Università di Pisa
Dipartimento di Informatica

Technical Report: TR-13-11

Distributed monitoring of cluster quality for car insurance customer segmentation

Mirco Nanni[2], Roberto Trasarti[2], Anna Monreale[1], Valerio Grossi[1], Dino Pedreschi[1]
University of Pisa[1], ISTI-CNR, Pisa[2]

July 21, 2013

Address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Distributed monitoring of cluster quality for car insurance customer segmentation

Mirco Nanni[2], Roberto Trasarti[2], Anna Monreale[1], Valerio Grossi[1], Dino Pedreschi[1]
University of Pisa[1], ISTI-CNR, Pisa[2]

July 21, 2013

Abstract

Customer segmentation is one of the most traditional and valued tasks in customer relationship management (CRM). In this paper, we explore the problem in the context of the car insurance industry, where the mobility behavior of customers plays a key role: different mobility needs, driving habits and skills imply also different requirements (level of coverage provided by the insurance) and risks (of accidents). In the present work, we describe a methodology to extract several indicators describing the *driving profile* of customers, and provide a clustering-oriented instantiation of the segmentation problem, based on such indicators. Then, we consider the availability of a continuous flow of fresh mobility data sent by the circulating vehicles, aiming at keeping our segments constantly up-to-date. We tackle a major scalability issue that emerges in this context when the number of customers is large, namely the communication bottleneck, by proposing and implementing a sophisticated distributed monitoring solution, which reduces the communications between vehicles and company servers to the essential. Finally, we validate the framework on a large database of real mobility data, coming from GPS devices of private cars.

1 Introduction and motivations

A key task in modern customer relationship management (CRM) is to understand the needs of each customer and to devise policies to harmonize them at the best with the company objectives. In most cases that translates into identify a portfolio of *customer profiles*, each representing the needs and requirements of a reasonable number of customers. Then, each profile can be treated separately in order to devise the market strategies and business models that best fit its customers' peculiarities. In the business intelligence field, this process is best known as *customer segmentation*, and is traditionally implemented by applying pre-defined customer classification rules (for instance based on RFM indices:

Recency of last contact with customer, Frequency of transactions, Monetary volume involved in the relation with the customer) or, in more recent times, by using clustering algorithms. This kind of process can be applied to any kind of business focused on services towards several customers, including the classical domain of retail selling as well as e-commerce and many others. In particular, in this paper we explore solutions for customer segmentation that put together several novel aspects: first, we focus on the very actual and rather uncommon ground of car insurance business, where the main features that characterize a customer are related to how he/she drives, and therefore on his/her mobility; second, we instantiate the general problem to the new context by adopting a data mining-oriented approach; third, we tackle the problem from a more realistic *big data* perspective, where the application can benefit from a continuous flow of fresh information. The first point leads us to model the characteristics of the customers through features extracted from his/her mobility. In particular, we analyze the trajectories that describe such mobility based on standard GPS traces, as those recently collected by specialized companies for the car insurance sector. At the second point, we propose a definition of customer segmentation based on the features mentioned above and a clustering procedure, together with a notion of quality of the segmentation that can be used to test the validity of a given segmentation in the context of dynamic data, i.e. when the features describing the customers are updated to reflect their recent behavior. This aspect is connected to the third point mentioned above, related to big data, which implies, on the one hand, that the application can in principle provide a customer segmentation which is kept up-to-date with the recent changes in real time; on the other hand, the huge size and speed of such data requires to adopt proper strategies to minimize the communication and storage requirements, since traditional centralized, off-line analysis methods might be unsustainable in this context.

In this paper we provide solutions that consider all three aspects discussed above, equipped with an extensive experimentation based on a large real dataset of GPS traces. In particular, we address the problem of continuous monitoring of the quality of the clusters (profiles) of similar drivers by adapting and extending the *safe-zones* approach [19], a method that supports the distributed monitoring of global functions while limiting as far as possible the need for communication between the distributed agents and a central server.

2 Problem definition

2.1 Background on trajectory data analysis

The history of a user is represented by the set of points in space and time recorded by their mobility device defined as $H = \langle p_1 \dots p_n \rangle$ where $p_i = (x, y, t)$

and x, y are spatial coordinates and t is an absolute time-point. Starting from this sequence we are interested in extracting the user's trajectories, where a trajectory is a subsequence of points representing the movement between two places where the user stops for an activity. In the literature there are complex techniques for stop computation, but in our experiences a simple cut when using a minimum period of time γ in considering a spatial buffer with radius r is a good trade-off between the computational efficiency and quality of result obtained. A common setup of the parameters in case of private cars is $\gamma = 2hrs.$ and $r = 50m.$ In the following, we will also use the concepts of "the most frequent location $L1$" and "the second most frequent location $L2$". These two areas represent for the user the most attractive places and in the literature, usually are interpreted respectively as *home* and *workplace*.

2.2 Application

For the description of the *user driving behavior* in a time window we identified four categories of measures: (i) *basic*, (ii) *space-time distribution*, (iii) *context-aware*, and (iv) *behavioral*. The first one contains measures describing the *basic* features of the trajectories in the time window such as:

Length: distance travelled by the user.
Duration: time spent travelling by the user.
Count: number of different user's trips.
MaxAcceleration: maximum user's acceleration.
MaxDeceleration: maximum user's deceleration.

This information is directly computable from the raw GPS traces without any complex process. However, they are useful to understand the behavior of the car usage. The second category comprehends more complex measures that capture how the territory is used, both spatially and temporally:

Avg_Dist_L1: average distance of the user from his most frequent location L1.
Radius_g: radius of gyration of the user (i.e. the standard deviation from the center of mass of his movements).
Radius_g_L1: radius of gyration w.r.t. to the user's L1.
TimeL1L2: time spent by the user in L1 or L2.
EntropyLocation: entropy of the location frequencies where the user stops.
EntropyTime: entropy of user's travel time frequencies.

This set of measures describes the spatial and temporal *user distribution movements*. The third category is composed of the *context-aware* features, where information about the user's movement is related to the spatial and temporal context in which he moves:

EntropyArc: entropy of road segment frequencies traversed by the user.
Phighway: distance travelled on highways by the user.
Pcity: distance travelled inside urban areas by the user.
Length_arc_crowded: distance travelled on top 20% most crowded road segments.
Pnight: distance travelled during night time (i.e. between 10 p.m. and 5

a.m.) by the user.
With this category the user is characterized by the different contexts in which he travel during the time window. The last category focuses on capturing some specific behaviors:

PAccelerationDeceleration: percentage of rapid accelerations/decelerations of the user during his movements.

Pover: how much the user drives over the speed limits.

Profile: how much the user follows his profiles, i.e. trips that he performs frequently.

Clustering-based customer segmentation. The indicators built at the previous step provide a summary of all factors deemed relevant to characterize the driving of an individual. On this base, the customer segmentation can be defined essentially in two ways: by means of *user-defined rules* to assign each customer to one out of a set of pre-defined segments; or by applying a *data-driven approach* that looks for the existence of meaningful groups of customers, each group containing individuals with similar feature values. The first solution mimics quite closely the traditional approach to CRM, where a set of golden rules, learned through years of experience or through prestigious studies, is assumed to hold perpetually, virtually unaffected by external factors. The second solution follows a data-mining perspective, and essentially trades the clear understanding of static, well-established golden rules for the capability of capturing the potentially dynamic group structure of customers directly suggested by their mobility indicators.

Quite obviously, the user-defined rules look advantageous in all contexts where such a set of such rules are known and the domain is characterized by a large inertia. In all other cases (no applicable rules are known or the domain is inherently dynamic, thus quickly turning such rules obsolete), the data-driven approach may come to help. Since the characteristics of the domain considered in this paper better fit the latter situation, in our work we will devise a data-driven solution to the customer segmentation problem.

Following well-known precedents in the business intelligence literature [4], we choose to build segments through a clustering algorithm. In selecting the most appropriate clustering schema, we should consider the following requirements of our application: first, each segment should contain customers that are significantly similar to each other, and therefore clusters should be basically compact; second, some customers might not fit well any segment, and therefore the clustering procedure should account outliers, to some extent; third, as the pool of customers might be very large, for instance in the order of several millions, the algorithms need to have a low computational complexity.

The first and last requirements can be met very simply by the standard K-means algorithm, which seeks globular clusters and has an almost linear complexity. In order to account outliers and yet keep the overall procedure efficient, we choose to first apply K-means on the input data, and afterwards apply a postprocessing to remove objects too far from the cluster center. The K-means algorithm starts from a set of random cluster centers (centroids), and then it-

eratively assigns each data object to the closest centroid and then recomputes the centroid of each cluster as the average of its data vectors; the process is repeated until convergence is reached, i.e. the cluster centroids are stable.

As we can see, the only parameter of the method is the number of clusters K. If no (correct) guesses for K are provided by the domain knowledge, a reasonable value can be found by applying standard methods (see, e.g. [15]) that run the clustering with several values for K, and select the largest one such that a further increase in K generates no significant improvement in the clusters compactness (the latter being measured through a quantity called SSE, defined in the next sections for other purposes).

2.3 Customer segmentation on dynamic data

The mobility of individuals is a phenomenon that, in principle, can highly change with time. For instance, some routines might be affected by sudden and temporary variations of the environment (special events or road works forcing people to adapt their daily routes), or they might be influenced by seasonal factors (actually, the whole lifestyle might change from winter to summer). Finally, a change in the mobility might be simply an effect of the natural evolution of individual lives, possibly involving changing working conditions, family status and taste on how to enjoy the spare time – cases in which the variation of the individual mobility is more likely to last relatively long.

For these reasons, in our context it is advisable to have mechanisms that ensure a good fit between the actual segmentation (the one that the company is using to shape its business) and the real mobility behavior of the customers.

Assuming to have access to a stream of continuously updated indicators for all our customers, the first question to tackle is the following: how and when the changes detected on single users should translate into changes in the customer segmentation? Indeed, some individual changes might be small enough to have negligible effects on the corresponding segments, or at least not to impact on the overall structure of segments. In these cases, it is advisable to simply keep the known segmentation, avoiding the practical troubles at the management level that would result from an excessively frequent redefinition of the segments.

In order to implement the general principles mentioned above, we follow an approach where the last computed segmentation is kept as long as some quality requirements are satisfied, which are defined and discussed below in some detail. Then, whenever such requirements are violated, the existing segmentation is discarded, and a new one is computed from the most recent data.

In the remaining of this section we provide the elements for redefining the customer segmentation in a dynamic context following the ideas mentioned above. We do that assuming to work in a centralized setting where it is possible to store and analyze all the mobility data streaming in. In Section 3 we will move to a more realistic setting, where the communication issues (inevitable on a large scale application like the one we are developing) are considered and dealt with.

Quality measure. A simple and very popular method for measuring the overall quality of a clustering is the so-called *Sum of Squared Error* (SSE in short), defined as follows:

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} ||p - c_i||_2^2 \tag{1}$$

where C_i represents the i-th cluster, and c_i is its center (average vector). This measure evaluates the dispersion of each cluster around its centroid, and therefore gives emphasis to the compactness of clusters. Having dispersed clusters does not necessarily mean that they are not meaningful or that they do not capture well the cluster structure hidden in the data. However, that might affect the reliability of a centroid as a representative of all the objects in the cluster, and the cluster might lack the homogeneity that was a requirement of our initial problem – segmenting customers into groups having homogeneous needs. For this reason the SSE measure seems to fit properly with our context, and will later be applied in our application.

We also notice that the use of SSE is very coherent with the algorithm adopted for clustering. Indeed, it is easy to see that K-means tries to reduce the SSE at each step of its iterative process, stopping when a local optimum is reached.

Monitoring formulation of the problem. Our approach to deal with dynamic data consists in continuously checking whether the last customer segmentation computed is still good enough, recomputing the segments only in the negative case. This requirement can be easily translated in terms of SSE by asking that the dispersion of the objects within the clusters did not grow, or at least not significantly. That means computing the SSE at each time stamp t, which we will denote with SSE_t, and test that it stays below some threshold. We refer to this continuous testing with the term *monitoring.* Finally, such a threshold should take into account the dispersion obtained at the very moment the clusters were created, which we denote with SSE_0 (i.e. time counting starts from the moment the most recent clustering was performed), suggesting to adopt a relative threshold. That is summarized in the following problem definition:

Definition 1 (Cluster Monitoring Problem)
Given a clustering $C = \{C_1, \ldots, C_k\}$ having initial SSE equal to SSE_0, and given a tolerance $\alpha \in \mathcal{R}^+$, we require to ensure that at each time instant t the following holds for the SSE of the (dynamic) dataset D_t:

$$SSE_t \leq (1 + \alpha) SSE_0 \tag{2}$$

When that does not happen, a recomputation/update of cluster assignments should be performed.

We should note that SSE describes all the clusters together, aggregating the dispersions of the single clusters. That means that in principle having a good SSE does not guarantee that each single cluster is compact, since some slightly over-dispersed cluster might be balanced in the sum by some virtuous one that

adds very little to the SSE. From this perspective, it might happen that the end user of our application wants to ask for stronger requirements in the monitoring problem. Therefore, we will consider also the following variant of the problem, where the constraints are imposed over each single cluster:

Definition 2 (Strict Cluster Monitoring) *Given a clustering $C = \{C_1, \ldots, C_k\}$ having initial SSE equal to SSE_0, and given a tolerance $\alpha \in \mathcal{R}^+$, we require to ensure that at each time instant t the following holds:*

$$\forall_{i=1}^{k}.SSE_t^{(i)} \leq SSE_0^{(i)} + \theta^{(i)} \tag{3}$$

where $SSE_t^{(i)}$ is the contribution of cluster i to the SSE at time t, i.e. $SSE_t = \sum_{i=1}^{k} SSE_t^{(i)}$, and the $\theta^{(i)} \in \mathcal{R}^+$ are fixed thresholds such that $\sum_{i=1}^{k} SSE_0^{(i)} + \theta^{(i)} = (1+\alpha)SSE_0$. When condition (3) is violated, a recomputation/update of cluster assignments should be performed.

3 Distributed monitoring

Our application context implies that our data sources are both streaming and distributed, since each vehicle involved continuously generates updates for its indicators, and all these data need somehow to be collected by a single unit to elaborate them. This creates a simple network with several *nodes* that communicate exclusively with a single special node, that we call *controller*, where nodes just communicate everything and the controller performs all the computation. However, such a purely centralized solution is applicable only on the small scale, since the communication stream generated by millions of vehicles would be hard to sustain. In this section we propose a solution based on the distributed monitoring paradigm, that aims to save a significant amount of communications by deferring a small part of the computation to the nodes of the network. More specifically, our objective is to continuously verify that the clustering/segmentation satisfies the quality constraint (2) or (3), and recompute the clustering only when the constraint is violated. Therefore, a simple way to distribute the computation consists in providing to the nodes some *local* conditions to test such that, if each local test is successful, they guarantee also the satisfaction of the *global* quality constraint. The basic idea is that the nodes themselves can recognize the data changes that do not significantly impact on the quality measure to monitor, thus avoiding to update the controller with useless information.

In the following subsections we describe the setting of distributed monitoring of functions that we are going to use, showing how our problem can be translated is such terms. Then, we describe the full framework that implements the monitoring and integrates several forms of predictive models that improve its communication reduction capabilities.

3.1 Distributed monitoring of functions

The work in [19] addresses the general problem of monitoring the value of a function computed over data that are distributed in a network. More specifically, the framework considers a two-tiered setting, with n geographically dispersed sites (also called *nodes*) and a central coordinator (also called *controller*) that is capable of communicating with every site, while pairwise site communication is only allowed via the coordinating source. Each site receives a stream of data updates and maintains a d-dimensional local measurements vector $v_i(t)$. The task of the coordinator is to ensure that at each time instant t the following holds:

$$f(v(t)) \leq T \tag{4}$$

where f is a given function, $f : \mathcal{R}^d \to \mathcal{R}$, $T \in \mathcal{R}$ is a threshold and $v(t)$ is the weighted average of the $v_i(t)$ of all sites, i.e. $v(t) = (\sum_i w_i v_i(t))/\sum w_i$ for some weights $w_i \geq 0$. While the latter condition is apparently a strong limitation to the applicability of the framework, it has been shown that several interesting problems can be reformulated in this way. A basic means to do this, which will also be used later in this paper, is a *vector augmentation trick*, consisting in adding to vectors $v_i(t)$ (sent by the sites to the coordinator) one or more extra components. The basic example is the task of monitoring the variance of all $v_i(t)$, assuming $d = 1$, i.e. ensure that $var_i(v_i(t)) \leq T$. While not directly fitting the form in (4), we can exploit the well known property $var(X) = avg(X^2) - [avg(X)]^2$ to rewrite our problem as $f(v(t)) = v(t)_1 - [v(t)_2]^2$, assuming that each site now communicates a 2-dimensional vector $\big(v_i(t), v_i(t)^2\big)$.

The algorithmic solution proposed in [19] to perform the monitoring of (4) follows a so called *geometric approach*. All the points in $\mathcal{R}^d$ where (4) is satisfied form the *admissible region* G, and our objective is simply to ensure that $v(t) \in G$. The method stems from the following property: the convex hull of a set $\{x_i\}_i \subset \mathcal{R}^d$ of points is entirely contained in $\bigcup_i B(x_i, e)$, where e is any point in $\mathcal{R}^d$ and $B(x_i, e)$ is the ball having the segment $\overline{x_i e}$ as diameter. In turn, it is straightforward to see that our $v(t)$ is contained in the convex hull of the set $\{v_i(t)\}_i$. Therefore, if every ball $B(v_i(t), e)$ is contained in the admissible region (namely, it is *monochromatic*), then also $v(t)$ will be, and therefore (4) will be satisfied. Once the controller has communicated to all sites the point e, each site will be able to test whether its ball $B(v_i(t), e)$ is monochromatic. As long as no site detects a failure, we are guaranteed to satisfy (4), without any need of communicating information to the controller. When a site fails, it notifies the controller, who, in the basic approach, will ask to every site to send their new vector values, and test condition (4). Notice that the test performed on each site might cause false alarms (its ball exits the admissible region, yet the overall $v(t)$ is still inside) but not false negatives, i.e. when condition (4) is violated the system will always discover that. While in principle the point e can be chosen freely, it is convenient in practise to compute it as $e = v(t')$, where t' is the time of the most recent *synchronization*, i.e. the phase where every site communicates its new values to the controller. Beside at the start-up of the system, synchronizations usually occur when a site rises an alarm. With

this choice, it results useful to slightly change the method by asking each site to check the ball $B(u_i(t), e)$ instead of $B(v_i(t), e)$, where $u_i(t) = e + (v_i(t) - v_i(t'))$ is called the *drift vector*. It is trivial to prove that the average $avg_i(u_i(t))$ is equal to $v(t)$, thus not changing the monitoring task. Yet, now, immediately after any synchronization we have that $u_i(t') = e$, therefore the vectors of all sites start from the same point e, and the balls $B(u_i(t), e)$ have actually the size of a single point, clearly reducing the chance that a false alarm will rise in the near future.

3.2 Distributed monitoring of cluster quality

The cluster monitor problem (Definitions 1 and 2) can be fitted to the geometric approach described above, by properly rewriting it and exploiting the vector augmentation trick seen for the variance monitoring (Section 3.1). Indeed, the formulation of SSE is very similar to a variance, though on d dimensions. To find the precise relation between the two, we observe that each cluster C_i, having centroid c_i, contributes to the SSE by the following value:

$$\begin{aligned} SSE^{(i)} &= \textstyle\sum_{p \in C_i} ||p - c_i||_2^2 \\ &= \textstyle\sum_{j=1}^{d} \sum_{p \in C_i} (p^j - avg_{q \in C_i}(q^j)) \\ &= |C_i| \textstyle\sum_{j=1}^{d} var_{p \in C_i}(p^j) \\ &= |C_i| \textstyle\sum_{j=1}^{d} \left[avg_{p \in C_i}((p^j)^2) - \left(avg_{p \in C_i}(p^j)\right)^2 \right] \end{aligned} \tag{5}$$

where p^j represents the j-th component of the d-dimensional vector p. This means that by augmenting the vector $v_i(t)$ of each node with the additional d features $v_{i,1}(t)^2, \ldots, v_{i,d}(t)^2$, we can compute the variance for each component, as requested in (5). Actually, we can do slightly better, by further aggregating the terms in the last line:

$$SSE^{(i)} = |C_i| \cdot \left[avg_{p \in C_i} \left(||p||_2^2 \right) - ||avg_{p \in C_i}(p)||_2^2 \right] \tag{6}$$

which means that only one additional component is needed, corresponding to $||p||_2^2$ of the node (p represents our $v_i(t)$).

3.3 Monitoring framework

The relation (6) states that the geometric approach discussed in Section 3.1 can be applied to monitor a single cluster, provided that we have defined a threshold value for it. This provides a direct solution to the strict version of our monitoring problem (Definition 2), since it already introduces an individual threshold for each cluster, and therefore implicitly partitions the problem into K separate sub-problems. The basic problem (Definition 1), instead, does not impose such a partitioning, and allows some interplay between the different clustering, e.g., clusters with a high $SSE^{(i)}$ might be *balanced* by others having a low value.

Problem partitioning. In our solution we choose to divide the monitoring into K separate sub-problems in both the basic and the strict problem definitions. That

means, in practice, we will introduce thresholds for each single $SSE^{(i)}$ also in the basic case, thus actually making it more restrictive. Additional mechanisms to exploit the interplay between clusters allowed by the basic problem will be introduced later in this section. Then, after each synchronization, where all nodes send their vectors to the controller (including the initial start-up of the system), the controller will send to the node of each cluster both its centroid and the corresponding SSE threshold. With this information, each node will be able to locally test sufficient conditions that ensure the cluster not to exceed the SSE threshold assigned to it, based on the geometric solution described in Section 3.1.

In order to perform the above mentioned threshold partitioning, we essentially need to define the values (or rather, a method to compute them) of the constants $\theta^{(i)}$ mentioned in Definition 2, i.e. the SSE increments allowed to each cluster. Two natural choices emerge: using the same value for all clusters; or making $\theta^{(i)}$ proportional to $SSE^{(i)}$. The first solution will be called *uniform SSE distribution*, and implies that we set $\forall i.\theta^{(i)} = \alpha SSE_0/K$. The second one will be called *proportional SSE distribution*, and requires to set $\forall i.\theta^{(i)} = \alpha SSE_0^{(i)}$. In order to allow a flexible choice between the two alternatives, we will adopt the following schema, parametric in $\beta \in [0, 1]$:

$$\forall i.\theta^{(i)} = \beta \left(\alpha SSE_0^{(i)}\right) + (1 - \beta) \left(\alpha \frac{SSE_0}{K}\right) \tag{7}$$

Thus, $\beta = 0$ will correspond to the uniform distribution, $\beta = 1$ to the proportional one, and any value in the middle to some trade-off between them.

Multi-level monitoring. The previous discussions suggest that the cluster monitoring can be organized at three levels, as depicted in Fig.1. At the bottom layer, each node checks its local constraints, based on the information it received during the last synchronization. Whenever a local violation occurs at some node, the controller reacts by verifying whether the alarm was true (the $SSE^{(i)}$ of the corresponding cluster is too large) or not, which constitutes the second layer. The basic action taken by the controller is to simply query all nodes of the cluster to compute the real $SSE^{(i)}$, yet smarter solutions are possible, including the node balancing strategy that will be discussed later in this section. When the alarm is true, the strict problem definition requires a recomputation of the clustering, and thus the controller also needs to start a full synchronization. In the basic problem definition a smarter approach is possible (also discussed later in this section) which checks whether other, more *virtuous* clusters have a $SSE^{(i)}$ that can balance our large one.

Algorithm 1 summarizes the whole monitoring process for a more complex case, i.e. the basic problem, where some balancing is possible between single nodes or between clusters.

First, in an initialization phase (steps 1–6) the controller receives all local vectors $v_i(0)$ from the nodes and computes a first clustering, sending back to the nodes all the information they need to start the distributed monitoring. The monitoring phase (steps 7–25) contains three main actions: first, each node tests

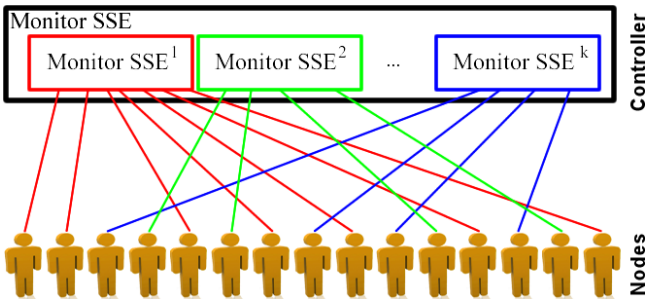

Figure 1: Logic organization of the framework.

its local constraints (steps 9–12) using the methods described in Sections 3.1 and 3.2. If no violation occurs, the monitoring cycles simply goes on without any communication and any action from the controller's side. If, instead, there is some local violation, the controller communicates with the nodes of the corresponding cluster to check is the violation can be balanced (steps 13–15). If that is not possible, it means the cluster SSE is too large, and the controller tries to see if other clusters can help to balance it (steps 16–17). If everything fails, it means the whole clustering is actually violating the monitoring condition (2), and therefore an initialization of the system is required (steps 18–19). The latter event triggers the execution of the last phase (steps 20–24), essentially requiring the same operations of the initialization phase. We remark that the strict version of the monitoring problem does not allow any interplay between clusters, therefore, in that case the cluster balancing (step 17) should simply be omitted. Also, as we will see later, the cluster balancing is executed in such a way that a when it fails all nodes will already have sent their up-to-date vectors to the controller, and therefore it is not required any further communication from the nodes side. When the cluster balancing is omitted, as for the basic monitoring problem, such communications are needed, and step 2 should be replicated before the clustering takes place (i.e., between steps 20 and 21).

3.4 Node-level improvements

The proposed solution can be improved in several ways, by exploiting recent developments of the general theory summarized in Section 3.1 (which will be discussed in the next paragraphs) or by extending it with application-specific improvements (topic mainly covered in the next section).

Safe Zones for convex inadmissible regions. The geometric method discussed in Section 3.1 provides a means to decide locally to the node which vector values guarantee that the overall function satisfies the global constraint to monitor. The set of such values is also called *safe zone* and, since it is only based on the global function and on the reference point e, all nodes have the same safe zone.

In [14] it is shown that the safe zones built by the geometric method can also be computed as the intersection of an infinite set of hyperplanes. Yet, it

Algorithm 1: Overall monitoring workflow

```
Input: number of clusters K; tolerance α; dynamic customers data set D
// Synchronization and setup
Initialize t := 0;
All nodes in D send vectors v_i(0);
Compute clustering;
foreach cluster C_i do
    Compute threshold SSE^(i)_thr;
    Send c_i and SSE^(i)_thr to all nodes in C_i;
// Monitoring
repeat
    t := t + 1;
    foreach node S_i ∈ D do
        S_i tests its local constraints;
        if Violation then
            Send alarm to Controller;
    foreach cluster C_i do
        if Controller receives at least one alarm from C_i then
            Try node balancing on C_i;
            if balancing failed then
                Try cluster balancing;
                if balancing failed then
                    Exit loop and request synchronization;
    if Requested synchronization then
        // Recompute clusters
        Compute clustering ;
        foreach cluster C_i do
            Compute threshold SSE^(i)_thr;
            Send c_i and SSE^(i)_thr to all nodes in C_i;
until Forever ;
```

is also shown that part of them are unnecessary, which makes the safe zones smaller (and thus less effective) than what strictly needed. A particular case is that where the inadmissible region (the set of values that violate the global constraint) is convex. In this situation we can easily find an optimal safe zone in two steps: first, find the point p of the inadmissible region which is closest to the reference point e; second, draw the hyperplane that passes through p and is orthogonal to the segment $\overline{ep}$, and then, of the two half-spaces determined by the hyperplane take as safe zone the one that contains e.

This is very relevant for our problem, since the $SSE^{(i)}$ of a cluster has a quadratic form with positive coefficients, and therefore our inadmissible region is the part of space that stands above an "upward" parabola, which is a convex set. For this reason, in our application we will adopt the smarted safe zones described above.

Balancing. As already mentioned earlier, when a node violates its local constraints (i.e., it exits its safe zone) there might be other nodes that can compensate it. A simple method to do that consists in applying a straightforward property: if we move the local vectors $v_i(t)$ and $v_j(t)$ of two nodes, respectively by δ and $-\delta$, the overall average $v(t)$ will not be affected. That means that if

$v_i(t)$ trespassed the border (an hyperplane, in our case) of the safe zone by an amount δ, generating an alarm, and we realize that $v_j(t) - \delta$ still remains in the safe zone, we can remove the violation by translating both vectors as described above, without any further effect on the monitoring process. Actually, since the border of the safe zone is a hyperplane, we only need to consider δ that are orthogonal to it, therefore we can code the translation by means of a single scalar, representing the distance between $v_i(t)$ and the hyperplane.

The balancing of nodes, then, is realized along the following strategy: when a node of cluster C_i raises a violation, it also communicates its amount δ (which will be negative); then, the controller asks to 2 other nodes of the cluster, chosen randomly, their distance δ_1 and δ_2 from the hyperplane. If $\delta + \delta_1 + \delta_2 \geq 0$, the balancing is successful, otherwise the controller tries with 4 more nodes, etc., at each iteration doubling the number of nodes involved, till either $\delta + \sum_i \delta_i \geq 0$, or all nodes have been contacted without success. In the latter case, the balancing fails.

When the balancing is successful, we have two alternatives, that we call respectively *balancing with memory* and *memoryless balancing*. In the first one, the node that rose the violation and the other ones needed to balance it, permanently translate their vectors; in the second one, the translation is only virtual, and all nodes keep their vector. The latter is expected to work better in situations where the violations are due to temporary noisy updates of the vectors, which should disappear at the next timestamp.

The balancing process described above, when successful, assigns a final distance from the hyperplane that is zero for the node who rose the violation and all those used to balance it, while all other nodes (those queried by the controller but not needed to reach the amount $|\delta|$) are left unchanged. A possible alternative consists in building a *buffer* between all the n nodes and the hyperplane, by redistributing the total amount of δ_i of the nodes not yet involved. That can be realized by setting the distance from the hyperplane of each node involved to $(\delta + \sum_i^n \delta_1) / (n+1)$.

Finally, the same kind of reasoning followed so far can be applied at the level of clusters, in the case of the basic problem definition: if a cluster exceeds its assigned threshold $\theta^{(i)}$, the controller can try to balance it with other clusters. In order to do that, all the nodes of each cluster C_j involved will send their up-to-date vectors, in order to compute the real $SSE^{(j)}$, and see if it can be used to balance $SSE^{(i)}$. Similarly as above, we can also introduce a buffer, in order to provide some extra-tolerance to the clusters for the next iterations of the monitoring. Since the number of cluster is usually relatively low, the exponential schema applied at the level of nodes can be replace by a linear one, i.e. the controller involves new clusters in the balancing one-by-one, instead of doubling their number at each step.

In Section 4 we will experiment all these variants, in order to verify which ones seem to fit better our application.

3.5 Introducing predictive models

The recent work in [8] extends the geometric approach by introducing predictive models. The basic idea is that the reference point e used to define the safe zones needs not to be static. As far as all nodes always share the same value, it can change in time. The basic idea consists in defining models that provide some predictions for each $v_i(t)$, which in turn are combined together by the controller to provide a model that predicts $v(t)$. Such model is distributed to all nodes, which will then use its predicted values as (time-varying) reference point. The final effect is that, if the predictive models work well, the drift vectors of the nodes (see Section 3.1) will all be very close to the real value $v(t)$, thus minimizing the chance of false alarms.

We distinguish two types of predictive models to use in our framework: one based on the detection of trends in the recent data; and one based on the detection periodicities of the data, obtained by analyzing historical information.

Trend-based models. From this family of models, introduced in [8], we will consider two alternatives: the *linear growth* model, which assumes that the vector of a node will evolve in time following the law $v_i(t) = \frac{t}{t_s} v_i(t_s)$, where t_s is the time of the last synchronization; and the *velocity/acceleration* model, which assumes the law followed has the form $v_i(t) = v_i(t_s) + (t - t_s)vel_i + (t - t_s)^2 acc_i$, where vel_i and acc_i are estimates of the velocity and acceleration (as vectors) of $v_i(t)$ estimated at time t_s by the node itself, by looking to a piece of its recent history. Both models favour simplicity and ease of computation, only requiring to send very little extra information on the network.

History-based models. This models extend the approach of [8] by learning regularities in the behaviour of a vector $v_i(t)$ from a segment of history of its node, in way similar to [16]. We consider two variants. The first one is called *history-based constant* model, very simply computes the average value of $v_i(t)$ on the history segment, which then represents the "default" vector for the node. The second one is called *history-based variable*, and performs a similar computation, yet providing different values for different time slots within the 24h of the day. This way, we will obtain a "default" vector which is dependent on the hour of the day (or other time unit adopted). Since they are based on longer-term history, as compared to the trend-based methods, these solutions require to send the local models of the nodes to the controller (and the aggregate global model backward) only at the beginning of the monitoring, thus not affecting communications in any significant way.

Adapting the *Choosing Among Alternative* approach shown in [8], we combine together all models listed above in the following way: at each synchronization only one model type is chosen, based on how well each model performed since the previous synchronization. Therefore, each node always applies all models, in order to test whether it would cause an alarm or not, even if only one of them is actually used to decide whether a violation is occurring. During the synchronization, for each model the coordinator counts the number of nodes where the model was successful (i.e., did not signal a false alarm), and selects

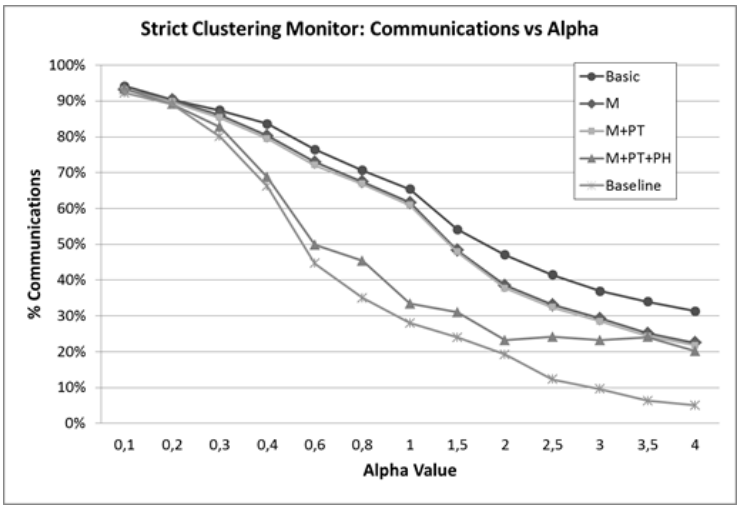

Figure 2: Communications during the *strict* clustering monitor by varying the α parameter.

the best one.

3.6 Dataset and data preprocessing

The dataset is provided by an Italian company called *OctoTelematics* collecting data for insurance purposes. This dataset is composed by GPS observations of 11,470 [1] private cars active in Tuscany in a period of 35 days between June and July 2011. Due some pre-processing (i.e. aggregation and filtering) performed by the device on board the sampling rate is reduced to a observation every 3 minutes and it is not regulated by any policy of synchronization. Moreover, we divided the dataset temporally in order to create a training and a test set respectively using the first week and the remaining 4 weeks. The two dataset are processed to extract the measures presented in Sec.2.2 using a time window of 3 days with a time granularity of 15 minutes. Unfortunately, due the low sampling rate, some of the measures can't be extracted, i.e. all the acceleration based measures, therefore we have excluded them from our experiments. Once all the measures are collected we studied their values and since some variables follow skewed distribution, they were transformed to a log scale. Finally, all variables have been normalized through z-score, thus making all variables have zero average and variance equal to 1. Since the idea is to build the *customer profiles* using a clustering method, we must consider also the correlation between the measures, hence we used the training set to produce a schema of correlations revealing that several strong correlations held. Therefore we selected a subset of measures to avoid strong biases in successive analyses and, as side

[1] The dataset is available at *kdd.isti.cnr.it/node/493*

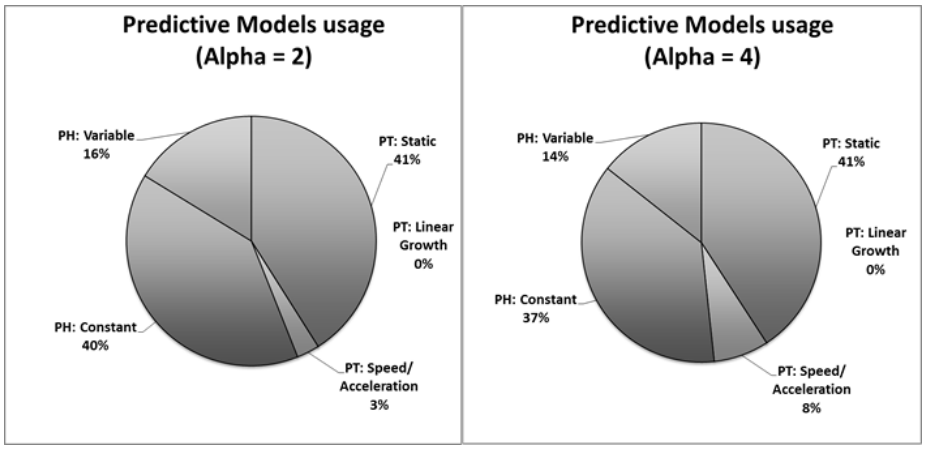

Figure 3: The percentage predictive model usage.

effect, reduce the dimensionality of the dataset. The attributes remained after this selection are the following: *Duration*, *Radius_g_L1*, *TimeL1L2*, *EntropyArc*, *Pcity*, *Phighway*, *Pnight*, *Pover* and *Profile*.

4 Experiments

4.1 Performance evaluation

To evaluate the performances of the proposed method we consider the amount of communications exchanged between the nodes and the controller. To better study the performances we separated two different kinds of messages: the ones coming from the nodes toward the controller and vice versa. The communications of the first type are always of the same size (a vector with d dimensions plus other parameters for predictive models) and the channel is a *point-to-point* link between the node and the controller; meanwhile the second group is composed by communications with different sizes that can use *broadcasting* capabilities of the networks to reach all the nodes at once. A worst case analysis of the communications from the controller revealed that they are dominated by the requests for the balancing and they have an upper bound of 4%, which in an empirically evaluation drops to values between 1.23% and 2.34%. For sake of readability we first present the results of the strict clustering monitor. In Fig.2 the method is evaluated varying the α parameter. The algorithm is applied with different variants presented in previous sections: the method with only node balancing without memory (**Basic**); with memory during the balancing (**M**); using the trend predictive models (**PT**); using the history predictive models (**PH**); and considering a buffer in the balancing (**B**). Moreover in the figure we report also the baseline representing a lower bound of the communications needed to monitor, computed as the communications generated by synchronizations. It is

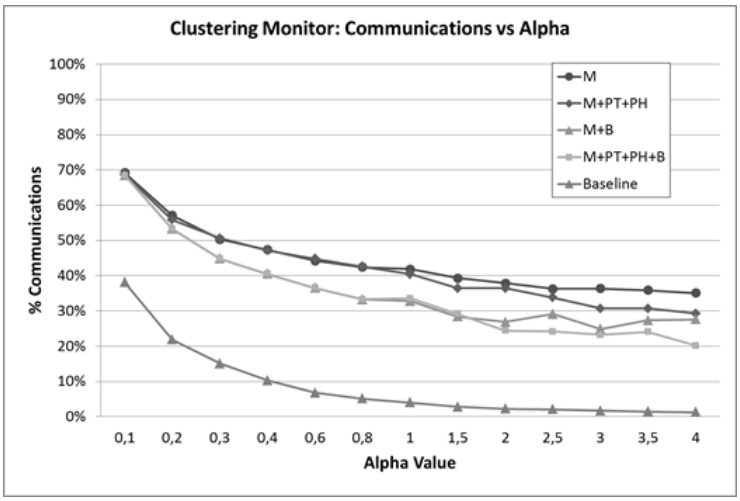

Figure 4: Communications during the clustering monitor by varying the α parameter.

interesting to see the effect of the introduction of the memory during the node balancing (in the strict problem the clusters are not balanced) which gives a first boost to the performances reducing the communications. The usage of the trend based predictive models contributes only marginally to reduce communications, while the history based ones have a rather large impact, reducing the communications close to the baseline. The impact of the history-based predictive models it is also confirmed by the Fig.3 where the usage of each predictive model is analyzed in terms of the amount of time in which it is used during the monitoring. It is important to notice how, excluding the static one, the constant history based model is the most used outperforming the others. The second most used is the variable version, which is probably more affected by noisy values than the constant version. The least used is the linear growth which is never used and clearly not suitable to describe the trend of the data. Considering the basic version of the clustering monitor problem, it is clear from the baseline in Fig.4 that the communications needed are less. Our method obtains good performances and, as in the strict version, the predictive models improve the performances. Experiments showed that the buffer feature presented in Sec.3.4, applied at the node level, does not improve the performances significantly, hence in the experiments we present it only in the case of the balancing applied at the cluster level, where we can see that it reduces significantly the communications, both with and without the predictive model.

4.2 Cluster evaluation

Given the result of a clustering operation, our end user needs to interpret the solution by defining and labeling the obtained clusters. Interpreting clusters

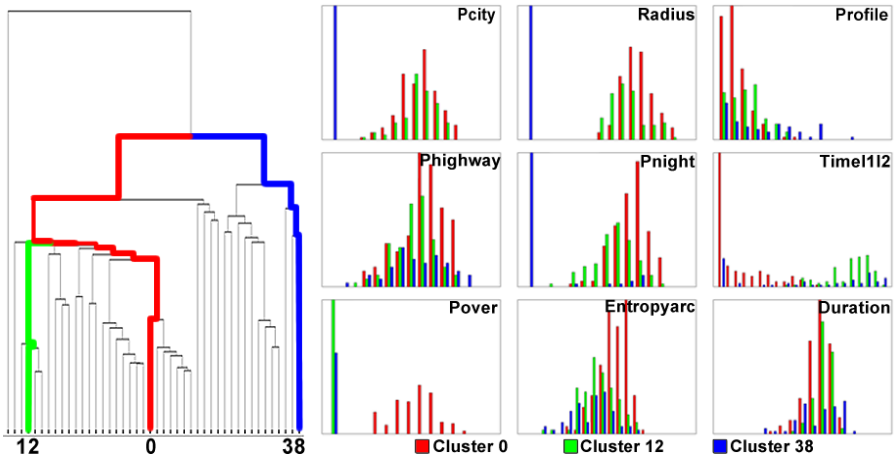

Figure 5: Hierarchical clustering on a set of centroids (left) and feature distributions (right).

usually involves examining their centroids, which represent the average behavior of each group of objects. This analysis is important because sheds light on whether the segments identified by the clusters are conceptually distinguishable. As a consequence we have decided to examine the values of centroids and tried to identify explanatory features to profile the clusters. The first step of the interpretation process was to understand the relation among the cluster centroids in order to understand their similarity (or dissimilarity). Therefore, we have applied on the set of centroids a hierarchical clustering algorithm with "single-linkage" method for the distance computation. The result simply highlights groups of centroids (clusters), which are similar and clusters that are particularly different. In this way, it is easy to understand which clusters represent similar customer profiles. In Fig.5 (left) we depict the result of the hierarchical clustering on a set of centroids. We observe that by analyzing the dendogram an analyst could easily understand that the customer profile represented by Cluster 12 (green) and that of Cluster 0 (red) should be more similar to each other than the customer profile of Cluster 38 (blue). This is confirmed by the behaviors of some of the feature distributions in each one of these three clusters. Indeed, in Fig.5(right), we observe that the feature distributions are more similar in Clusters 0 and 12; especially, for almost all the *context-aware* features.

Analyzing the feature distributions in Fig.5(right) we can characterize each one of the three clusters by labeling them with a particular customer profile. We classified Cluster 0 as the group of drivers that we call *explorers*. The people in this cluster travel a lot without following any particular systematic behavior and the entropy of their movements is high. Moreover, this category of drivers tends not to respect the speed limits. Cluster 12, as explained above, is very similar to Cluster 0 but the people here have a more systematic behavior (Profile), spend more time in their frequent locations – typically representing home and working

locations –, which probably determine most of their mobility. We identify these people as *long-range commuters*. Finally, Cluster 38 represents drivers called *Sunday drivers*; indeed, they travel rarely, typically use the highways and do not travel within the city. Also, they tend to respect the speed limits.

Another important point in the monitoring of clustering is how much time passes between consecutive reclusterings. As expected, we verified that it strongly depends on the α parameter. As an example, for $\alpha = 4$ we perform the re-clustering on average once every 20 hours, and an average of 11 hours for $\alpha = 0.8$. A deeper analysis reveals that this average is strongly affected particularly unstable periods of time where many clusterings are executed consecutively. However, once good profiles are detected the relative clustering persists longer.

5 Related Work

We briefly summarize some works in the literature on fields that are tightly connected to our proposal.

Distributed Clustering can be classified into two main groups. The first one includes completely decentralized methods requiring a significant amount of communications among nodes [5, 7, 3]. Most of these works propose algorithms for k-means clustering over a P2P network. The second group requires that the nodes build local clustering models and send them asynchronously to a central station, that forms a combined global model [18, 17, 13]. The basic assumption of these algorithms is that each node contains more than one point. Some of these works propose algorithms for hierarchical clustering [17], others extend the density-based clustering to this setting [13] and others present a k-means version suitable for wireless sensor networks with a hierarchical structure [18]. As opposed to the methods mentioned above, in our context, in a given timestamp each node has a single point that is transmitted to the central station for the clustering. Moreover, our setting does not allow communications among nodes.

Clustering on Data Streams is addressed in [10]: given a sequence of points, the approach maintains a good clustering of the sequences observed, using a small amount of memory and time. Some other works in the data stream scenario may be found in [1, 2, 6]. [9] proposes an algorithm for clustering distributed data streams. Given a network of nodes, where each of them receives its share of a distributed data stream, the goal is to obtain a common clustering. Here,the nodes cannot share single points of their datasets, but only aggregate information. This setting differs from ours because they image to have more than one point in each node.

Monitoring. Besides the methods addressing the monitoring of arbitrary threshold functions over distributed data streams described in [19, 8] and already discussed in previous sections, in the literature some works treat the problem of clustering monitoring by considering different aspects and settings. For instance, [20] studies the monitoring of the L2 norm, useful for monitoring the accuracy of a k-means clustering. Others consider the clustering monitoring in a P2P

network by proposing a hierarchical structure of nodes [12].

Customer Segmentation. Market segmentation is one of the most fundamental strategic marketing concepts: grouping people according to their similarity in several dimensions related to a product under consideration [4]. Typically, for this task predefined customer classification rules are applied. However, recently, some methods propose the use of clustering algorithms. As an example, in [11] authors propose the use of k-means clustering analysis to group consumers into segments by collecting historical data accumulated in business support systems of a telecommunication company. They do not consider any distribution of the data and the analysis is performed off-line.

6 Conclusions

In this paper we explored an innovative instance of the customer segmentation problem in the context of car insurances, characterized by a strong human mobility component and the large scale, distributed and streaming nature of its data sources. We developed and tested all the components of the application, exploiting a large dataset of car mobility data, showing the feasibility of the approach. We believe that the distributed monitoring of cluster quality proposed here has a validity beyond the application domain, and we plan to validate it into an extensive array of clustering problems so that to achieve a general methodology.

References

[1] C. Aggarwal, J. Han, J. Wang, P. Yu. A framework for clustering evolving data streams. VLDB 2003, pp. 81-92.

[2] B. Babcock, M. Datar, R. Motwani, L. O'Callaghan. Maintaining variance and k-medians over data stream windows. ACM PODS 2003, pp. 234-243.

[3] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, S. Datta. Clustering distributed data streams in peer-to-peer environments. Inf.Sci.176(14):1952-1985,2006.

[4] M. J. Berry, and G. S. Linoff. Data mining techniques: for marketing, sales, and customer relationship management. Wiley Computer Publishing, 2004.

[5] S. Datta, C. Giannella, H. Kargupta. Approximate Distributed K-Means Clustering over a Peer-to-Peer Network. IEEE TKDE 21(10): 1372-1388, 2009.

[6] P. Domingos, L. Spencer, G. Hulten. Mining time-changing data streams. ACM KDD 2001, pp. 97-106.

[7] P. A. Forero, A. Cano, G. B. Giannakis. Distributed Clustering Using Wireless Sensor Networks. J. Sel. Topics Signal Processing 5(4): 707-724, 2011.

[8] N. Giatrakos, A. Deligiannakis, M.N. Garofalakis, I. Sharfman, A. Schuster. Prediction-based geometric monitoring over distributed data streams. SIGMOD 2012, pp. 265-276.

[9] A. Guerrieri, A. Montresor: DS-Means: Distributed Data Stream Clustering. Euro-Par 2012, pp. 260-271.

[10] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan. Clustering data streams. FOCS 2000, pp. 359-366.

[11] X.Hong; Q. Gangyi. Data Mining in Market Segmentation and Tariff Policy Design: A Telecommunication Case. Information Processing (APCIP) 2009, pp. 328-331.

[12] M. Hua, M. Ki Lau, J. Pei, K. Wu. Continuous K-Means Monitoring with Low Reporting Cost in Sensor Networks. IEEE TKDE 21(12): 1679-1691, 2009.

[13] E. Januzaj, H.-P. Kriegel, M. Pfeifle. DBDC: density based distributed clustering. EDBT 2004, pp. 88-105.

[14] D. Keren, I. Sharfman, A. Schuster, A. Livne. Shape Sensitive Geometric Monitoring. In IEEE TKDE 24(8): 1520-1535, 2012.

[15] P.-N. Tan, M. Steinbach, V. Kumar. Introduction to Data Mining. Addison-Wesley 2005.

[16] M.Nanni, R. Trasarti, G. Rossetti and D. Pedreschi. Efficient distributed computation of human mobility aggregates through User Mobility Profiles. In UrbComp'12: ACM SIGKDD International Workshop.

[17] N. Samatova, G. Ostrouchov, A. Geist, A. Melechko. RACHET: An efficient cover-based merging of clustering hierarchies from distributed datasets. Distrib. Parallel Databases 11 (2): 157-180, 2002.

[18] P. Sasikumar, S. Khara. K-Means Clustering in Wireless Sensor Networks. Conference on Computational Intelligence and Communication Networks, 2012.

[19] I. Sharfman, A. Schuster, and D. Keren. A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams. ACM Trans. Database Systems, 32(4), 2007.

[20] R. Wolff, K. Bhaduri, and H. Kargupta. Local L2 Thresholding Based Data Mining in Peer-to-Peer Systems. SDM 2006.