaming data. We need effective techniques to process IoT data streams and to gain insights and actionable information from real-world observations and measurements. Most existing approaches are application or domain dependent. We propose a method which determines how many different clusters can be found in a stream based on the data distribution. After selecting the number of clusters, we use an online clustering mechanism to cluster the incoming data from the streams. Our approach remains adaptive to drifts by adjusting itself as the data changes. We benchmark our approach against state-of-the-art stream clustering algorithms on data streams with data drift. We show how our method can be applied in a use case scenario involving near real-time traffic data. Our results allow to cluster, label and interpret IoT data streams dynamically according to the data distribution. This enables to adaptively process large volumes of dynamic data online based on the current situation. We show how our method adapts itself to the changes. We demonstrate how the number of clusters in a real-world data stream can be determined by analysing the data distributions.**

***Index Terms*—Internet of Things, Stream Processing, Adaptive Clustering**

I. Introduction

The shift from the desktop computing era towards ubiquitous computing and the IoT has given rise to huge amounts of continuous data collected from the physical world. The data produced in the IoT context has several characteristics which makes it different from other data used in common database systems and machine learning or data analytics. IoT data can come from multiple different heterogeneous sources and domains, for example numerical observations and measurements from different sensors or textual input from social media streams. Common data streams usually follow a Gaussian distribution over a long-term period. However, in IoT applications we need to consider short-term snapshots of the data, in which we can have a wider range of more sporadic distributions. Furthermore the nature of IoT data streams is dynamic and its underlying data distribution can change over time. Another point is that the data comes in large quantities and is produced in real-time or close to real-time. This necessitates development of IoT specific data analytics solutions which can handle the heterogeneity, dynamicity and velocity of the data streams.

To group the data coming from the streams, we can use clustering or classification methods. Classification methods require supervised learning and need labelled training data. There are usually huge amount of data produced in IoT applications, however, these data lack having labels, which makes these types of methods infeasible to be used. While clustering methods avoid this pitfall since they do not need supervised learning, they work best in offline scenarios where all data is present from the start and the data distribution remains fixed. In this paper, we propose a clustering method with the ability to cope with changes in the data stream which makes it more suitable for IoT data streams.

Data is usually clustered according to different criteria; e.g. similarity and homogeneity. The clustering results in a data analysis scenario can be interpreted as categories in a dataset and can be used to assign data to various groups (i.e. clusters). In this paper we discuss an adaptable clustering method that analyses the distribution of data and updates the cluster centroids according to the online changes in the data stream. This allows creating dynamic clusters and assigning data to these clusters not only by their features (e.g. geometric distances), but also by investigating how the data is distributed at a given time. We evaluate this clustering method against several state-of-the-art methods on evolving data streams.

To showcase the applicability of our work, we use a case study from an intelligent traffic analysis scenario. In this scenario we cluster the traffic sensor measurements according to features such as average speed of vehicles and number of cars. These clusters can then be analysed to assign them a label; for example, a cluster that always includes the highest number of cars, according to the overall density of the cars at a given time and/or the capacity of a street, will be given the "busy" tag. By further abstracting we can identify events such as traffic jams, which can be used as an input for automated decision making systems such as automatic rerouting via GPS navigators.

The remainder of the paper is organised as follows. In Section II we present the state of the art and discuss the benefits and drawbacks of different stream cluster algorithms. We present related work to analyse stream data with concept and data drifts. The silhouette coefficient is chosen as a metric for measuring the cluster quality and the mathematical backgrounds of the method described in Section II. In Section III, we introduce the concepts of our adaptive online clustering method which automatically computes the best number of clusters based on the data distribution. Section IV describes the proposed adaptive clustering method in more technical details. We present evaluations of our work in Section V. In Section V-A we compare our method against state-of-the-art methods on a synthesised data set. We have conducted a case study using traffic data and present the results in Section V-B. In Section VI, we discuss the significance of our work and outline the future work.

Month XX, 2016

D. Puschmann, P. Barnaghi and R. Tafazolli are with the Institute for Communication Systems at the University of Surrey

Manuscript received July 2016; revised September 2016.

II. Related work

Several approaches for the clustering problem exist, however we will only take a closer look at a particular approach: Lloyds Algorithm, better known under the name k-means [1]. It should be noted that this particular approach has been selected to be improved for the purpose of streaming data clustering because of its simplicity. The concept of utilising the data distribution can be also applied to determining the parameter k for k-Median [2] or the number of classes in unsupervised multi class support vector machines [3] and other clustering algorithms.

k-means splits a given data set into k different clusters. It does so by first choosing k random points within the data sets as initial cluster centroids and then assigning each data point to the most suitable of these clusters while adjusting the centre. This process is repeated with the output as the new input arguments until the centroids converge towards stable points. Since the final results of the clustering is heavily dependent on the initial centroids, the whole process is carried out several times with different initial parameters. For a data set of fixed size this might not be a problem; however in the context of streaming data this characteristic of the algorithm leads to heavy computational overload.

Convergence of k-means to a clustering using the random restarts not only means that this procedure takes additional time, but depending on the data set, k-means can produce lower quality clusters. k-means++ [4] is a modification of k-means that intelligently selects the initial centroids based on randomised seeding for the initial cluster centroids. While the first center is chosen randomly from a uniform distribution, the following centroids are selected with probability weighted based on their proportion to the overall potential.

STREAM [5] is a one-pass clustering algorithm which treats data sets, which are too large for to be processed in-memory, as a data stream; however, the approach has shown limitations in cases where the data stream evolves over time leading to misclustering. Aggarwal *et al.* [6] introduce their approach called CluStream that is able to deal with these cases. CluStream is also able to give information about past clusters for a user defined time horizon. Their solution works by dividing the stream cluster problem into an online micro-cluster component and an offline macro-clustering component. One drawback of Aggarwal *et al.'s* approach is that the number of clusters has to be either known in advanced and fixed, or chosen by a user in each step, which means that human supervision has to be involved in the process.

Another well-known approach in stream clustering is StreamKM++ [7]. This approach is based on k-means++ [4]. However, again the number of clusters needs to be known beforehand. StreamKM++ constructs and maintains a core-set representing the data stream. After the data stream is processed, the core-set is clustered with k-means++. Because of that, StreamKM++ is not designed for evolving data streams.

There are several approaches to deal with the problem of identifying how many different clusters can be found in a data set. Chiang and Mirkin [8] have conducted an experimental study in which they proposed a new method called ik-means that chooses the right k with seven other approaches. In their experiment the method which performs best in terms of choosing the number of clusters and cluster recovery works as follows: Clusters are chosen based on new anomalies in the data and a threshold based on Hartigans rule [9] is used to eliminate small superfluous clusters.

DenSream was introduced by Cao *et al.* [10] to cluster streaming data under the conditions of changing data distributions and noise in data streams. DenStream creates and maintains dense micro-clusters in an online process. Whenever a clustering request is issued, a macro-cluster method (e.g. DBSCAN [11]) is used to compute the final cluster result on top of the micro-cluster centroids.

It should be noted that in Chiang and Mirkins experimental setting only uses data generated from clusters with a Gaussian distribution. We argue that data from the real-world not necessarily follows a Gaussian distribution. There are a large range of distributions which might fit the data better in different environments and applications such as Cauchy, exponential or triangular distributions. In order to reflect this, our selection criteria for the number of clusters is the shape of the data distribution of the different data features.

Transferring the cluster problem from a fixed environment to streaming data brings another dimension into play for interpreting the data. This dimension is the situation in which the data is produced. For our purpose we define situation as the way the data is distributed in the data stream combined with statistical properties of the data stream in a time frame. This situation depends both on the location and time. For example, categorising outdoor temperature readings into three different categories (i.e. cold, average, warm) is heavily dependent on the location, e.g. on the proximity to the equator. For example, what is considered hot weather in the UK is perceived differently somewhere in the Caribbean.

Similarly our interpretation of data can change when we fix the location but look at measurements taken at different points in time. For example, consider a temperature reading of 10° in the UK. If this measurement was taken in winter, we certainly consider this as warm. If it was taken in summer though it would be considered as cold temperature. This phenomenon where the same input data leads to a different outcome in the output is known as concept drift [12], [13].

There are several existing methods and solution focusing on concept drift; some of the recent works in this domain are reviewed in [14]. Over the last decade a lot of research was dedicated to handling concept drift in supervised learning scenarios mainly utilising decision trees [15] or ensemble classifiers [16]; however adaptation mechanisms in unsupervised methods have only recently started to be investigated [14].

There are different types of concept drift. If only the data distribution changes without any effect on the output, it is called virtual drift. Real concept drift denotes cases where the output for the same input changes. This usually has one of the following reasons. Either the perception of the categories or objectives has changed or changes in the outcome are triggered by changes in the data distribution. We argue that in the IoT domain and especially in smart city applications, the latter type of concept drift is more important. In order to avoid confusion between the different types of concept drift we introduce the term "data drift" to describe real concept drift that is caused by changes in the data stream.

Smith *et al.* [17] have developed a tool for creating data streams with data drifts through human interactions. In their experiments they have found that current unsupervised adaptation techniques such as Near Centroid Classifier (NCC) [18] can fall victim to cyclic mislabelling, rendering the clustering results useless. While Smith *et al.* [17] found that semi-supervised (Semi-supervised NCC (SNCC)) and hybrid adaptations of the technique (Semi- and Nearest Centroid Classifier (HNCC)) lead to more robust results; adaptive methods are also needed in scenarios for which labels are not available and therefore only unsupervised learning can be applied.

Cabanes *et al.* [19] introduce a method which constructs a synthetic representation of the data stream from which the data distribution can be estimated. Using a dissimilarity measure for comparing the data distributions, they are able to identify data drifts in the input streams. Their work is limited by the fact that they only present preliminary results and are still working on an adaptive version of their approach.

Estimating the data distribution is an essential step for identifying and handling data drifts. The data distribution can be calculated using Kernel Density Estimation (KDE) [20], [21]. The most important parameter for KDE is the bandwidth selection. There are different methods to choose this parameter automatically from the provided data. They include computationally light rules of thumb such as Scotts rule [22] and Silvermans rule [23] and computationally heavy methods such as cross-validation [24]. A detailed survey on bandwidth selection for KDE is provided in [25]. However, the easily computed rules are sufficient for most practical purposes.

Bifet et al. [26] introduced Massive Online Analysis (MOA), a framework for analysing evolving data streams with a broad range of techniques implemented for stream learning. Initially MOA only supported methods for stream classification; extensions of the framework have added additional functionalities for data stream analytics. Particularly interesting for the presented work is an extension of the framework which provides an easy way to compare different stream clustering algorithms. In addition to providing implementation of state-of-the-art stream clustering algorithms and evaluation measures, Kranen *et al.* [27] introduce new data generators for evolving streams based on randomised Radial Base Functions (randomRBFGenerator). We compare our method against their implementations of DenStream [10] and CluStream [6] which are both designed to handle evolving data streams.

Stream cluster algorithms such as CluStream [6] and DenStream [10] stay adaptive to evolving data streams by splitting the clustering into offline and online parts. The online part continuously retrieves a representation of the data stream. This is done through the computation of micro-clusters. The micro-clusters allow for efficient and accurate computations of clusters by applying common clustering methods such as k-means++, DBSCAN [11] or similar methods as a macro cluster whenever a cluster request is issued by the end-user or an application which uses the stream clustering mechanism. This means that the actual clusters and the labels for the data items are only computed when a clustering request on the data stream is made.

This approach works in scenarios where the clustering result is not needed continuously. However, if the clustering result is needed on a continuous basis and the offline calculation of the data stream representation has to be issued in high frequency, the efficiency gain of the methods are lost and the response time in online applications with large volumes of dynamic data is limited by applying the macro clusters. Therefore a new method with low computational complexity that can produce cluster results directly during processing the stream is required. Our proposed solution to this problem is to create a clustering mechanism in which the centroids change and adapt to data drifts. We propose an adaptive method to re-calibrate and adjust the centroids.

A. Silhouette coefficient

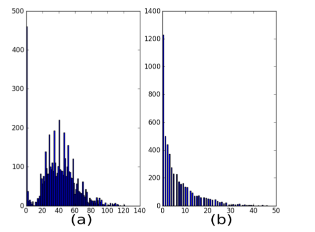
The common metrics to evaluate the performance of clustering such as homogeneity, completeness and v-measure are mainly suitable for offline and static clustering methods where a ground truth in the form of class labels is available. However, in our method, as the centroids are adapted with the data drifts, the latter metrics will not provide an accurate view of the performance. In order to measure the effectiveness of our method we use the silhouette metric. The use of the silhouette coefficient as a criterion to choose the right value for number of clusters has been proposed by Pravilovic *et al.* [28]. This metric is used in various works to measure the performance of the clustering methods including the MOA framework [26], [27] that is used in this work for the evaluation and comparisons.

The silhouette metric as a quality measure for clustering algorithms was initially proposed by Rousseeuw [29]. Intuitively it computes how well each data point fits into its assigned cluster compared to how well it would fit into the next best cluster (i.e. the cluster with the second smallest distance).

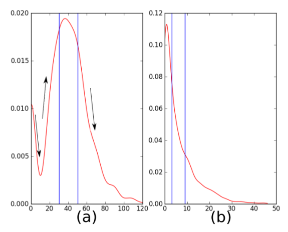
$$s(i) = \frac{b(i) - a(i)}{max((a(i), b(i))} \tag{1}$$

The silhouette for one data points is defined in Equation 1, whereby i represents the data point, $b(i)$ is the average distance to each of the points in the next best cluster and $a(i)$ is the average distance to each of the points of the assigned cluster. The total silhouette score is obtained by taking the average of all s(i). From this definition we can see that the silhouette width $s(i)$ is always between -1 and 1. The interpretation of the values is as follows: values closer to 1 represent better categorisation of the data point to the assigned cluster, while a value close to -1 denotes less efficiency in the categorisation, i.e. the data point would have better fit into the next-nearest cluster. Following that a silhouette width of 0 is neutral, that is to say a data point with this value would fit equally well in both clusters. We average over all silhouette values $s(i)$ to obtain the score of the overall clustering. This average value is the overall silhouette score and can be used to compare the quality of different cluster results.

Rousseeuw [29] points out that single, strong outliers can lead to misleading results; therefore it has to be made sure that



(a) Distribution of the features "average speed" (a) and "number of cars" (b)



(b) PDF of the features "average speed" (a) and "number of cars" (b) split into equi-probable areas

Fig. 1: Distribution of the different features and their resulting PDFs

there are no singleton clusters in the results. In order to use the Silhouette Coefficient in a streaming setting, we have to define the time frame from which the data points are taken into account for measuring the cluster quality. A natural contender for that time frame is the last time the centroids have been recalculated, since this is the point in time when we discovered a data drift and the new clustering has to adapt to the data stream from then on.

III. Adaptive Streaming Clustering

Most of the stream clustering methods need to know in advance how many clusters can be found within a data stream or at the very least are dependent on different parametrisation for their outcome. However, we are dealing with dynamic environments where the distribution of data streams can change over time. There is a need for adaptive clustering algorithms that adapt their parameters and the way they cluster the data based on changes of the data stream.

With the abundance of data produced, one of the main questions is not only what to do with the data but also what possibilities have not been yet considered. If new insights are obtained from the data these can in turn inspire new applications and services. One can even go further and ignore any prior knowledge and assumptions (e.g. type of data categories of results) in order to retrieve such insights. However, previously known knowledge can influence the expectations and therefore can enhance and/or alter the results.

A. Finding the right number of clusters

One of the key problems in working with unknown data is how to determine the number of clusters that can be found in different segments of the data. We propose that the distribution of the data can give good indications of the categories. Since usually the data has several features we look at each of the distributions of different features. The shape of the probability distribution curve gives good approximations of how many clusters we need to group the data.

Figure 1a shows the distribution of two different features (average speed and number of cars) from the use case described in Section V-B. Figure 1b shows the Probability Density Functions (PDFs) that were computed using KDE [20], [21] from the data shown in Figure 1b. We follow the intuition that a directional change, referred to as a turning point (tp), in the probability distribution can signify the beginning of a new category. The tps which ended up producing the best cluster result are visualised as arrows in Figure 1b. We split the PDF in areas of equi-probable distributions as visualised by the blue vertical lines. This idea is inspired by the Symbolic Aggregate Approximation (SAX) algorithm [30] where a Gaussian distribution is split into equi-probable areas that are used to map continuous data from streams to discrete symbolised representations. Following this approach we can obtain smaller and denser cluster in areas with many data points, whereas in areas with less data points we get wider and sparser cluster.

Following that the number of areas in the PDF can be considered as a possible k for the k-means algorithm, the centres of these areas are then considered as possible initial centroids. In contrary to the random initialisation of the original k-means [1] we propose a way to intelligently select the initial centroids. This makes the clustering part of the algorithm deterministic and random restarts become unnecessary.

Since in general, different features of a data stream do not follow the same distribution, the PDF curves obtained from different features contain more than one possible number for k and also provide different candidates for the centroids even if k happens to have the same value. Furthermore, the combination of the different feature distributions could also allow for combinations of optimal clusters which lie between the minimum and maximum number of turning points of the distribution functions. We test for:

$$k \in [tp_{min}, tp_{min} + tp_{max}] \tag{2}$$

How can we then decide which of these value of k and which centroid candidates lead to a better cluster results? In order to answer this question we need a metric with which we can compare the resulting clusters for different values of k when we apply the clustering mechanism to an initial sample set of size n. The metric must satisfy the following properties:

1) independence of k
2) independence of knowledge of any possible ground truth

Property one comes as no surprise. Since we have to compare cluster results with different k values, the metric must not be biased by the number of k's. For instance this would be the case if we chose variance as a comparison criterion. In this case there would be a strong bias towards higher values of k since the variance within the clusters converges to zero as the number of clusters converges to the sample set size.

The second property is derived from the fact that our approach does not take prior knowledge into consideration. On one hand the approach is domain independent. On the other hand one of the main objectives is to extract information which is inherent in the data itself, and therefore should not be obstructed by assumptions of any kind. This property instantly excludes the majority of commonly used metrics for cluster evaluation. Evaluation criteria such as purity, homogeneity and completeness all evaluate the quality of the clusters by comparing the assigned labels of the data to the labels of the ground truth in one way or another.

The silhouette metric described in Section II-A satisfies both properties. In order to estimate the quality of the clusters, they are examined by computing how well the data points fit into their respective cluster in comparison to the next-nearest neighbour cluster.

B. Dealing with the data drift

In scenarios where the input data is fixed, once the k-means algorithm with random restarts converges, the clusters become fixed. The resulting clusters can be then reused to categorise similar data sets.

However in the case of streaming data two observation which are taken on two different (and widespread) time points do not necessary have the same meaning and consequently will belong to different clusters. This in turn leads to a different clustering of the two observations. Identical data can have different meaning when produced in a different situation. For example, imagine observing 50 people at 3pm during a weekday walking over a university campus. This would not be considered as ”busy” given the usual number of students in the area. Observing the same number of people at 3am would however be considered as exceptional, giving the indication that a special event is happening.

We incorporate this into our clustering mechanism by adapting the centroids of the clusters based on the current distribution in the data stream. The data drift detection is triggered by changes in the statistical properties of the probability density function. The justification for our method is based on properties of stochastic convergence. Convergence in mean square (see equation 3) implies convergence in probability, which in turn implies convergence in probability and distribution [31]. The formula for convergence in the mean square is given in equation 3.

$$\lim_{n\to\infty} E|Xn - X|^2 = 0 \tag{3}$$

During training we store the standard deviation and expected value of the data with the current distribution. When processing new incoming data, we track how the expected value and standard deviation changes given the new values. Equation 3 states that as a sequence approaches infinite length, its mean squared approaches the value of the random variable X (in our case defined by the previously computed PDF). However, we can make the assumption that as we get more and more values, that if the expected value converges to such that $E|Xn - X|^2 = \varepsilon$ for $n >> 0$ and $\varepsilon > 0$, the current time series data is no longer converging to the distribution that we predicted with the PDF estimation. Therefore we have a change in the underlying distribution of the data stream and trigger a recalibration of our methods. If we can observe a higher quality in the new clusters, the old centroids will be adjusted.

A detailed description of the algorithm follows in the next Section.

IV. ADAPTIVE STREAMING k-MEANS: A CLOSER LOOK

Algorithm 1 shows how the centroids in our adaptive method are computed. This takes place after a configurable initialisation period and is repeated at the beginning of each adjustment step. More information about the data collection and the adjustment can be found in Section V-B. Initially the probability density functions of each of the features of the data are computed using KDE [20], [21]. The continuous PDFs are represented by discrete arrays.

Algorithm 1 DETERMINECENTROIDS(A, k, n)

```
Require: Data matrix $A = \{a_0, a_1, \ldots, a_n\}$ with each $a_i$ being an array of length $m$ containing all values of feature n
 1: %Therefore sample j is the data point: $[a_0[j], \ldots, a_n[j]]$
Ensure: List $C = \{c_0, c_1, \ldots, c_{max(tps)}\}$ of clusterings, with each $c_i$ being a list of centroids with length $k = tp_{min} + i$
 2: $pdf[] = \emptyset$
 3: $tps[] = \emptyset$
 4: for $i \leftarrow 1$ to $n$ do
 5:     %Array containing the PDFs of each feature
 6:     $pdf[i] = gaussianKDE(a[i])$
 7:     %Array containing the number of turning points of the PDF
 8:     $tps[i] = countTurningPoints(pdf[i])$
 9: end for
10: $C[] = \emptyset$
11: for $i$ in $range(min(tps), min(tps) + max(tps)$ do
12:     $betas[] = \emptyset$
13:     %Each $f$ represents the PDF of a feature
14:     for $f$ in $pdf$ do
15:         $betas[] = findBetas(f, tps[i])$
16:         $C[i] =$ list of means between two adjacent betas
17:     end for
18: end for
19: return $C$
```

These PDF representations are then fed into Algorithm 2. Turning points can be determined by analysing the first derivative. They have the property that $dy/dx = 0$, where dx is the difference between two infinitely close x values of the PDF, dy is the difference between two infinitely close y values of the PDF and dy/dx is the slope of the PDF. This is a necessary but not sufficient criteria for having a turning point. Only if the sign of dy/dx changes from negative to positive or vice versa, we actually have a turning point in our function. These are just the definitions for local maximum and minimum points respectively. Finding these points we can present the turning points of a feature PDF and this number can be used to determine the right number of clusters.

We use the heuristic that the right amount of clusters lies between the smallest number of turning points of a feature PDF and this number added to the maximum number of turning points found in any of the PDFs.

Once the number of turning points - and therefore the possible values for the number of clusters - for each feature are computed, Algorithm 1 determines candidates for the initial centroids.

The computation then splits the PDF curve into equi-probable areas (similar to the SAX algorithm [30]). The boundaries of these areas are called beta points. Since we are interested in the centre of these region, the middle points between two adjacent betas are computed and saved as initial centroids.

Algorithm 2 COUNTTURNINGPOINTS(f)

```
Require: Array $f$ representing a probability density distribution
Ensure: Number of turning points $tps$
 1: $\Delta_y = \varepsilon_y * max(f)$
 2: % Little gradient should be recognised as no gradient
 3: $\Delta_x = \varepsilon_x * max(f)$
 4: % Areas of no ascent/descent should only be counted if
    they last long enough
 5: for $x$ in $f$ do
 6:     if $changedDirection(x, f, \Delta_y, \Delta_x)$ then
 7:         $tps++$
 8:     end if
 9: end for
10: return $tps$
```

Once the candidates for the initial clusters are identified - one for each feature - a normal k-means is run on the dataset with the initial centroids as starting points for the clustering. The results are then compared by computing the silhouettes. For an in-depth description we refer the reader to the paper by Rousseeuw [29]; however a short elaboration on the mathematical background is also given in Section II-A.
Incoming data points are fed into the k-means with the current cluster centroids and assigned to their nearest cluster. The cluster centroid of the assigned cluster is adjusted to reflect the new centre of the cluster including the inserted value.

We give a brief complexity analysis of Algorithms 1 through 3. We start with Algorithm 2, since it is used by Algorithm 1. Since the Algorithm goes along the array f, which represents the PDF, the complexity lies in $\mathcal{O}(length(f))$. This array has exactly the same length as the array which has been fed into the function $gaussianKDE$, computing the pdf representation. Algorithm 1 is called with a matrix of dimensions $n \times l$. Here n is the number of features, l being the length of the initial data sequence. Therefore $length(f) = l$ and we have a complexity of $\mathcal{O}(l)$ for Algorithm 2. At the same time, the $gaussianKDE$ function scales linearly with the size of the input array, resulting as well in the complexity of $\mathcal{O}(l)$.
For Algorithm 1 we first look at line 3 to 6. Here for each of the n feature vectors both $gaussianKDE$ and Algorithm 2 are called. This results in a complexity of $\mathcal{O}(n \cdot l)$.
We then examine line 8 to 14. We can see that the outer for loop runs exactly $max(tps)$ times, which in practice is a small constant. The inner for loop runs in the length of the number of the PDFs; since we compute one PDF for each feature this is equal to n times. Function $findBetas$ goes along the input array to find the beta values and therefore scales in the length of the input array. Putting this information together results again in a complexity of $\mathcal{O}(n \cdot l)$. Since both parts of the Algorithm have the same complexity, the total complexity equals $\mathcal{O}(n \cdot l)$.
Algorithm 3 uses Algorithm 1 for finding the initial centroids. Running k-means with determined initial centroids takes $\mathcal{O}(nkd)$ since no iterations of the algorithm are needed. Then for each clustering we compute the silhouette score. Calculating the silhouette score is computationally intensive since each distance pair has to be computed. Here we can apply the following steps to increase the performance. Instead of calculating the distance pairs for each value of k, we can initially compute the distance pair matrix and pass it to the silhouette calculation. For cases where we have a huge size of data values, we perform random sampling to decrease the number of distances pairs that have to be computed. In practice, sampling has been shown to provide close approximations to the actual silhouette score at a fraction of the computational cost. During the online clustering phase, assigning the nearest cluster to new incoming values takes only $\mathcal{O}(1)$ time. Recalibrating the cluster centroids requires one call of Algorithm 1 ($\mathcal{O}(n \cdot l)$) and another run of k-means ($\mathcal{O}(nkd)$).

Algorithm 3 STREAMINGKMEANS(D)

```
Require: Data stream $D$, length of data sequence used for
    initialisation $l$
Ensure: Continuous clustering of the data input stream
 1: % initialisation phase
 2: for $cCs$ in $determineCentroids(D.next(l))$ do
 3:     $current_k = length(cCs)$
 4:     $nCs = kmeans(cCs, D.next(l), current_k))$
 5:     if $silhoutte(nCs) < lastSil$ then
 6:         % We found a new best clustering
 7:         $centroids = nCs$
 8:         $lastSil = silhoutte(nCs)$
 9:         $k = current_k$
10:     end if
11: end for
12: % Continuous clustering phase
13: loop
14:     if $changeDetected(D)$ then
15:         $centroids = determineCentroids()$
16:         $centroids =$
17:             $kmeans(centroids, D.getData(), k)$
18:     else
19:         $centroids =$
20:             $kmeans(centroids, D.getData(), k)$
21:     end if
22: end loop
```

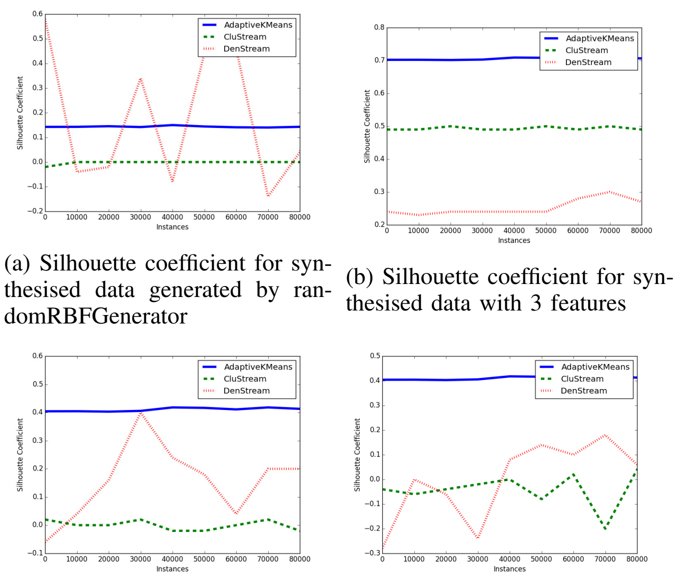
V. EVALUATION

We test the proposed method both on synthesised data and on a real-world data set. We evaluate the method against state-of-the-art methods using data sets which are generated in different ways and discuss in which cases the method has an advantage over existing approaches and in which cases it is outperformed by them.
We introduce a novel way of generating data streams with data drift. The drift is introduced both by shifting the centroids in randomised intervals and by changing the data distribution function used to randomly draw the data from the centroids.

Here we can scale up the dimensionality of the generated data, with each feature having its own distribution to draw the data from.
Finally, we show how our method can be used in a real-world case study by applying it to the output of traffic sensors which measure the average speed and the number of cars in street segments.

A. Synthesised Data

To evaluate our method we test our method against two established stream cluster algorithms: CluStream [6] ($horizon = 1000$, $maxNumKernels = 100$, $kernelRadiFactor = 2$) and DenStream [10] ($horizon = 1000$, $epsilon = 0.02$, $beta = 0.2$, $mu = 1$, $initPoints = 1000$, $offline = 2$, $lambda = 0.25$) . For this we use two different ways of generating data streams with data drift. The first one, randomRBFGenerator, was introduced by Kranen *et al.* [27]. Given an initial fixed number of centroids and dimension, the centroids are randomly generated and assigned with a standard deviation and a weight. New samples are generated as follows: Using the weight of the centroids as a selection criteria, one of the centroids is picked. By choosing a random direction, the new sample is offset by a vector of length drawn from a Gaussian distribution with the standard deviation of the centroid. This creates clusters of varying densities. Each time a sample is drawn, the centroids are moved with a constant speed, initialised by an additional parameter, creating the data drift.



(a) Silhouette coefficient for synthesised data generated by randomRBFGenerator

(b) Silhouette coefficient for synthesised data with 3 features

(c) Silhouette coefficient for synthesised data with 4 features

(d) Silhouette coefficient for synthesised data with 5 features

Fig. 2: Silhouette coefficient comparison on synthetic data sets

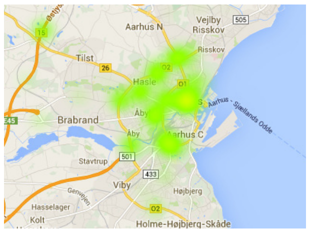
This however, has a drawback that the data drift is not natural, as the centroids are constantly shifting. We argue that during a short time frame, the data stream roughly follows a certain distribution. The underlying distribution can then change between time-frames, triggered by situational changes. These changes can be re-occurring in time (for example in the case of traffic during rush hours and off-peak times) or more sudden changes (for example traffic congestions caused by an accident).
For that reason we introduce a novel way of generating data with data drift. The centroids are selected through Latin Hypercube Sampling (LHS) [32]. The number of clusters and dimensions are fixed beforehand. Similar to the method before, each centroid is assigned with a standard deviation and weight. Furthermore, each dimension is given a distribution function, which later is used to generate the data samples. Considering that each dimension represents a feature of a data stream, this models the fact that in IoT applications we are dealing with largely heterogeneous data streams in which the features do not follow the same data distribution. Our current implementation supports triangular, Gaussian, exponential, and Cauchy distributions. The implementation is easily expandable and can support other common or custom distributions. The data generation code is available via our website at: http://iot.ee.surrey.ac.uk/
Data drift is added sporadically and is independent for each dimension through two different ways. The first is a directional change of random length. The second is that over the course of generating the data, the data distribution used for the dimension is changed for one or more of the dimensions. Both changes appear in random intervals.
We compare our method against CluStream and DenStream. Figure 2a shows the performance on data generated by the randomRBFGenerator with data drift. The results for the data generated by the introduced novel way with different number of features are shown in Figures 2b to 2d. 100 centroids have been used for the data generation. For the visualisation, the silhouette score has been normalised to a range between 0 and 1 as done within the MOA framework[1].
On the data produced by the randomRBFGenerator, our novel method constantly outperforms CluStream by around 13%. DenStream performs better at times, however the silhouette score of DenStream drops below the levels of CluStream at times, suggesting that the method does not adapt consistently to the drift within the data. As seen in Figures 4 to 7, for the synthesised data with different number of features, our novel method constantly perform around 40% better than CluStream and more than 280% than DenStream.

B. Case Study: Real-Time Traffic Data

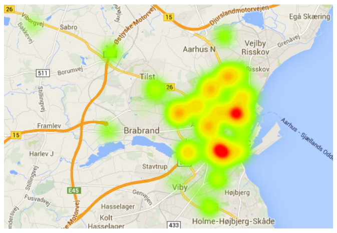
To showcase how our approach can be applied to real-world scenarios, we use (near-)real-time traffic data from the city of Aarhus [2]. 449 Traffic sensors are deployed in the city which produce new values every five minutes. The data is pulled and fed into an implementation of our clustering algorithm that is described Section IV. Before the value of k is computed and the initial centroids are determined, the data is collected for one hour which equates to 5035 data points. The main data is collected for a period of 5 months. Over the course of one

[1]http://www.cs.waikato.ac.nz/\~abifet/MOA/API/_silhouette_coefficient_8java_source.html

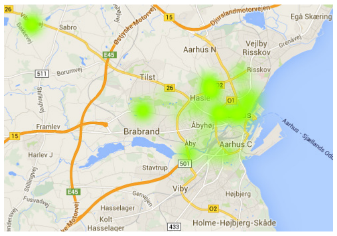
[2]http://www.odaa.dk/dataset/realtids-trafikdata



(a) Traffic Density Morning



(b) Traffic Density Noon



(c) Traffic Density Evening

Fig. 3: Traffic densities at different times in Aarhus

Number of Cluster	Silhouette Coefficient
3	**0.450828**
4	0.361470
5	0.336280
6	0.409701

Fig. 4: Silhouette Coefficients

day, 122787 samples are collected. For the clustering we use *number of cars* and *average speed* measured by the sensors. For the purpose of evaluation and visualisation, a timestamp and location information are also added to each data point. The intuition is that the clustering of traffic data is dependent on the time of day and the location. This perspective allows us to cluster incoming data in a way to better reflect the current situation. Figures 3a, 3b and 3c visualise the traffic density as a heat map in the morning, at noon and in the evening in central Aarhus respectively. Light green areas in the map show a low traffic density, while red areas indicate a high density. No colouring means that there are no sensors nearby. In the morning (Figure 3a) there is only moderate traffic density spread around the city. At noon (Figure 3b) we can see that two big centres of very high density (strong red hue) have emerged. Figure 3c shows that in the evening there is now only very low to moderate traffic density in the city.

Several reasons for data shift on varying degrees of temporal granularity are conceivable. During the day the density of traffic changes according to the time. In rush hours where people are simultaneously trying to get to or back from work, the data distribution of traffic data differs greatly from less busy times during working hours or at night. Because of the same reasons, the data distribution is also quite different in weekends as in weekdays. During holidays, e.g. around Christmas or Easter, the dynamics of traffic can change a great deal. All these changes in the traffic data distribution lead to a need of reconsidering what can be categorised as a ”busy” or ”quiet” street, in other words we are dealing with data drift in these cases, as the same input leads to different output at different times.

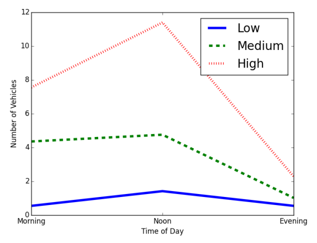
Our approach deals with the data drift by re-calculating the centroids based on the current distribution. This means that defining a street as ”busy” or ”quiet” is relative and depends on the time of the day and the location. For example, 15 cars at a given time in one street could mean ”busy” while in another situation it could mean ”very quiet”. Similarly 15 cars in the city centre can have a different meaning during the day than at night.

We use a ring buffer as a data cache, that captures the data produced in the last hour. Whenever a re-calculation process is triggered based on a detected data drift because the data no longer converges to the mean square (see Section III-B), we use the silhouette coefficient score to check if the new centroids lead to a better cluster quality. If that is the case, the current centroids are then adjusted. The definition of the silhouette coefficient and its computations can be found in Section II-A.

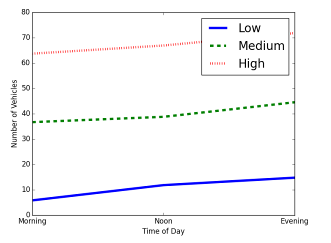
Figure 4 shows the computed silhouette coefficients for clustering the initial data sequence with different number of clusters. The number of clusters is chosen according to the highest value of the coefficient and is emphasised in bold. For performance reasons, a sampling size of 1000 has been chosen to compute these values. It can be decided based on the data input and the task at hand if the number of clusters should stay constant through the remainder of clustering the data. In our use case scenario, we do not change the number of clusters.

Figures 5 shows how the centroids of the clusters change at different times on a working day and on a Saturday for the different re-calculation times. The way the data is clustered differs significantly between the two days. While the average speed remains roughly the same, the amount of vehicles varies a lot. Most prominently this difference can be seen in the centroids of the cluster representing high number of cars. For example, Figure in 5a the number of cars is considerably higher at noon than in the evening. Resulting from the changed centroids, data points may be assigned to different clusters depending on the time compared to using non-adaptive clustering. For example, using the centroids on working day at noon on the data published during the same time on Saturday, 180 out of 3142 values would be assigned to a different cluster.

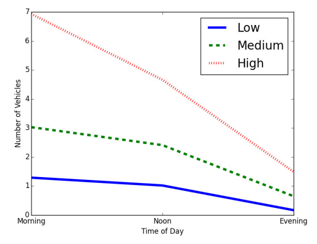
In order to interpret how busy an area is, it is necessary to also take into consideration all adjacent time points in that area. Therefore the output of our approach can be used to further abstract from the data by matching patterns within a time frame in an area. The results can be fed into an event detection and interpretation method. To clarify the reasoning behind this, two examples are given. Let’s consider a measurement of a fast moving car. Only if the measurements in close time range are similar, we can interpret this traffic as a good traffic flow. If this is not the case, a different event has taken place, e. maybe an ambulance rushed through the



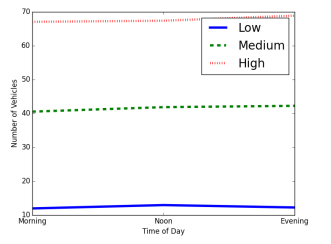
(a) Cluster centroids for number of cars on a working day



(b) Cluster centroids for average speed on a working day



(c) Cluster centroids for number of cars on a Saturday



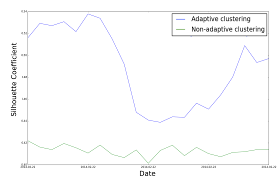
(d) Cluster centroids for average speed on a Saturday

Fig. 5: Centroids adapting to changes in the data stream. Low, medium and high refer to the level of traffic density the cluster is representing.

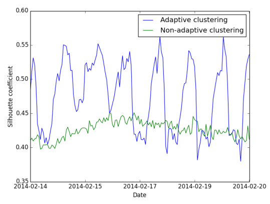
street.

Another example would be the measurement of slow moving cars. If this is a singular measurement, it could mean for example that somebody is driving slow because s/he is searching for a parking space. However if the same measurement accumulates it could mean that a traffic jam is taking place.

In order to ensure that the adaptive method leads to meaningful results we have conducted another experiment. We compare our adaptive stream clustering method with a non-adaptive streaming version of the k-means algorithm, i.e. the centroids are never re-calculated. The silhouette coefficient of the clusters in both setting are computed in equal time intervals. Figure 6a shows how the silhouette coefficients compare over the course of one day. For example, at 22/02/2014, 05:30:00 the non-adaptive approach scores a silhouette coefficient of 0.410 while the adaptive approach scores 0.538, an improvement of 31.2%. This means items clustered by the adaptive methods have a better convergence considering the distribution of the data at that time.



(a) Silhouette coefficient over the course of one day based on hourly measures



(b) Silhouette coefficient over the course of one week based on hourly measures

Fig. 6: Silhouette coefficient on traffic data set

Figure 6b shows how the silhouette coeeficients compare over the course of one week. The adaptive clustering performs mainly better than the non-adaptive. The cluster quality of the adaptive solution follows a daily pattern. During the day the quality drops to levels of the non-adaptive solution. This can be explained through the fact that during the night the traffic flow is more clear cut, for example there are more cases where there is no traffic at all. The adaptive solution is able to exploit this fact and adapt itself to produce better clusters that are closer to the actual categories that can be found in the data streams. During the day, there are many more data samples on the edge of clusters that could be clustered into either one of adjacent clusters, leading to a worse silhouette coefficient score. Here the quality of the adaptive clustering at times drops to values near the quality of the non-adaptive. However, the next iteration of the adaptive clustering improves the cluster quality again automatically based on changes in the data distribution. At some points in time it drops below the quality of the non-adaptive one: in these cases the quality quickly recovers to better values again. Overall the mean of the silhouette coefficient is 0.41 in the non-adaptive setting and 0.463 in the adaptive setting which translates to an average improvement of 12.2% in cluster quality.

VI. Conclusion

In this paper we have introduced an adaptive clustering method that is designed for dynamic IoT data streams. The method adapts to data drifts of the underlying data streams. The proposed method is also able to determine the number of categories found inherently in the data stream based on the data distribution and a cluster quality measure. The adaptive method works without prior knowledge and is able to discover inherent categories from the data streams.

We have conducted a set of experiments using synthesised data and data taken from an traffic use-case scenario where we analyse traffic measurements from the city of Aarhus. We run adaptive stream clustering method and compare it against a non-adaptive stream cluster algorithm. Overall the clusters produced using an adaptive setting have an average improvement of 12.2% in the cluster quality metric (i.e. silhouette coefficient) over the clusters produced using a non-adaptive setting.

Compared to state-of-the-art stream cluster methods, our novel approach shows significant improvements on synthesised data sets: Against CluStream there are performance improvements between 13% and 40%. On data generated by randomRBFgenerator, DenStream has better cluster quality at few points of the experiment, is generally outperformed by our method though. On the other synthesised data streams, our novel approach shows an improvement of more than 280% compared to DenStream.

The results of our clustering method can be used as an input for pattern and event recognition methods and for analysing the real-world streaming data. To clarify our approach we

have used k-means as the underlying clustering mechanism, however the concepts of our approach can also be applied to other clustering methods. For the latter the distribution analysis and cluster update mechanisms can be directly adapted from the current work and only the cluster and centroid adaptation mechanisms should be implemented for other clustering solution.
For the future work we plan to apply the proposed solution to different types of multi-modal data in the IoT domain. We will also investigate the concept drift and clustering updates based on user requirement changes and target changes. In this work we proposed a clustering method designed to deal with drifts in the data. For this we have not considered the spatial dimension of the data. Spatial clustering and auto-correlation are important topics in data mining and we aim to extend our work with solutions to this problem.

Appendix

We have applied our approach on an additional multi-variate, real world data set well-known in stream clustering and classification tasks, the forest cover types data set. The data set was originally introduced by Blackard *et al.* [33] and is available online in the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/Covertype). The forest cover types for 30 x 30 meter cells were obtained from the US Forest Service (USFS) Region 2 Resource Information System (RIS) data, contains 10 continuous variables and has more than 580000 data samples.
Figure 7 shows that while DenStream performs better on average, the cluster quality drops down to misclustering (values below 0.5) at times during the stream clustering. Our approach shows a consistent cluster quality while processing the data stream.

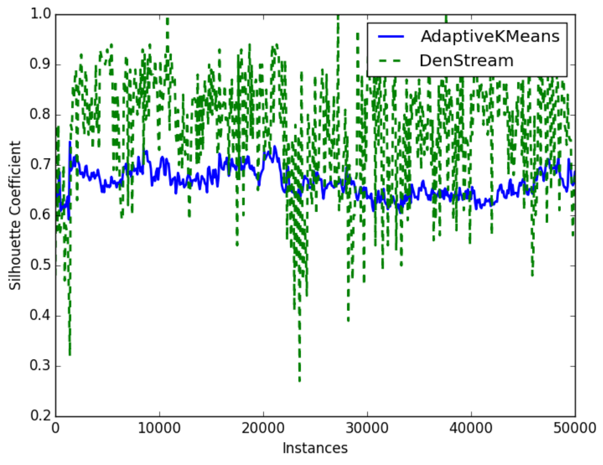


Fig. 7: Forest cover type data set

Acknowledgment

The research leading to these results has received funding from the European Commissions Seventh Framework Programme for the CityPulse project under grant agreement no. 609035.

References

[1] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.
[2] P. S. Bradley, O. L. Mangasarian, and W. N. Street, "Clustering Via Concave Minimization," *Advances in Neural Information Processing Systems*, pp. 368–374, 1997.
[3] L. Xu and D. Schuurmans, "Unsupervised and Semi-Supervised Multi-Class Support Vector Machines," in *Proc. of the 20th National Conference on Artificial Intelligence*, 2005, pp. 904–910.
[4] D. Arthur, "k-Means ++ : The Advantages of Careful Seeding," in *Proc. of the 18th annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics*, 2007, pp. 1027–1035.
[5] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering Data Streams," *Proc. 41st Annual Symposium on Foundations of Computer Science*, pp. 359–366, 2000.
[6] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A Framework for Clustering Evolving Data Streams," in *Proc. of the 29th International Conference on Very Large Data Bases*, 2003, pp. 81–92.
[7] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "StreamKM++: A Clustering Algorithm for Data Streams," *Journal of Experimental Algorithmics*, vol. 17, no. 1, pp. 173–187, 2012.
[8] M. M.-t. Chiang and B. Mirkin, "Intelligent Choice of the Number of Clusters in k-Means Clustering : An Experimental Study with Different Cluster Spreads," *Journal of Classification*, vol. 40, no. 1, pp. 3–40, 2010.
[9] J. A. Harrington, *Clustering Algorithms*. John Wiley and Sons, 1975.
[10] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-Based Clustering over an Evolving Data Stream with Noise," in *Conference on Data Mining*, no. 2, 2006, pp. 328–339.
[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Conference on Knowledge Discovery & Data Mining*, vol. 2, 1996, pp. 226–231.
[12] J. C. Schlimmer and R. H. Granger, "Incremental Learning from Noisy Data," *Machine Learning*, vol. 1, no. 3, pp. 317–354, 1986.
[13] G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.
[14] J. Gama, I. Zliobaite, A. Bifet, M. Pecheniztkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Computing Surveys*, vol. 46, no. 6, 2014.
[15] H. Yang and S. Fong, "Countering the Concept-Drift Problem in Big Data Using iOVFDT," in *IEEE International Congress on Big Data*. Ieee, 2013, pp. 126–132.
[16] D. M. Farid, L. Zhang, A. Hossain, C. M. Rahman, R. Strachan, G. Sexton, and K. Dahal, "An Adaptive Ensemble Classifier for Mining Concept Drifting Data Streams," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5895–5906, 2013.
[17] J. Smith and N. Dulay, "Exploring Concept Drift using Interactive Simulations," *IEEE International Conference on Pervasive Computing and Communications Workshop (PERCOM Workshops)*, pp. 49–54, 2013.
[18] K. Förster, D. Roggen, and G. Tröster, "Unsupervised Classifier Self-Calibration through Repeated Context Occurrences : Is there Robustness against Sensor Displacement to Gain?" in *IEEE International Symposium on Wearable Computers*, 2009, pp. 77–84.
[19] G. Cabanes, Y. Bennani, and N. Grozavu, "Unsupervised Learning for Analyzing the Dynamic Behavior of Online Banking Fraud," *2013 IEEE 13th International Conference on Data Mining Workshops*, pp. 513–520, 2013.
[20] E. Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.
[21] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.
[22] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley and Sons, 1992.
[23] B. W. Silverman, "Density Estimation for Statistics and Data Analysis," in *Density Estimation for Statistics and Data Analysis*, 1986.

[24] D. W. Scott and G. R. Terrell, "Biased and Unbiased Cross-Validation in Density Estimation," *Journal of the American Statistical Association*, vol. 82, no. 400, pp. 1131–1146, 1987.

[25] M. C. Jones, J. S. Marron, and S. J. Sheather, "A Brief Survey of Bandwidth Selection for Density Estimation," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 401–407, 1996.

[26] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive Online Analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[27] P. Kranen, H. Kremer, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer, "Clustering Performance on Evolving Data Streams : Assessing Algorithms and Evaluation Measures within MOA," in *IEEE International Conference on Data Mining Workshops*, 2010.

[28] S. Pravilovic, A. Appice, and D. Malerba, "Integrating Cluster Analysis to the ARIMA Model for Forecasting Geosensor Data," *Foundations of Intelligent Systems*, pp. 234–243, 2014.

[29] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[30] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms," in *Proc. of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003, pp. 2–11.

[31] R. J. Sefling, *Clustering Algorithms*. John Wiley and Sons, 2009, vol. 162.

[32] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[33] J. A. Blackard and D. J. Dean, "Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables," *Computers and Electronics in Agriculture*, vol. 24, no. 3, pp. 131–151, 1999.

Daniel Puschmann is currently pursuing his Ph.D. degree from the Institute for Communication Systems, University of Surrey. His research is focused on information abstraction and extracting actionable information from streaming data produced in the Internet of Things using stream processing and machine learning techniques.

Payam Barnaghi is a Reader at the Institute for Communication Systems Research at the University of Surrey. Hes also the coordinator of the EU FP7 CityPulse project. His research interests include machine learning, the Internet of Things, the Semantic Web, adaptive algorithms, and information search and retrieval. He is a senior member of IEEE and a Fellow of the Higher Education Academy.

Rahim Tafazolli is a professor and the Director of the Institute for Communication Systems, University of Surrey. He has been active in research for more than 20 years, has authored and co-authored more than 360 papers in refereed international journals and conferences. He is the Founder and past Chairman of IET International Conference on 3rd Generation Mobile Communications. He is a Fellow of the IET and WWRF. He is Chairman of EU Expert Group on Mobile Platform (e-mobility SRA) and Chairman of Post-IP working group in e-mobility, past Chairman of WG3. He is a senior member of the IEEE.