A new *cluster-histo-regression* analysis for incremental learning from temporal data chunks

Nagabhushan P.[1], Syed Zakir Ali[1*] and Pradeep Kumar R.[2]
[*1]Department of Studies in Computer Science, University of Mysore, Mysore, India
[2]Amphisoft Technologies Private Limited, Coimbatore, India

Abstract - In scenarios where data chunks arrive temporally, a good algorithm for exploratory analysis should be able to generate the knowledge and with the next chunk of data arriving, the process should be the one of just updating online by accumulating the knowledge derived from the recent chunk. Such an incremental learning process in most of the cases indent a lot of memory requiring to carry all earlier data in the process of updating the knowledge successively. In this research work we propose to employ a novel *Cluster-Histo-Regression* analysis of the chunk to extract the knowledge for the temporal instant and fuse this knowledge through *Histo-Regression-Distance* analysis with the already accumulated knowledge. We have designed a methodology which (i) discards all those data samples from the chunk which have participated in the knowledge generation process (ii) indents minimum amount of memory to carry the accumulated knowledge and (iii) proposes to carry forward only those limited data samples (referred to as hard samples) which could not contribute to knowledge generated at that moment. Knowledge of each cluster is represented in the form of a histogram for each dimension of the clustered data and is transformed to regression line for the compact representation of the knowledge. The regression line parameters of the clusters obtained by incremental augmentation have shown an accuracy of up to 100% for some of the data sets that are considered for experimentation.
Key words: Zero instance memory learning, Partial instance memory learning, Knowledge generation, Cluster analysis, Regression analysis, Incremental learning, Incremental augmentation of knowledge

Introduction

Needless to mention that knowledge is generated through a process of learning from the available data. Histogram is a better representation of the knowledge. Clustering is the most important form of learning in pattern recognition applications. In traditional learning systems the entire data is processed at one stretch and hence it is referred as *one-shot* learning [1]. If the data is of a huge size and that it cannot be processed at one stretch with the available resources, then the traditional one-stretch learning has to be replaced by segmenting the data into smaller chunks or batches and assimilating the acquired knowledge in a piecemeal way leading to *piecemeal* learning. On the other hand, when the availability of data is intrinsically temporal and when sufficient quantum of data is already available to initiate the learning process, waiting for further data is undesirable and/or impractical because (i) the time gap required for initiating the processing of the data increases both in terms of wait time and one-shot processing time at the end (ii) the volume of data becomes prohibitively high and (iii) there should be a need for extracting the accumulated knowledge as and when the data chunk is received temporally (such as in trend analysis); hence one has to move from traditional *one-shot* learning to *piecemeal* learning temporally. In machine learning parlance, this type of learning from temporal sequence of chunks in a piecemeal way is often referred to as *on-line* learning [1] [2].

On-line learning systems [2] require two types of memory – (i) *concept memory* - memory for storing and carrying forward the *batch knowledge,* which is the description of the knowledge extracted from each chunk of data and (ii) *instance memory* - memory for storing the processed data and memory meant to accommodate the incoming chunk.

Clustering is the most established method to learn form chunks of data. In this research, statistical summary of each of the clusters (such as number of elements, mean, standard deviation, histogram etc) obtained from the chunk of data is provided as a *knowledge packet*. In the process of learning on-line using clustering, if the system retains a *knowledge packet* for each of the cluster of every chunk, the *concept memory* grows continuously with the addition of new clusters from the subsequent chunks of data. In this case, merging of *knowledge packets* towards the end will be a cumbersome process. Similarly, if the online systems retain the data of each chunk in the *instance memory*, it also leads to a continuous growth of *instance memory*; which increases the computational overheads and will act as a bottleneck for the continuous flow of chunks. With the increase in volume of data, there is a strong need for some learning system which (i) always retains only one knowledge structure in the memory for assimilating all the fragmented facts (*batch knowledge*) to get an *overall knowledge* and (ii) does not re-indent the past data, but which expects that the new process of learning should become more intelligent.

To incorporate the concepts of incremental learning in data mining, Michalski [3] has observed that provision should be made for inserting background knowledge and knowledge about the goal into the knowledge generation process and has called the process as Incremental Knowledge Mining. In [4], the following model for the knowledge generation process has been proposed:

DATA + PRIOR_KNOWLEDGE + GOAL → NEW_KNOWLEDGE

Specifying the *GOAL* makes the learning process *supervised* or more precisely *directed* learning. Since in many applications the goal may not be known apriori and since learning in an *unsupervised* scenario is more demanding [5-7], we propose to adopt clustering mechanisms for learning.

In machine learning process, knowledge updation via incremental learning is achieved by three forms [2] depending on the volume of data used or reprocessed or available for reprocessing from the knowledge generated in earlier stages. They are: (i) Zero instance memory learning, where none of the previously processed data are re-called or re-indented (ii) Full instance memory learning where all the past data are retained and re-processed and (iii) Partial instance memory learning where some of the past data which are likely to play a vital role in the learning process are carried forward for further processing. Zero instance memory learning is the most optimal in terms of space requirements and in our recent work [8], an attempt to achieve reasonably good learning through Zero instance memory learning has been made. Full instance memory learning is not desirable because of the computational overheads [2] [3]. Next is the Partial instance memory learning (PIML), which we would like to explore to greater depths in this research work since there has been relatively not much work done on this aspect by the machine learning community [2] as also in Data Mining community.

The important issue that is being addressed in PIML is *devising strategies to select samples from the input stream which can be of use in subsequent stages of learning* [2]. Another important issue which we feel important is *devising strategies to store the concept descriptions, which minimize the concept memory* and thereby reduce the computational overheads.

Various schemes that have been explored for the selection of samples to be carried forward are: (i) representative samples [9,10] (ii) consecutive sequence of samples over a fixed or changing window of time [11,12]; (iii) extreme samples that lie on the boundaries of concept descriptions [13,14]. In all these schemes, the samples, which describe the knowledge, are also being carried forward in partial instance memory. These samples increase continuously till all the samples are processed which directly increase the requirement of partial instance memory. Our contention is that *once the concepts/knowledge of the given data is extracted, there will not be any need for such samples.* Our next question is that *can we not keep the outliers separate from the concept descriptions so that the density of the concepts becomes stronger and the outliers can be forwarded to subsequent stages?* In fact, we require only those data samples which could not participate in the knowledge generation process, specifically with regard to temporal data chunks. Such samples may be referred as *hard samples*. Moreover, the *hard samples* can get merged with the succeeding chunks of data and the quantum of *hard samples* will remain minimal at each stage of learning, thus minimizing the continuously growing requirement of partial instance memory.

Various attempts to extract knowledge incrementally through classification and clustering have been made in [15-23]. Knowledge has also been extracted in the form of association rules [24–27]. Another way of knowledge extraction is through dimensionality reduction [28]. However, none of the above works have concentrated on incremental agglomeration of knowledge. For example, to get the overall knowledge about an area during surveillance by aircraft, image mosaic is being done at physical level and the information is collected from the overall mosaic image [29]. Our idea is that *if the time gap between the arrivals of two image frames can be utilized to extract the knowledge contained in image frame and if this knowledge could be updated with the knowledge of the next image frame, the burden of mosaicing the image frames physically and processing the bigger image frame can be eliminated*. At each stage, only the accumulated knowledge up to the end of the previous image frame should be utilized and all the previous image frame(s) may be dropped.

For incremental learning using clustering, an algorithm called *Incremental DBSCAN* has been proposed [16]. In this method, updates to the database are processed separately. It is observed that whenever an object is being inserted into an existing cluster, the objects already contained in that cluster can change their property (for example, core objects may become non core objects and *vice versa*; border objects may become noise objects and *vice versa*); in this connection the objects present in the cluster are assumed to be intact and are accessible, which increases the partial memory requirements. Further, the incremental DBSCAN cannot withstand the loss or unavailability of the objects of a cluster.

Some inspiring works have been reported from data stream point of view [30, 31]. The data stream is assumed to be divided into chunks of manageable sizes. The first chunk of data is processed offline by applying k-means clustering algorithm. The number of clusters obtained initially from the first chunk is made proportional to the available memory that means that the initial number of clusters will be at maximum level and are significantly larger than the natural number of clusters in the data. For each of the data sample that arrives from the subsequent chunk, a check is made for absorption into any of the existing clusters. If merging is accepted, then the data sample is merged otherwise the oldest cluster (which has not got updated from a long time) is deleted to make way for the new cluster to be created with this sample. This process is nothing but merging of data with the existing knowledge. However, this mechanism of updating knowledge suffers from the following drawbacks. (i) Clusters generated at the initial stages could be sparse (ii) Subsequent learning process becomes biased by the clusters generated from the initial chunk of data (iii) Deleting the oldest cluster is nothing but loosing some important knowledge forever which otherwise could be of importance for the future data (iv) Clear view of stagnancy cannot be detected to enforce stopping criteria for the continuous learning process that is when there is no significant change in the generated knowledge, there is no necessity of continuing the learning process.

A new clustering algorithm called LSEARCH has been proposed to learn from the stream of data [31]. The method of handling the data stream is similar to the one proposed earlier [30]. But, instead of specifying the number of clusters in advance, the authors propose a mechanism called facility location to find out the optimal number of clusters. In this method, k-medians are assigned initially and the quality of clustering is measured by the sum of squared distances of data points from their assigned medians.

The data stream methods proposed in [30] and [31] assume the steady arrival of data and they also assume that the time gap between the arrivals of two data samples is sufficient for getting the knowledge updated. If the same concept can be extended to the chunks of data, the time gap between the arrivals of two consecutive chunks should be enough to perform mining and knowledge integration. In [30], when it is assumed that the data stream can wait till the first chunk is processed, why the stream cannot wait for other chunks? If it can wait, we can overcome the necessity of using online clustering algorithms for learning as every chunk could be processed offline and only agglomeration of knowledge need be done online. Since in many practical scenarios, the number of clusters is not known in advance, a clustering algorithm which avoids the specification of number of clusters in advance could provide dense clusters. Further, there should be some mechanism in the clustering algorithm to separate outliers as they greatly affect the quality of clusters.

The density based spatial clustering of applications of noise (DBSCAN) provides a mechanism to separate outliers from the data [15, 32-34]. It is argued that DBSCAN is one of the most efficient algorithms on large databases and is applicable to any database containing data from metric space to a spatial database or to a WWW-log database. Any insertion(s)/deletion(s) of object(s) affect the current clustering only in the neighborhood of the object because of the density based nature of DBSCAN. It is also clear that DBSCAN requires only one scan to cluster the objects. Because of these advantages, we are motivated to use DBSCAN algorithm for local cluster analysis in the present research. However, DBSCAN is sensitive to its parameters *MinPts* and *Eps*. These parameters must have to be managed properly with the *background knowledge* [2] during the knowledge generation process.

For the other issue of storing the *concept descriptions (knowledge parameters),* which minimizes the concept memory and computational overheads, [15, 30] have suggested Linear sum of element*s* (LS), Sum of squares of all elements (SSQ), Number of elements in a cluster as the knowledge parameters. It is observed that these parameters are the compact representation of the data and are sufficient for calculating the measurements required for clustering decisions. Undoubtedly, these parameters are good for spherical shaped clusters. However, for clusters of elongated and irregular shapes, we have to identify proper knowledge parameters. It is noted that only histogram has the generic ability to effectively characterize most of the data types and the internal pattern of distribution of elements within the clusters can be easily depicted by histograms [35]. Hence we used histograms to represent the concept descriptions. It is observed that algorithms for producing histograms have parameters such as number of bins and bin width [36, 37]. It is required to have same number of bins and same bin width for all datasets or all clusters within a dataset for the effective characterization of data into histograms. Hence the data has to be normalized to maintain the histogram spread between 0 and 1 and the number of bins could be fixed as 10 or 20 depending upon the required precision. Spread of the histogram is divided by the number of bins to get the bin width. It is noted that storage of histograms requires memory for number of bins, bin width and memory for the details of each bin. It is further shown that histograms can be easily converted to regression lines. Regression line is also a powerful knowledge representative which holds more details about the data in a compressed form which requires only two parameters - slope and intercept. Since we propose to have a histogram for each feature or each dimension of the clustered data, we opted to have a regression line instead of a histogram which reduces the memory requirement. Ultimately, because of linear regression, the parameters for histograms such as bin width, number of bins etc will not be of concern. Proofs have also been provided to show that regression line is a condensed form of knowledge representation [38]. Moreover, suitability of regression analysis by dividing the data into smaller blocks and finally combining the regression results has been established by Fan et al [38], though it is not for temporal data. Hence, we have employed *Cluster-Histo-Regression* analysis in this research work. For comparison between the *knowledge packets*, distance between the corresponding *histo-regression* lines has to be measured and for this we employed *Histo-Regression-Distance* [36, 37]. To make it complete, a glimpse at the conversion of histograms to regression lines through an intermediate stage of constructing its cumulative histogram and finding distance between regression lines as proposed by [36, 37] is given in Appendix.

As discussed earlier, we concentrate here on achieving incremental learning in the context of data arriving temporally where, incremental learning is the only possibility or a better alternate. Further we presume that the data which is arriving temporally is computationally manageable and can be easily processed at one step without the requirement of splitting. If the chunk becomes voluminous then it has to be split internally, the problem of different orders of processing the split chunk may crop up as the different orders of processing may lead to different results and we may face the problem of *order effects* [39] and this aspect requires an extensive study, which is being pursued by us.

Since temporal datasets are not available or not so easily available, various methods of simulation of temporal datasets have been explored in [19, 32, 38]. We are inclined to use the one explained in [19] where an offline situation is easily converted to an on-line situation by dividing the given data into smaller batches or chunks and processing these batches one by one to simulate the temporal behavior. Before dividing the data into chunks, we randomly arranged the samples of a given dataset and then divided them into chunks of manageable sizes.

The proposed model for incremental learning through clustering using partial instance memory in a nut shell can be formulated as follows:

(i) The *hard* samples from the just previous chunk which are retained in partial instance memory are appended to the data of current chunk. DBSCAN clustering mechanism is applied on the data to generate knowledge and the overall knowledge is updated; *hard* samples of the present chunk are sent to partial instance memory;

(ii) At the end of arrival of all chunks, knowledge of each of the *hard sample* stored in partial instance memory is extracted and is updated with the overall knowledge.

As an alternative for comparison, we have retained all the hard samples without mixing them with the fresh incoming chunks and processed them at the end after arrival of all chunks.

The Proposed approach

During incremental progress, when a batch or chunk of data arrives, learning is achieved by (i) clustering (ii) extracting the knowledge of clusters to obtain the *batch knowledge* and (iii) updating the overall knowledge which is maintained as *knowledge packet*(s) in the *concept memory*. The *batch knowledge* as well as the given chunk of data samples are discarded except those data samples which have been identified as *hard samples* during the knowledge generation process (clustering). During the process of updating the overall knowledge, the *knowledge packets* of the *batch knowledge* might either get absorbed in the existing *knowledge packet*(s) or might get added as entirely new knowledge packet(s) resulting in accumulation of knowledge packets into the concept memory. If a *knowledge packet* from the *batch knowledge* gets absorbed, the corresponding *knowledge packet* which has absorbed the packet will grow and the distance between the current *knowledge packet* and the existing *knowledge packets* get reduced forcing the *knowledge packets* of the overall knowledge to get merged further. The continued addition of new knowledge packets into the concept memory and merging of the nearest knowledge packets could cause uneven distribution of knowledge requiring splitting of knowledge packet(s) which requires access to the past data which is already discarded. Hence, there should be some mechanism at the time of merging itself which can avoid splitting of the knowledge packet at a later stage.

In case of *data arriving temporally*, a chunk of data would be available at the end of some interval of time. From an ith data chunk or batch $[B]_i$, ith knowledge set $[K]_i$ can be derived. Sooner the $[K]_i$ is available, it has to be merged with the existing knowledge $[Knowledge]_{i-1}$ which is the one updated up to the end of (i-1)th stage; then,

$[Knowledge]_i \leftarrow f([Knowledge]_{i-1}, [K]_i);$

where,

$[Knowledge]_{i-1} \leftarrow f([Knowledge]_{i-2}, [K]_{i-1}),$

$[Knowledge]_{i-2} \leftarrow f([Knowledge]_{i-3}, [K]_{i-2})$ and so on.

Finally overall knowledge is:

$[Knowledge] \leftarrow [Knowledge]_n$ *if* n^{th} stage *is the last stage.*

It has been shown that the knowledge of any kind of data (generic data) can be conveniently represented by a histogram and the knowledge assimilated in histograms can further be compressed by converting them to regression lines [36, 37]. We have identified *number of elements* in each cluster, *mean* (μ), *standard deviation* (σ) and *regression line* as knowledge parameters. The number of elements in each cluster will allow us to keep the knowledge packets in normalized form and simplify the process of obtaining the overall knowledge especially while merging the knowledge packets. Though the presence of regression line nullifies the requirement of μ and σ, we have retained μ for classical requirements and σ for estimating the density for the predefined condition of merging the knowledge packets, which is discussed in detail in the following section. Regression line effectively represents the knowledge of both spherical as well as non-spherical clusters. These parameters are sufficient for easier assimilation of *knowledge packets* and overcome the drawbacks of maintaining the samples that lie on the boundaries of concept descriptions [13] as wells as retaining the consecutive samples over a fixed and changing window of time [11]. For an n-dimensional data space, if the current batch $[B]_i$ shows up 'k' knowledge packets $[Cl_1]$ $[Cl_2]$… $[Cl_k]$, then knowledge structure $[K]_i$ is as shown in Table I.

Therefore, a *knowledge packet* in our research is composed of *number of elments* in a cluster; *mean*, *standard deviation* and regression line (represented by *slope* and *intercept*) of each dimension of the cluster depending upon the number of dimensions in the data.

The new computational model

Here we present the model for incremental learning. In our earlier work [8], a similar model for achieving incremental learning without re-indenting the past data leading to the concept of zero-instance memory learning [2] was proposed. The basic model for generation of knowledge of an ith chunk/batch $[B]_i$ is:

$[DATA]_i \leftarrow ([DATA]_i \cup [\textit{hard samples}]_{i-1})$ --- (1)

$[K]_i + [\textit{hard samples}]_i \leftarrow [DATA]_i$ --- (2)

The basic model for updation of knowledge after the processing of ith batch $[B]_i$ is:

$[Knowledge]_i \leftarrow f([Knowledge]_{i-1}, [K]_i)$ --- (3)

where, $[K]_i$ is the set of knowledge packets obtained by the local cluster analysis on the chunk/batch $[B]_i$ and $[Knowledge]_{i-1}$ is the set of knowledge packets obtained by the incremental agglomeration up to the end of (i-1)th chunk/batch.

Let us assume that updated $[Knowledge]_{i-1}$ has 'p' knowledge packets up to the end of (i-1)th batch.

$[Knowledge]_{i-1} = \{[CL_1], [CL_2], \dots [CL_P]\}$ ---(4)

where, $[CL_X] = \{\mu_X, \sigma_X, L_X\}$;

Let us assume that the present batch of data $[B]_i$ has resulted into *k* number of knowledge packets.

$[K]_i = \{[Cl_1], [Cl_2] \dots [Cl_k]\}$ --- (5)

where, $[Cl_j] = \{\mu_j, \sigma_j, L_j\}$;

The entire process of representation of knowledge packet in terms of histogram, transformation of histogram to normalized histogram then to cumulative histogram and subsequently to normalized regression line and finding distance between regression lines is presented in detail [36] [37]. For the sake of completeness, a summary of the regression distance measure is presented in Appendix.

The creation of $[Knowledge]_i$ is performed by updating $[Knowledge]_{i-1}$ based on the relation (3). To enable this it is required to find the nearest knowledge packets one from $[Knowledge]_{i-1}$and the other from $[K]_i$. The nearest two knowledge packets $[CL_X] \in [Knowledge]_{i-1}$ and $[Cl_j] \in [K]_i$ are accomplished by measuring the distance between the corresponding lines of regression $L_X \in [Knowledge]_{i-1}$ and $L_j \in [K]_i$. If two clusters $[CL_X]$ and $[Cl_j]$ can be merged then the proposed merging operation will result in:

$$\text{new\_}\mu = \left(\frac{(n_j * \mu_j) + (n_X * \mu_X)}{n_j + n_X} \right) \quad \text{--- (6)}$$

new_σ =

$$\sqrt{((n_j * (\sigma_j^2 + d_j^2)) + (n_X * (\sigma_X^2 + d_X^2))) / (n_j + n_X)} \quad \text{--(7)}$$

where, $d_j^2 = (\mu_j - \text{new\_}\mu)^2$; $d_x^2 = (\mu_x - \text{new\_}\mu)^2$;

new_L in terms of slope and intercept is obtained as follows:

$\text{new\_S} = ((n_j * S_j) + (n_x * S_x)) / (n_j+n_x)$ --- (8)

$\text{new\_I} = ((n_j * I_j) + (n_x * I_x)) / (n_j+n_x)$ --- (9)

where, S_j is the slope of the line L_j and S_x is the slope of the line L_x ; I_j is the intercept of line L_j and I_x is the intercept of line L_x.

Proofs for these formulae are available in statistics [41]. These details are also reported in [8].

The merging is allowed based on the following criteria:

Let $q_1 = (\text{new\_}\sigma - \sigma_j)$; and let $q_2 = (\text{new\_}\sigma - \sigma_X)$; then $q = \min(q_1, q_2)$;

Merging of the two *knowledge packets* can be done if (q < = *predefined range*); Here the *predefined range* is the prior knowledge [4] about the density of the dataset.

At the end of this procedure *[Knowledge]$_i$* is created which even if no merge happens then results in (p+k) packets of knowledge. Upon complete merging of the packets the resulting number of *knowledge packets* produced is min(p,k). Therefore the number of *knowledge packets* will be between [*min*(p,k) and (p+k)]. In fact, it is possible that the number of *knowledge packets* at each level could be less than the number of knowledge packets as worked out above, because of induction of continued merging process. We have considered the natural merging only at one level because of the implementation constraints as well as to prevent the possibility of over merging causing a *knowledge packet* to get split later.

The above process of updating continues till all *n* batches are processed. Finally [*Knowledge*] $\leftarrow$ [*Knowledge*]$_n$ = f([*Knowledge*]$_{n-1}$, [*k*]$_n$). However, the process of merging has to be continued to get the optimal number of *knowledge packets* at the end. Finally, the process is terminated when the desired number of *knowledge packets* are produced or the density of the proposed merge becomes greater than the predefined threshold.

While designing the new model, we have also stressed upon retaining the *hard samples* of the previous chunk in the partial instance memory. i.e., at any stage of combining [*Knowledge*]$_{i-1}$ with [*k*]$_i$, the *hard samples* of [*B*]$_{i-1}$ will be utilized. We can totally reject the outliers or *hard samples*, but this will result in loss of information which may lead to inconsistent learning. At every stage, the partial instance memory is emptied and the *hard samples* present in partial instance memory are appended with the next immediate chunk of data samples for further processing. This allows the outliers or *hard samples* of each batch to participate in the knowledge generation process resulting in more accurate final knowledge. The *hard samples* of the last batch are divided into packets of single sample and are processed for merging.

As mentioned earlier, for the sake of comparative study*,* instead of processing the hard samples with the next immediate chunk, we have retained all the hard samples and processed them at the end. For this, the basic model for generation of knowledge of an i^th batch *[B]$_i$* is:

[*K*]$_i$ + [*hard samples*]$_i$ $\leftarrow$ [*DATA*]$_i$ --- (10)

[*hard samples*]$_i$ $\leftarrow$ [*hard samples*]$_i$ $\cup$ [*hard samples*]$_{i-1}$ --- (11)

It should be observed that, a suitable strategy is presented for merging the knowledge packets, but not for splitting a knowledge packet. Our argument is that since merging operation is prevented under unfavorable conditions, the question of subsequent splitting does not arise.

Algorithm

Algorithm: Partial Instance Memory learning using clustering

Input: Temporal flow-in of data in terms of batches; DBSCAN parameters *MinPts* and *Eps*

Output: Clusters of data and their knowledge; and outliers, if any.

Let us assume that a batch of data samples is available at i-th time instant.

Consider the first/initial batch of data; set the parameters *Eps* – the radius that delimitate the neighborhood area of a point and *MinPts* – the minimum number of points that must exist in the *Eps*-neighborhood. Apply the DBSCAN clustering algorithm [34] to find natural groups/clusters in the given data. *Outliers*/*hard data* are to be retained and appended to the next batch of data; let us say 'p' natural clusters are obtained. The 'p' clusters are the 'p' knowledge packets.

```
1. do for all the clusters of initial batch
2. extract_knowledge to obtain 'p' knowledge_packets;
3. end for
4. do for all the knowledge packets of the initial batch
5. find_distance between the knowledge_packets and record them in the distance_matrix;
6. end for
7. do for the batches of data arriving sequentially (Data_t for t = 2 to n)
8. data_to_be_processed = (Data_t U hard_samples_{t-1})
9. Apply the same clustering algorithm on data_to_be_processed; let us say 'k' natural clusters are obtained.
10. do for the 'k' natural clusters obtained
11. extract_knowledge;
12. find_distance between this packet of knowledge to all the knowledge_packets available in the initial batch;
13. Record the packet number which shows the minimum distance; the packet number is destination_pkt_number and the minimum distance is min_val;
14. In the distance_matrix, search the row/column represented by the destination_pkt_number for a value less than min_val.
15. if found
  a. Move this knowledge packet as an additional knowledge packet of the initial batch; this increases the knowledge_packets of the initial batch by one.
  b. Re-compute the distance_matrix;
16. else
  /* Check density_of_the_proposed_merge;*/
  if density_of_the_proposed_merge is in a pre-defined acceptable range
  (i) Merge this knowledge packet with the knowledge packet of the initial batch which is represented by the destination_pkt_number;
  (ii) Re-compute the distance_matrix;
  else
  (iii) Move this knowledge packet as an additional knowledge packet of the initial batch; this increases the knowledge_packets of the initial batch by one.
  (iv) Re-compute the distance_matrix;
  end if;
17. end if
18. end for;
19. end for;
20. do while no. of knowledge packets > Goal
21. find the packets with minimum distance;
```

```
22. if they can be merged as per the density_of_the_proposed_merge, merge
    them;
23. reduce the number of knowledge packets by one;
24. Re-compute the distance_matrix;
25. else set minimum distance to a very high value;
26. end if;
27. end for;
28. end Partial Instance Memory Learning
```

Algorithm: extract_knowledge

```
1. do for each cluster
2. keep the number of elements as knowledge;
3. do for each dimension of cluster
4. get the mean ($\mu$); get the standard deviation ($\sigma$);
5. normalize the data values and fix up the number of bins.
6. construct a histogram; Obtain a cumulative histogram; Normalize the
   cumulative histogram and fit a first order polynomial to the normalized
   cumulative histogram to obtain a regression line ($L$);
7. end do;
8. end do;
9. return $\mu, \sigma, L$;
10. end extract_knowledge;
```

Algorithm: find_distance

/* from a packet of knowledge to all other packets of knowledge */

```
1.  distance[1] = 0;
2.  do for know_pkts = 2 to p
3.  initial_distance = 0;
4.  do for dimensions = 1 to d
5.  dist[dimensions] = distance_between_lines(line[dimensions], pkt[1],
    line[dimensions], pkt[know_pkts], n1, n2);
6.  initial_distance = initial_distance + dist[dimensions];
7.  end for
8.  distance[know_pkts] = initial_distance;
9.  end for;
10. return distance;
11. end /* find_distance*/
```

Algorithm: distance_between_lines(line[ ], pkt[1],line [ ], pkt[ ], n1,n2)

/* n1 = no of elements in line 1 and n2 = no of elements in line 2 */

```
1.  If two lines do not intersect
2.  dist1=distance between the first end points of given lines;
3.  dist2=distance between the second end points of given lines;
4.  else
5.  get the point of intersection of the two lines ($X_i, Y_i$);
6.  dist1= (distance between the first end points of given lines * $Y_i$);
7.  dist2= (distance between the second end points of given lines * (1- $Y_i$));
8.  end /* if */
9.  area_bw_lines = (dist1 + dist2)/2;
10. len1 = length of line1;
11. len2 = length of line2;
12. behavior_of_lines = abs(len1 – len2);
13. distance_bw_line1_and_line2 = ($\alpha$ * area_bw_lines) + (1- $\alpha$) *
    behavior_of_lines;
    /* $\alpha$ is a tunable parameter between 0 and 1 */
14. return distance_bw_line1_and_line2;
15. end /* distance_between_lines */
```

Complexity Analysis

(i) Time complexity of DBSCAN clustering mechanism is O(*n*log*n*) [16, 34, 42] where '*n*' is the number of samples in each chunk; If there are '*k*' number of batches, then the time complexity becomes *k* * (*n*log*n*);

(ii) Extracting knowledge of each cluster:

If there are *m* clusters in a batch and if there are *d* dimensions of each cluster, then for extracting knowledge of each batch, the time required is O(*md*);

For k number of batches, the time required to extract knowledge is *k* * (*md*)

(iii) Finding distance between knowledge packets:

(a) If there are *m* knowledge packets obtained from the initial batch, time required to find distance between them is *m (m*-1*)/2*;

(b) Finding the distance between each packet of knowledge of the subsequent batches with the *m* knowledge packets of the initial batch:

1. Finding distance between a packet of knowledge to *m* knowledge packets of initial batch is 1* *m*; If there are m_1 knowledge packets of the subsequent batch, finding distance between these m_1 knowledge packets with the *m* knowledge packets of the initial batch, the time required is *m* * m_1;
2. Re-computing the distance matrix whether the packet of knowledge of the subsequent batch is merged with the initial batch or the packet of knowledge is moved as an additional knowledge packet of initial batch is *m (m*-1*)/2*;
3. There are *k*-1 number of subsequent batches, then the distance between each packet of knowledge of the subsequent batches with the *m* knowledge packets of the initial batch is: [(*k*-1*)* * (*m* * m_1) * *m (m*-1*)/2*];

In all, the complexity is: *m (m*-1*)/2* + [(*k*-1*)* * (*m* * m_1) * *m (m*-1*)/2*];

(iv)Time for final merging of knowledge packets: The dominant operation in this process is the re-computing of the distance matrix; hence the time required is *m (m*-1*)/2*;

The Overall complexity is:

$[k * (n\log n)] + [k * (md)] + [m\,(m-1)/2 + [(k-1) * (m * m_1) * m\,(m-1)/2]] + [m\,(m-1)/2]$;

$\rightarrow O\,(m^3 n)$;

It should be noted that m is the number of knowledge packets. Further since $m = n$, we can assume that m is always bound by a constant. In practical scenario it is also true that one could be interested in designing predefined number of knowledge packets only. Thus the above expression may be simplified to O ($\lambda\, n$) where, λ is in terms of m. This implies that the scan through the data samples is only once and the time of computation also depends upon the knowledge packets contained in the data space.

An Illustrative Example for Partial Instance Memory Learning using clustering

The synthetic dataset has been used for clear understanding and a visual demonstration of the proposed method. The synthetic dataset of 5000 points in a 2D space distributed over five classes is shown in Fig 1. Similar dataset has been used in to simulate the stream of data [31]. For exhaustive profile analysis, we have also conducted experiments on synthetic dataset by (i) keeping the number of samples in a batch constant and varying the total number of samples (ii) keeping the total number of samples constant and varying the number of processing batches. Also for supervised experimentation we have varied the number of knowledge packets present in the overall data mass.

We randomly arranged the samples of the dataset and segmented the dataset into four batches of equal size and processed the four batches one at a time in a sequential order simulating the temporal arrival of data. For DBSCAN, *Eps* was set to 0.25 and *MinPts* was set to 20; the threshold for merging of batches was set to 0.15. Batch B_1 was considered as the initial batch. Upon applying DBSCAN, we obtained 22 clusters as shown in Fig 2.

In Fig 2, the first row indicates cluster numbers 1 to 7, the second row indicates 8 to 14, third row indicates 15 to 21 and the last row cluster number 22. The same representation scheme is followed for other figures which represent the clusters. Knowledge of all these clusters was extracted to obtain the 22 knowledge packets.

Out of the 1250 elements of batch B_1, 566 elements were marked as outliers (because of the stringent parameters *Eps* and *MinPts* of DBSCAN) and appended to the second batch B_2 as per the requirement of our proposed method of PIML. From this batch B_2, we obtained 39 clusters as shown in Fig 3. Knowledge of these clusters was extracted to obtain 39 knowledge packets.

Out of the 39 knowledge packets obtained from this batch, 24 got merged with some of the knowledge packets of batch B_1, which is now K_1, using the *Histo-Regression-Distance* [36, 37]. The fifteen unmerged knowledge packets of B_2 are appended to the *concept memory* which brings the total number of knowledge packets to 37 (22 + 15) at the end of batch B_2. The clusters represented by these knowledge packets are shown in Fig 4. As the data samples of these clusters are assumed to be not available, just for visualization, these clusters have been projected here.

Because of the merging of 24 knowledge packets into some of the knowledge packets of K_1, the knowledge packets of K_1 have grown and may attract the existing knowledge packets forcing internal merging of the knowledge packets. At the first level of internal merging of the knowledge packets at the end of B_2, the number of knowledge packets were reduced to 25 which is now K_2 and the clusters of these knowledge packets are shown in Fig 5.

Clusters obtained by batch B_3 are shown in Fig 6. From these clusters, knowledge packets are obtained.

Nineteen knowledge packets of this batch got merged into some of the existing knowledge packets of K_2. The six unmerged knowledge packets of this batch are appended to the *concept memory*, which brings the total knowledge packets to 31 (25 + 6) and after internal merging there are 21 knowledge packets (after B_3) which is now K_3 and the corresponding clusters of the knowledge packets of K_3 are shown in Fig 7.

Similarly, the clusters obtained from B_4 are shown in Fig 8 and the corresponding knowledge packets are obtained. As usual, some of the knowledge packets got merged into the existing knowledge packets of K_3 and after internal merging (because of the growth in knowledge packets), we obtained twenty four knowledge packets and the corresponding clusters are shown in Fig 9. Each *hard sample* of B_4 is considered as an individual knowledge packet and is merged into the existing knowledge packets. Many knowledge packets got merged into the knowledge packets of K_4 and these knowledge packets merged further resulting in 29 knowledge packets; the clusters of these knowledge packets are shown in Fig 10. In this figure, we can easily observe the five dominating clusters.

Dynamic merging of the nearest knowledge packets repeatedly until the desired number of knowledge packets are obtained or the desired threshold is met, yields the five knowledge packets. The corresponding clusters of these five knowledge packets are shown in Fig 11.

The details of the number of knowledge packets, number of *hard samples* at each stage, and number of unmerged clusters are shown in Table II.

From Table II, it is clear that even though at each stage few knowledge packets are getting added, the overall number of knowledge packets is getting reduced by the internal merging operation and thus minimizing the concept memory. Also, the outliers of one stage are getting merged with the succeeding batch, thus reducing the requirement of partial instance memory.

Table III gives the summary of deviation of knowledge parameters of all the five knowledge packets. It is clear that a deviation of less than 1% in almost all the knowledge parameters is negligibly small considering that the knowledge has been gathered incrementally which should have suffered a significant estimation error.

Table IV shows the comparison of deviation in knowledge parameters with zero instance memory learning [8] and the present partial instance memory learning. Since the *hard samples* at each stage of learning was assumed unavailable in zero instance memory learning, the deviation in knowledge parameters is higher. Still this deviation can be considered as minimal since a lot of time and space could be saved. Further, if the number of *hard samples* at each stage would have been zero, then the results obtained by zero instance memory learning would have been same as partial instance memory learning.

The percentage of average deviation in knowledge parameters of zero and partial instance memory learning is shown in Fig 12.

It is clear that there is significant improvement in the final knowledge parameters in partial instance memory when compared with zero instance memory [8]. This is at the cost of additional memory for maintaining the *hard samples* and computational time for reprocessing of *hard samples* at each stage of learning. An average deviation of less than 1% in the knowledge parameters of partial instance memory is meager when the knowledge is fused in a piecemeal fashion. This indicates the power of histogram based regression line as a parameter for knowledge representation and the suitability of *histo-regression-distance* [36, 37] for incremental learning through clustering.

As mentioned earlier, instead of processing the *hard samples* of a chunk with samples of the next immediate chunk, if all the *hard samples* are accumulated and processed at the end, the deviation in knowledge parameters up to nth batch is similar to that of the zero instance memory learning. However, when all the accumulated *hard samples* are processed, we get the values for the knowledge parameters similar to the one shown in Table III.

In order to get clear idea about the average size of the partial instance memory, we have also conducted experiments by dividing the synthetic dataset of 5000 points into (i) 5 batches of 1000 samples each (ii) 8 batches of 625 samples each (iii) 10 batches of 500 samples each and (iv) 20 batches of 250 samples each and processed them in different orders. The average number of samples retained in partial memory when the given dataset of 5000 samples is divided into 4, 5, 8, 10 and 20 batches of different sizes is depicted in Fig 13. From the graph, it is clear that the average number of samples maintained in partial instance memory is less than 500. For the experiments with varying number of batches as well as with different batch sizes, the results obtained are similar.

A similar experiment of dividing the dataset of 5000 samples into 5 batches of 1000 samples each, 8 batches of 625 samples each, 10 batches of 500 samples and 20 batches of 250 samples each, was conducted. The growing size of the partial instance memory is depicted in Fig 14.

From Fig 14, it is clear that if the batch size is smaller as for example, the batch sizes of 625, 500 and 250, the entire data of all batches is considered as *hard samples* and gets accumulated in the partial instance memory. Only when all the batches have arrived, the process of learning begins with accumulated *hard samples*; in which case learning is no longer incremental. Even after waiting for the batches to arrive and even after wasting lot of effort in trying to group the elements of each batch unsuccessfully, we still have to process a larger group of elements which requires lot of memory and the available memory, though large is finite [38].

Worst will be the scenario if the process of learning is based on zero instance memory, as no knowledge is generated at both intermediate as well as at the final stages.

Experiments were also conducted by keeping the number of samples in a batch as constant and varying the total number of samples. The batch size was fixed at 1000. The initial dataset of 5000 samples was increased to 10,000 samples with 2000 samples in each class. Further the total number of samples increased to 15,000, 20000, 30000, 40000 and 50000 with 3000, 4000, 6000, 8000 and 10000 samples respectively in each class. The average of total number of knowledge packets obtained when the number of batches was 5, 10, 15, 20, 30, 40 and 50 were projected in Fig 15. It is clear that requirement of memory to store concepts has got stabilized. Actually in principle, the number of knowledge packets should die down with the increase in the number of batches. However, as mentioned earlier, because of the internal merging of only one level to avoid over merging, we have obtained only the stability.

Result obtained with a similar experiment by accumulating all the *hard samples* in partial instance memory instead of processing them with the next immediate batch or chunk, is depicted in Fig 16. It is clear that the requirement of partial instance memory keeps on increasing with the increase in number of batches. Further there will be a deviation in the intermediate knowledge as the *hard samples* are not allowed to participate at all levels.

We have also conducted experiments by increasing the number of classes to six, seven and up to ten and the results obtained was similar for average number of *hard samples*.

Experimentation

We have conducted experiments on three bench mark datasets –(i) Temperature dataset [43, 44] which is a twenty four dimensional data of 37 samples; (ii) Iris [45] which is a four dimensional data of 150 samples distributed over three classes; and (iii) 700X3 dataset [37] which is a dataset of three dimensions with 700 samples distributed over five classes.

Temperature dataset

Average of daily minimum and maximum temperatures of each month from January to December covering 37 cities all over the globe was considered to be the data with 24 features (maximum, minimum of each month for 12 months). This data can be found in any engagement book or dairy. This data was used for clustering [43, 44]. This is a typical pattern recognition problem. Several classes are present in the data since the data covers many zones (tropical, temperature, equatorial, frigid etc).

If we process all the 37 cities with the data of each month as a batch or a chunk, then the cities will change their classes in the subsequent stages of learning, which requires splitting of knowledge packets as well as to address the *order effects* [39]. Hence in this study, we are not considering the cases where samples will be changing their classes over a period of time. For example, a child moves from the class childhood to adult over a period of time; a seed changing its class to plant/tree over a period of time and so on. In this study, for the present temperature dataset we assume that all the details (temperature of 12 months) of the Asian cities are available at one point of time, all European cities at another point of time and so on.

From the entire data set, with the DBSCAN parameters *MinPts* set to 2 and *Eps* set to 25, we obtained 7 clusters out of which 3 clusters of one element each (samples that have been marked as outliers are considered as a cluster of one element each with a standard deviation of zero). Further as explained earlier, the data has been divided into 4 batches. Three batches to have 9 samples each and one batch to have 10 samples. Threshold for merging of batches is set to 0.5. The details of the number of clusters, outliers at each stage, number of unmerged clusters are shown in Table V.

Table VI gives the summary of deviation of knowledge parameters of all the seven knowledge packets. Since the number of features is more, the average deviation has been projected. Since all the elements of the knowledge packets 4, 5, 6 and 7 remain same the deviation is at its lowest level. However, since some of the elements of other knowledge packets have got interchanged with that of the originals, the deviation is on a slightly higher side. This is because of the fact that once an element is moved to a packet, in the next batch of samples some elements may not get the desired MinPts required for the DBSCAN clustering algorithm and may remain as outliers. Only at the final stages of merging, these points may merge with the nearest packet resulting in some deviation.

IRIS data [45]

The standard iris dataset [45] has150 points in 4-dimensional space. First 50 samples belong to class 1; the second 50 belong to class 2 and the third 50 belong to class 3; Class 1 is clearly separable from class 2 and 3, whereas class 2 and 3 are not separable. This dataset has been used extensively to study the behavior of different clustering algorithms. With the DBSCAN parameters *MinPts* set to 2, *Eps* set to 1.4, we obtained 3 clusters and zero outliers from the entire dataset of 150 samples. As explained earlier, the samples were arranged in random order and divided into six batches of equal size (25 elements each) and processed the batches in sequential order to simulate the temporal arrival of data. Threshold for merging of batches set to 0.065.

Details of the number of clusters, outliers at each stage, number of unmerged clusters are shown in Table VII. Dynamic merging of the nearest batches yields three knowledge packets. Table VIII gives the summary of average deviation of knowledge parameters of all the three knowledge packets. Since Packet 1 is clearly separable, we got the minimum deviation.

700X3 Dataset [37]

This dataset has 700 elements with 3 features. There are five classes with 140 samples in each class. First 140 samples belong to class 1, next 140 belong to class 2 and so on. To get the clear separation of classes all the three parameters are mandatory. This dataset has been used as a regression line symbolic sample set and has been established that the first two principle components are good enough to classify this symbolic dataset [37]. With the DBSCAN parameters *MinPts* set to 2, *Eps* set to 6, we obtained 5 clusters and zero outliers from the entire dataset of 700 samples. Further, we have randomly arranged the samples and divided them into 7 batches of 100 samples each. We set the threshold for merging of batches to 0.3.

Details of number of clusters, outliers at each stage and number of unmerged clusters are shown in Table IX. Repeated merging yields the five knowledge packets. We have obtained very good results with an accuracy of 100% in regression line parameters of the knowledge packets and are shown in Table X.

Conclusion

If the requirement is to understand the trend in the data as the observations keep arriving in case of temporal chunks, then it is necessary to extract the knowledge at the arrival of every chunk of data and keep augmenting the knowledge status with the progress in arrival of data chunks in temporal sequence. This warrants incremental learning. In fact incremental learning is also the best solution since one cannot handle the pile of data that could get pooled if one desires to employ one-shot learning and further, the time gap between the arrival of two successive chunks, in many cases should be more than enough to complete the processing of data of the chunk. However, incremental learning has problems in terms of accumulating the knowledge and carrying forward the hard data samples. A new model of incremental learning for generation of batch knowledge and for augmentation of knowledge with the arrival of subsequent batch of data has been presented in this research work. For the purpose we have employed DBSCAN procedure to generate clusters and *Cluster-Histo-Regression* is used for getting the knowledge packets and *Histo-Regression-Distance* is used for incremental augmentation of knowledge packets. The regression line parameters of the knowledge packets obtained by incremental augmentation have shown an accuracy of 100% in some of the experiments conducted on standard datasets. Applications which involve backtracking might not be suitable since the *batch knowledge* is discarded once the *overall knowledge* is updated and there is no option to delete the elements from the knowledge packets. Further, only one level of internal merging could be a drawback of the proposed strategy as optimal description of the accumulated knowledge at any intermediate level may not be clearly visible.

Appendix

The regression distance measure proposed by [36, 37] is summarized below.

Consider a histogram H with 10 bins; H = {b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}} where b_i is the frequency count of the bin centered at C_i. For example the corresponding histogram for the data say A = {10 30 40 50 40 30 20 20 30 10} is as shown in Fig 17. Now a cumulative frequency distribution is computed for each of these 10 centers resulting in the cumulative histogram (CH); CH = {cn_1, cn_2, cn_3, cn_4, cn_5, cn_6, cn_7, cn_8, cn_9, cn_{10}} where cn_i = sum (cn_k) for k = 0 to i; for the histogram of Fig 17, CH becomes {10 40 80 130 170 200 220 240 270 280} and the cumulative histogram is as shown in Fig 18.

CH is then normalized by dividing cn_i for i = 1 to 10 by cn_{10}. Now 10 points are marked on the top of each bin in the CH corresponding to the bin centers and a first order polynomial is fitted across these 10 points to obtain regression line with y_i ranging between 0 and 1 and x_i's range is decided by the minimum and maximum co-efficient values at a particular scale. The regression line fitting is done as shown in Fig 19.

Distance between such obtained lines can be computed by finding the area between the two lines as well as by finding the behavior of the two lines. The calculation of distances is as shown in Fig 20.

Acknowledgement

Authors would like to thank Caledonian College of Engineering, Muscat, Sultanate of Oman for their encouragement to take up this research work. Authors would also like to thank the research scholars – Mr. Manjunatha S, Mr. Harish B.S, Mr. Prabhudev Jagdeesh, Mrs. Nirmala Shivanada; immensely for their continuous support during the implementation of the experiments. Authors would also like to thank Dr. Naushath of HP Labs, Mr. Siva Ramakrishna of Indian Institute of Science and Mr. B. Raghavendra for their support in literature review.

References

[1] Christophe G.C. (2000) *AI Communications*, 13(4), 215-223.

[2] Maloof A.M., Michalski R.S. (2004) *Artificial Intelligence* 154, 95-126.

[3] Kaufman K.A., Michalski R.S. (2004) *Reports of the Machine Learning and Inference Laboratory,* MLI 04-4.

[4] Michalski. R.S. (2003) *Invited talk at the Sanken Symposium on Data Mining and Semantic Web*, *Osaka university, Japan*.

[5] Jain A.K., Dubes R.C. (1998) *Prentice Hall*, *Englewood Cliffs, NJ*.

[6] Jain A.K, Murthy M.N., Flynn P.J. (1999) *ACM Computing surveys*, 31(3), 264-323.

[7] Yuanhong Li, Ming Dong, Jing Hua (2008) *Pattern recognition Letters* 29, 10-18.

[8] Syed Zakir Ali, Nagabhushan P., Pradeep Kumar R. (2009) *International Conference on Data Mining (DMIN-09),* 375-381.

[9] Kibler D. Aha (1987) *Proceedings of the Fourth International Conference on Machine Learning*, *Morgan Kauffmann, San Francisco, CA*, 24-30.

[10] Sebastian Luhr, Mihai Lazarescu (2009) *Data and Knowledge Engineering* 68, 1-27.

[11] Widmer G., Kubat M. (1996) *Machine Learning* 23, 69-101.

[12] Widmer G. (1997) *Machine Learning* 27, 259-286.

[13] Maloof A.M., Michalski R.S. (2000) *Machine Learning* 41, 27-52.

[14] Sigita Misina (2006) *Proceedings of the International Conference on Computational Intelligence, Theory and Applications*, *9th Fuzzy days in Dortmund, Germany*, 20, 545-553.

[15] Tian Zhang, Raghu Ramakrishnan, Miron Livny (1996) *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, *Montreal, Canada*, 103-114.

[16] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Michael Wimmer, Xiaowei X. (1998) *Proceedings of the 24th VLDB conference New York*, 323-333.

[17] Fazli C., Edward A.F., Cory D.S., Robert K.F. (1995) *International Journal of Information Sciences—Informatics and Computer Science*, 84, 101-114.

[18] Khalid M. H., Mohammed S. K. (2003) *IEEE/WIC International Conference on Web Intelligence, Halifax, Canada*, 597-601.

[19] Chaudhuri B.B. (1994) *Pattern Recognition Letters* 15, 27-34.

[20] Narasimhamurthy M., Sridhar V. (1991) *Pattern Recognition Letters* 12, 511-517.

[21] Martin H.C.L. (2006) *Doctoral Dissertation of the Michigan State University*.

[22] Nong Y., Xiangyang L. (2002) *Journal of Computers and Industrial Engineering* 43, 677-692.

[23] Seiichi ozawa, Nikola kasabov (2008) *IEEE Transactions on Neural Network*, 1061-1074.

[24] Sarda N.L., Srinivas N.V. (1998) *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, 240 – 245.

[25] Ahmed M. A., Nagwa M.E.M., Yousry Taha (2001) *Proceeding of the 1st SIAM conference on Data Mining, Chicago, IL*.

[26] Masseglia F., Poncelet P., Teisseire M. (2003) *Data & Knowledge Engineering*, 46(1), 97-121

[27] Eric C., Akanksha B., Tig C., Anhai D., Jeffrey N. (2007) *Proceedings of the 33rd International Conference on VLDB*, 1045-1056.

[28] Jieping Y, Xiong H, Haesun P, Ravi J., Vipin K. (2005) *IEEE Transaction on Knowledge and Data Engineering*, 17(9), 1208-1222.

[29] Kolonia P. (1994) *Popular Photography*, 58(1), 30-34.

[30] Charu C. A., Jiawei Han, Jianyong Wang, Phillip S. Yu. (2003) *Proceedings of the 29th VLDB Conference,* 81-92.

[31] Liaden O'Callagahan, Nina Mishra, Adam Meyerson , Sudipto Guha, Rajeev Motwani (2002) *Proceedings of the ICDE*, 685-694.

[32] Han Kamber (2006) *Second Edition, Elsevier*, 51 -56.

[33] Soman K.P., Shyam Diwakar, Ajay V. (2006) *Prentice Hall of India*.

[34] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu. (1996) *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96),* 226-231.

[35] Diday E., (2002), *Electronic Journal of Symbolic Data Analysis*, 1-25.

[36] Pradeep Kumar R., Nagabhushan P. (2007) *Engineering Letters*, 14(1), EL_14_1_30, www.engineeringletters.com/issues_v14/issue_1/EL_14_1_30.pdf

[37] Pradeep Kumar R. (2006) *PhD Thesis of the University of Mysore, India*.

[38] Tsai-Hung Fan, Dennis K.J. Lin, Kuang-Fu Cheng (2007) *Data and Knowledge Engineering* 61, Elsevier, 554-562.

[39] Langley P. (1995) *P. Reimann & H. Spada (Eds), Elsevier, Amsterdam*.

[40] John F. Roddick, Myra Spiliopoulou (2002) *IEEE Transactions on Knowledge and Data Engineering*, 4(4), 301-316.

[41] Michael J. Parik (2008) *Elsevier Academic Press*.

[42] Yi-Pu Wu, Jin-Jiang Guo, Xue-Jie Zhang (2007) *International Conference on Machine Learning and Cybernetics*, 5, 19(22), 2608-2614.

[43] Bapu B. K. (2004) *PhD thesis of the University of Mysore, India*.

[44] Lalitha Rangarajan (2004) *PhD Thesis of the University of Mysore, India*.

[45] UCI Machine Learning Repository, http://archieve.ics.uci.edu/ml/datasets/Iris

Table I-   Knowledge parameters for incremental learning

[Bi]	No. of elements	f1			f2			…	fn		
[Cl1]	*n1*	${}^{i}\mu_1^1$	${}^{i}\sigma_1^1$	${}^{i}L_1^1$	${}^{i}\mu_1^2$	${}^{i}\sigma_1^2$	${}^{i}L_1^2$	…	${}^{i}\mu_1^n$	${}^{i}\sigma_1^n$	${}^{i}L_1^n$
[Cl2]	*n2*	${}^{i}\mu_2^1$	${}^{i}\sigma_2^1$	${}^{i}L_2^1$	${}^{i}\mu_2^2$	${}^{i}\sigma_2^2$	${}^{i}L_2^2$	…	${}^{i}\mu_2^n$	${}^{i}\sigma_2^n$	${}^{i}L_2^n$
…	…	..	…	…	..	…	…	…	..	…	…
[Clk]	*nk*	${}^{i}\mu_k^1$	${}^{i}\sigma_k^1$	${}^{i}L_k^1$	${}^{i}\mu_k^2$	${}^{i}\sigma_k^2$	${}^{i}L_k^2$	…	${}^{i}\mu_k^n$	${}^{i}\sigma_k^n$	${}^{i}L_k^n$

Where, *μ* – Mean; *σ* – Standard deviation/Density;
L – Regression Line in terms of slope and intercept; *Cl* – Cluster;

Table II- Details of knowledge packets and outliers in PIML from Synthetic data

Batch No	No. of elements	No. of outliers	No. of knowledge packets	No. of unmerged knowledge Packets	No. of knowledge packets before internal merging	No. of knowledge packets after internal merging
1	1250	566	22	--	22	22
2	1250+566	219	39	15	(22+15)=37	25
3	1250+219	517	25	06	(25+6) =31	21
4	1250+517	299	40	10	(21+10)=31	24
			299	17	(24+17)=41	29

Table III- Summary of deviation of knowledge parameters of Synthetic dataset in Partial Instance Memory learning

Knowledge Parameter	Values	Packet 1		Packet 2		Packet 3		Packet 4		Packet 5	
		f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
μ	Actual	1.0034	0.9966	4.0025	0.9958	2.5040	2.4973	1.0025	3.9958	4.0032	3.9965
	Computed	1.0060	0.9990	4.0069	0.9911	2.5075	2.5009	0.9989	3.9992	4.0039	3.9962
	Difference (%)	0.25	0.24	0.1	0.4	0.1	0.1	0.3	0.08	0.01	0.007
σ	Actual	0.5401	0.4533	0.5393	0.4527	0.5400	0.4533	0.5393	0.4527	0.5381	0.4348
	Computed	0.5483	0.4556	0.5187	0.4525	0.5404	0.4521	0.5410	0.4538	0.5425	0.4449
	Difference (%)	1.5	0.5	3.8	0.04	0.07	0.26	0.3	0.2	0.8	2.3
Slope	Actual	0.1428	0.1413	0.2153	0.1412	0.2881	0.2870	0.1427	0.2148	0.2152	0.2147
	Computed	0.1431	0.1417	0.2147	0.1402	0.2880	0.2870	0.1420	0.2144	0.2151	0.2147
	Difference (%)	0.2	0.28	0.27	0.7	0.03	1.4	0.4	0.18	0.04	0
Intercept	Actual	0.4928	0.4971	-0.2883	0.4976	-0.1706	-0.1675	0.4932	-0.2864	-0.2882	-0.2864
	Computed	0.4915	0.4958	-0.2880	0.5006	-0.1712	-0.1677	0.4958	-0.2862	-0.2882	-0.2863
	Difference (%)	0.26	0.26	0.1	0.6	0.35	0.11	0.5	0.06	0	0.03

Table IV- Comparison of Deviation of Knowledge Parameters in Zero Instance Memory [8] and Partial Instance Memory

Knowledge Packet No.	Feature	Percentage of deviation in knowledge parameters							
		Zero Instance Memory				Partial Instance Memory			
		μ	σ	Slope	Intercept	μ	σ	Slope	Intercept
1	f_1	1	22	0	0	0.25	1.5	0.2	0.26
	f_2	5	2	7	4	0.24	0.5	0.28	0.26
2	f_1	2.7	24	4.7	6	0.1	3.8	0.27	0.1
	f_2	4	6.6	7.1	0	0.4	0.04	0.7	0.6
3	f_1	0	16.6	0	0	0.1	0.07	0.03	0.35
	f_2	3.2	8.8	3.4	11.7	0.1	0.26	1.4	0.11
4	f_1	5	24	7.1	8.1	0.3	0.3	0.4	0.5
	f_2	0.75	8.8	4.7	0	0.08	0.2	0.18	0.06
5	f_1	1	24	4.7	3.4	0.01	0.8	0.04	0
	f_2	1.7	11.6	0	3.4	0.007	2.3	0	0.03

Table V- Details of knowledge packets, outliers in partial instance memory learning from temperature dataset

Batch No	Number of elements	Outliers	Number of knowledge packets	Number of unmerged knowledge packets	Number of knowledge packets at the end of each batch
1	9	9	0	Not applicable	00
2	10+9	5	3	00	03
3	9+5	5	3	01	04
4	9+5	10	1	00	04+(10 clusters of one element each)

Table VI-Summary of deviation of knowledge parameters of temperature data

Knowledge parameters	Values	Pkt 1	Pkt 2	Pkt 3	Pkt 4	Pkt 5	Pkt 6	Pkt 7
M	Actual	9.50	18.52	26.27	18.88	10.45	18.50	10.45
	Computed	9.40	15.10	25.52	18.88	10.45	18.50	10.45
	Difference (%)	1	18	2.8	0	0	0	0
σ	Actual	3.57	4.18	3.19	2.92	0.00	0.00	0.00
	Computed	3.59	2.80	3.69	2.92	0.00	0.00	0.00
	Difference (%)	0.5	33	15.6	0	0	0	0
Slope	Actual	0.02	0.02	0.02	0.02	0.01	0.02	0.01
	Computed	0.02	0.02	0.02	0.02	0.01	0.02	0.01
	Difference (%)	0	0	0	0	0	0	0
Intercept	Actual	0.33	0.17	0.07	0.15	0.33	0.20	0.33
	Computed	0.33	0.22	0.08	0.15	0.33	0.20	0.33
	Difference (%)	0	29	14.2	0	0	0	0

Table VII-Details of knowledge packets, outliers, for iris dataset

Batch No	Number of elements	Outliers	Number of knowledge packets	Number of unmerged knowledge packets	Number of knowledge packets at the end of each batch
1	25	2	2	Not applicable	2
2	25+2	0	3	01	3
3	25+0	0	3	01	4
4	25+0	0	3	01	5
5	25+0	0	2	00	5
6	25+0	1	2	00	05+(1 knowledge packet of one element each)

Table VIII-Summary of average deviation of knowledge parameters of Iris data

Knowledge Parameters	Values	Packet 1	Packet 2	Packet 3
μ	Actual	2.5330	4.1224	3.2815
	Computed	2.5330	3.9497	3.2583
	Difference (%)	0	4.1	0.7
σ	Actual	0.2510	0.4662	0.2724
	Computed	0.2624	0.5661	0.3237
	Difference (%)	4.5	21	18.8
Slope	Actual	0.0658	0.1055	0.0871
	Computed	0.0658	0.1017	0.0874
	Difference (%)	0	3.6	0.3
Intercept	Actual	0.4753	0.1134	0.2993
	Computed	0.4753	0.1519	0.2994
	Difference (%)	0	33.9	0.03

Table IX-Details of knowledge packets, outliers for 700 X 3 dataset

Batch No	Number of elements	Outliers	Number of knowledge packets	Number of unmerged knowledge packets	Number of knowledge packets at the end of each batch
1	100	0	6	Not applicable	6
2	100+0	0	6	00	6
3	100+0	1	6	00	6
4	100+1	0	6	00	6
5	100+0	0	6	00	6
6	100+0	0	6	00	6
7	100+0	0	6	00	6

Table X-Summary of average deviation of knowledge parameters of 700 X 3 data

Knowledge Parameters	Values	Pkt 1	Pkt 2	Pkt 3	Pkt 4	Pkt 5
μ	Actual	49.8163	45.1366	40.4568	45.1366	41.4879
	Computed	49.8163	45.1366	40.4568	45.1366	41.4879
	Difference (%)	0	0	0	0	0
σ	Actual	2.3142	2.1091	1.9041	2.1091	1.9635
	Computed	2.3684	2.1581	1.9505	2.1624	2.0051
	Difference (%)	2.3	2.3	2.4	2.5	2.1
Slope	Actual	0.0079	0.0067	0.0055	0.0067	0.0094
	Computed	0.0079	0.0067	0.0055	0.0067	0.0094
	Difference (%)	0	0	0	0	0
Intercept	Actual	0.1610	0.2591	0.3572	0.2591	0.1481
	Computed	0.1610	0.2591	0.3572	0.2591	0.1481
	Difference (%)	0	0	0	0	0

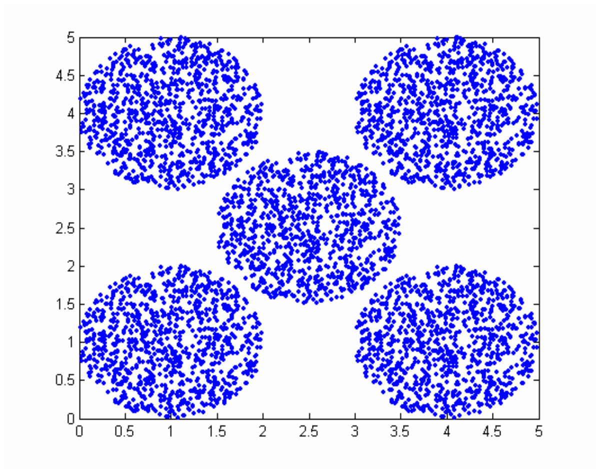


Fig. 1- Initial Dataset

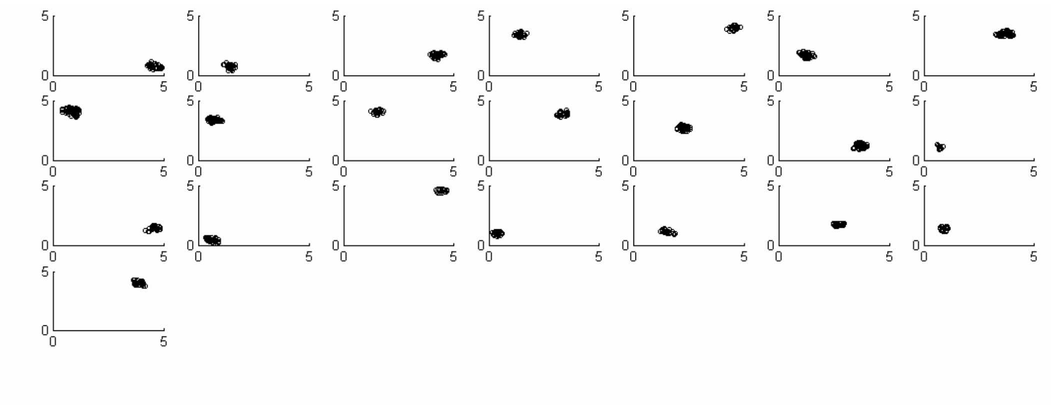


Fig. 2- Twenty-Two Clusters of batch B_1

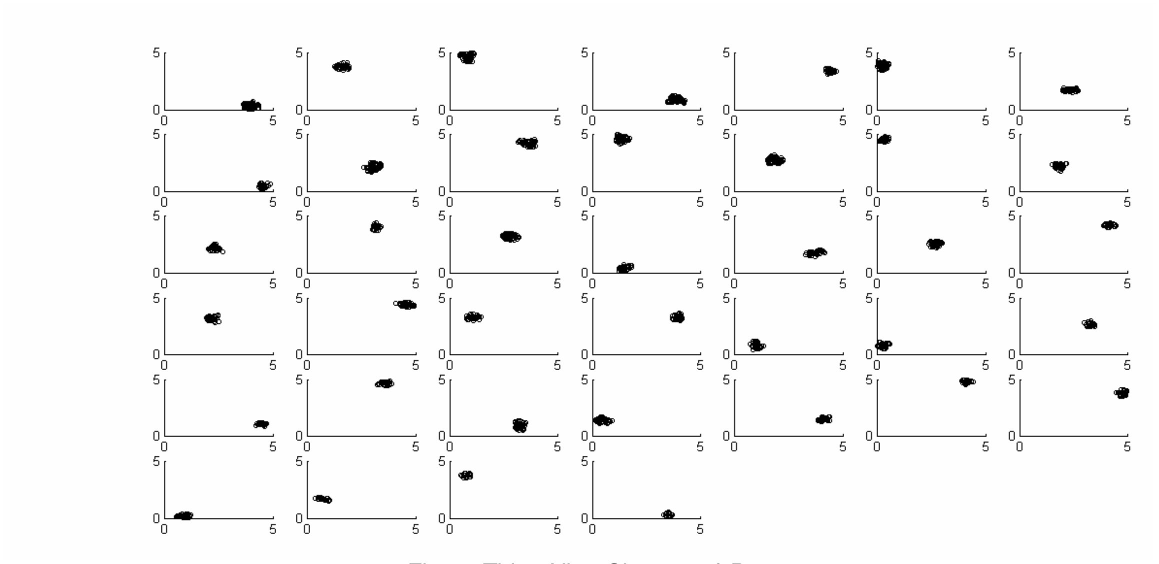


Fig. 3- Thirty Nine Clusters of B_2

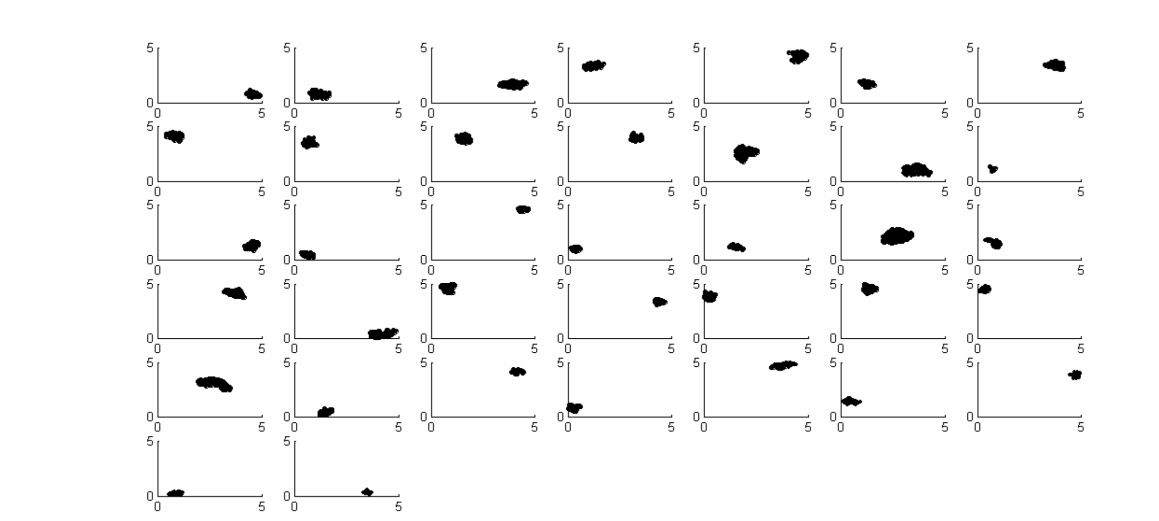


Fig. 4- Thirty Seven Clusters at the end of B_2

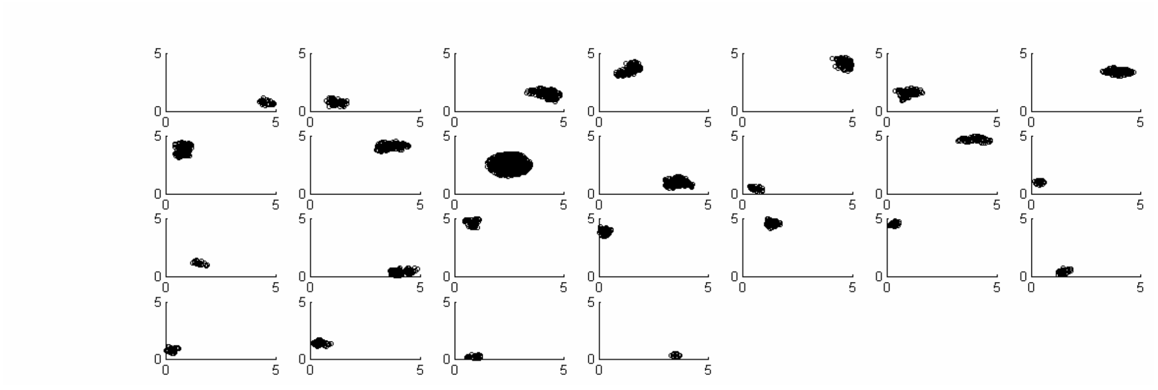


Fig. 5- Clusters at the end of B_2 after internal merging

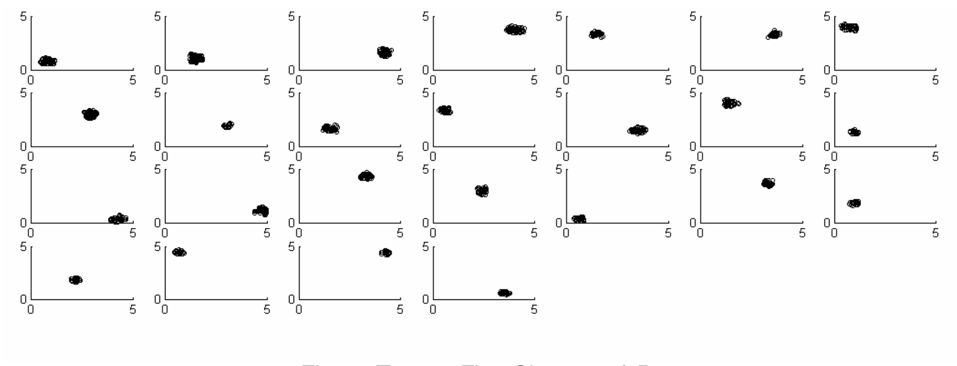


Fig. 6- Twenty Five Clusters of B_3

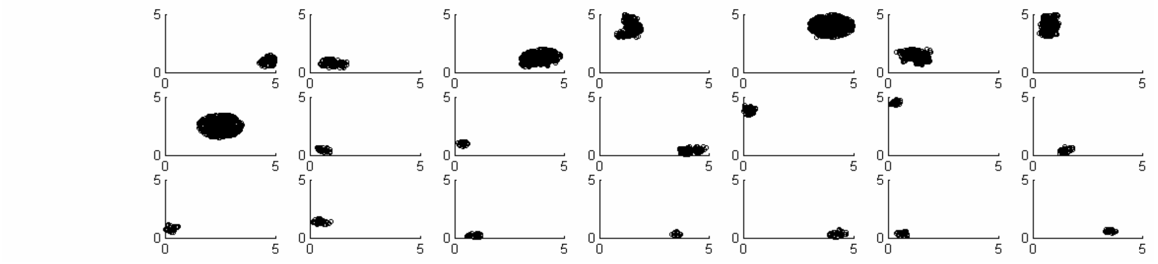


Fig. 7- Twenty One Clusters at the end of B_3 after internal merging

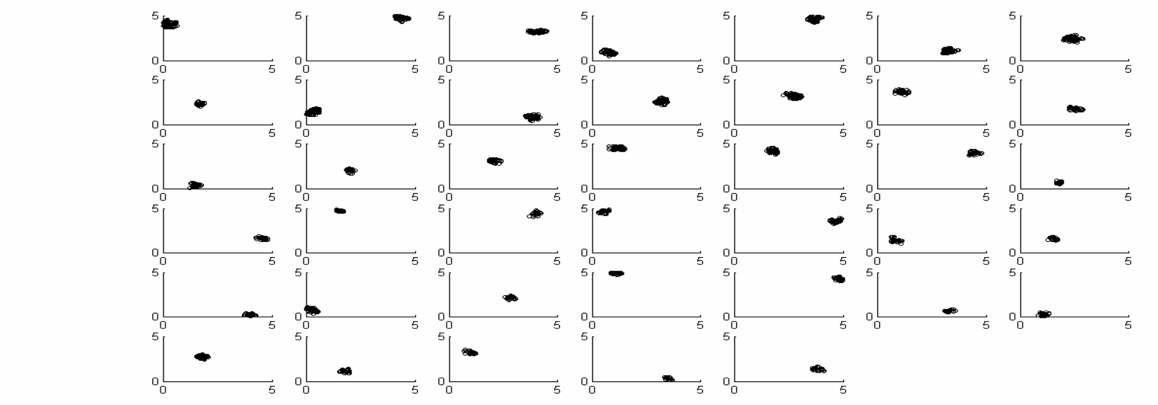


Fig. 8- Forty Clusters of B_4

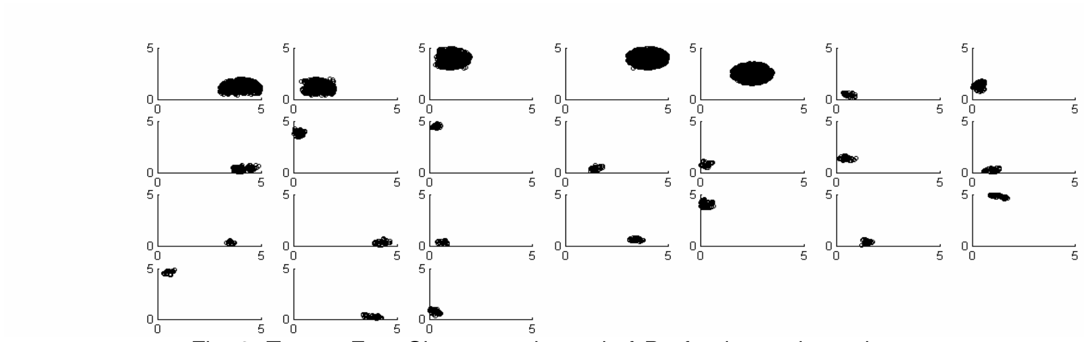


Fig. 9- Twenty Four Clusters at the end of B_4 after internal merging

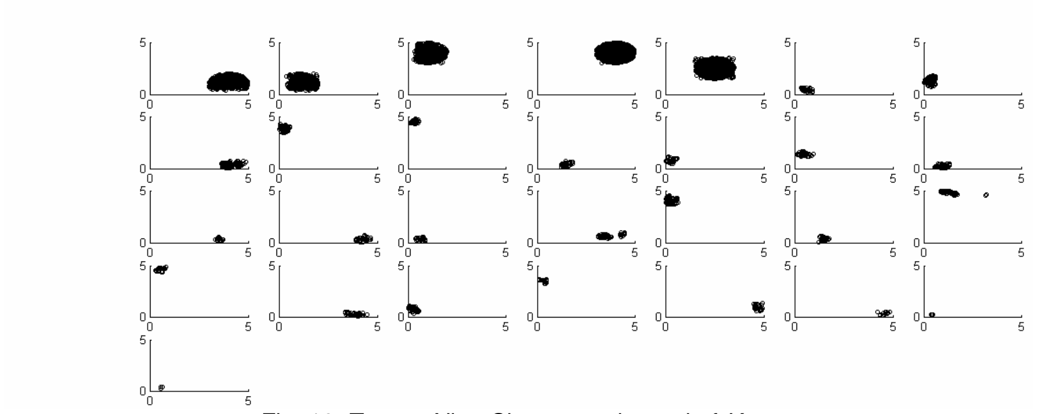


Fig. 10- Twenty Nine Clusters at the end of K_4

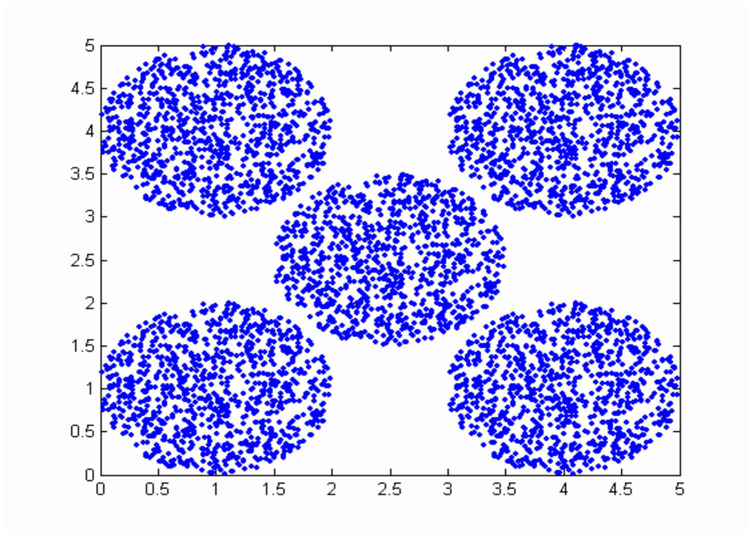


Fig. 11- Final Five Knowledge Packets/Clusters

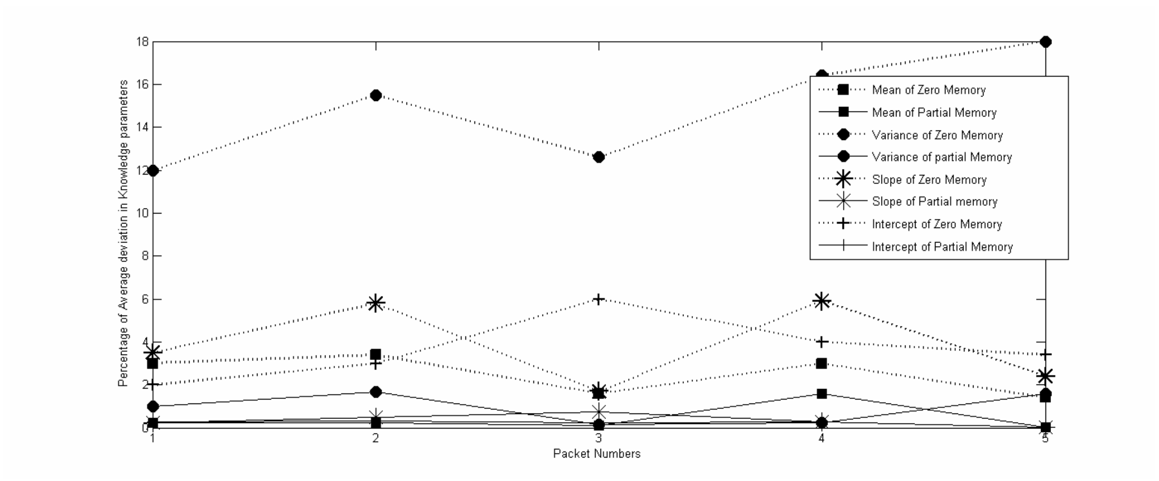


Fig. 12- Percentage of average deviation of knowledge parameters in zero and partial

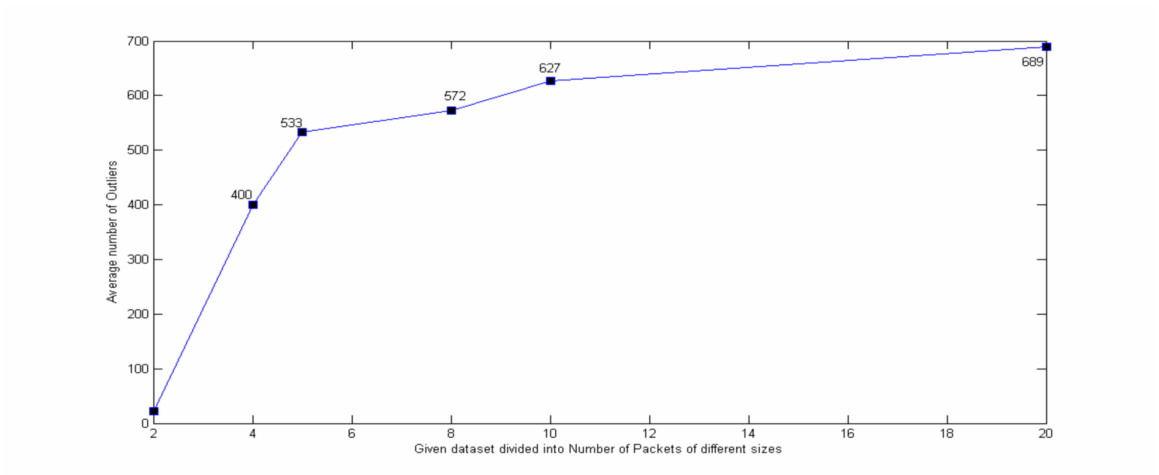


Fig. 13- Average samples maintained in partial instance memory when the given dataset of 5000 samples is divided into 4, 5, 8, 10 and 20 batches of different sizes.

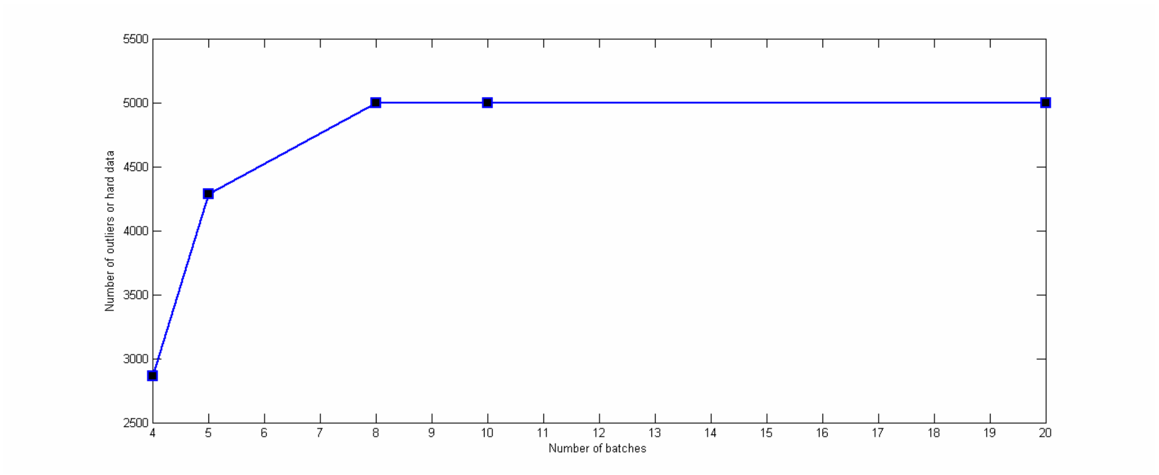


Fig. 14- Number of *hard samples* in partial instance memory when the initial dataset of 5000 samples is divided into batches of different sizes

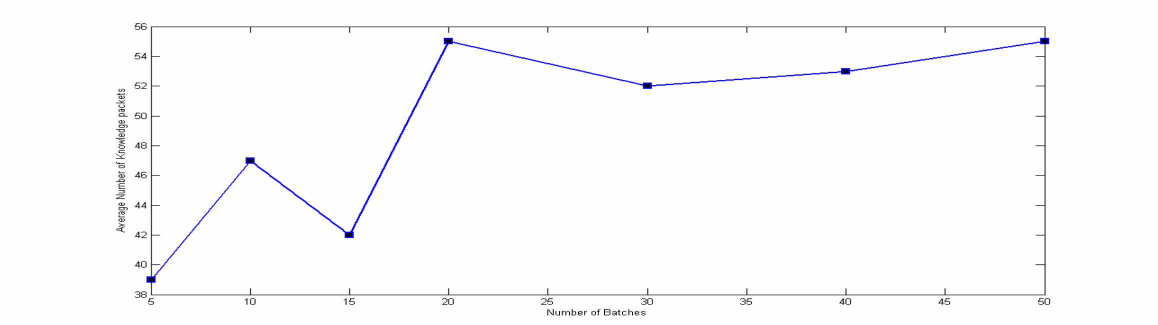


Fig. 15- Average knowledge packets retained in concept memory in partial instance memory learning when the number of batches increased from 5 to 50 in different steps.

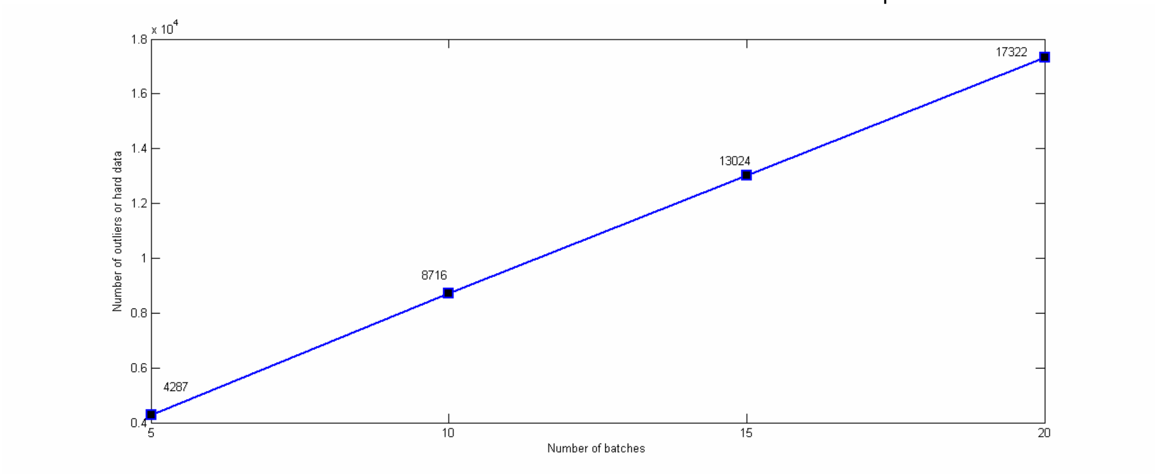


Fig. 16- Number of accumulated *hard samples* in partial instance memory learning when the batch size is fixed at 1000 and the number of batches increased from 5 to 20 in different steps.

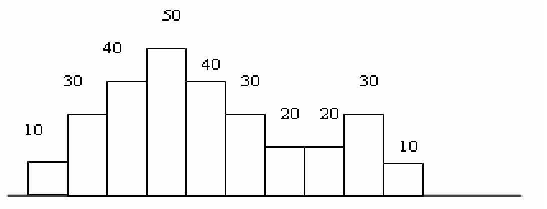


Fig. 17- A Sample Histogram

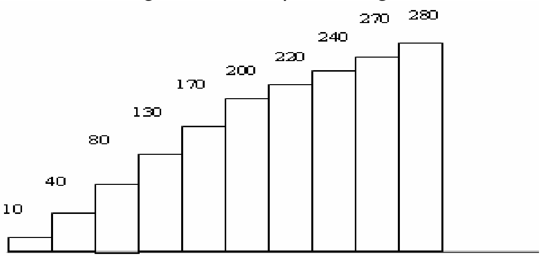


Fig. 18- Cumulative Histogram for the histogram of Fig 17.

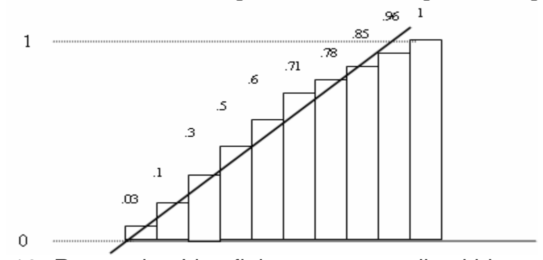
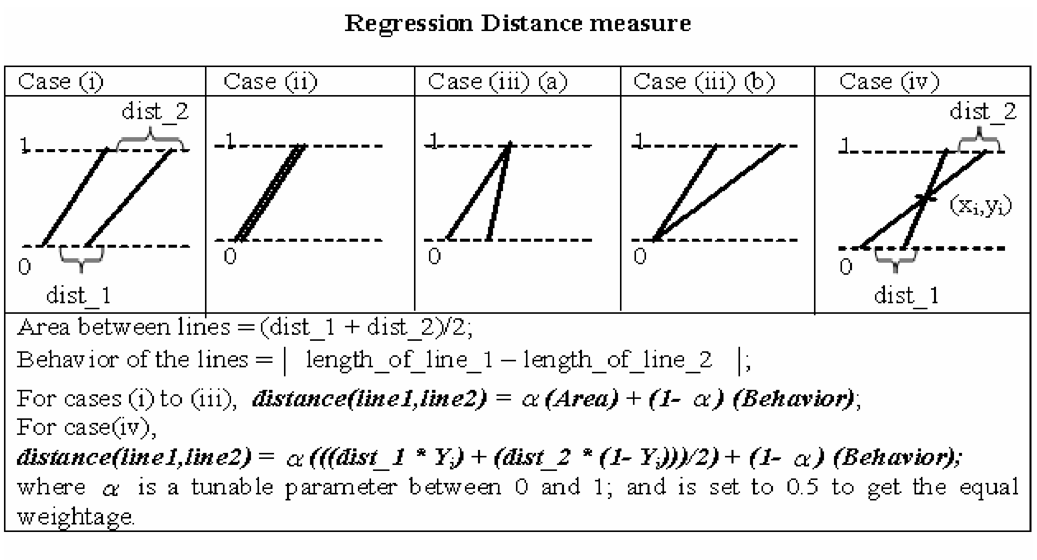


Fig. 19- Regression Line fitting on a normalized histogram



Area between lines = (dist_1 + dist_2)/2;

Behavior of the lines = | length_of_line_1 − length_of_line_2 |;

For cases (i) to (iii), ***distance(line1,line2)*** = α ***(Area)*** + ***(1-*** α***) (Behavior)***;

For case(iv),

distance(line1,line2) = α ***(((dist_1 * Y_i) + (dist_2 * (1- Y_i)))/2) + (1-*** α***) (Behavior);***

where α is a tunable parameter between 0 and 1; and is set to 0.5 to get the equal weightage.

Fig. 20- Distance between two lines

Vitae

P Nagabhushan (BE-1980, M.Tech–1983, PhD-1989) is a full professor at the department of studies in Computer Science and is Director – Planning Monitoring and Evaluation Board at the University of Mysore, India. He is an active researcher in the areas pertaining to Pattern Recognition, Document Image Processing, Symbolic Data Analysis and Data Mining. Till now he has successfully supervised 18 PhD candidates. He has over 400 publications in journals and conferences of International repute. He has chaired several international conferences. He is a visiting professor to USA, Japan and France. He is a fellow of Institution of Engineers and Institution of Telecommunication and Electronics Engineers, India.

Syed Zakir Ali (1970) is a research scholar at the department of studies in Computer Science, Manasagangothri, Mysore. He completed his Bachelor's degree in Computer Science and Engineering in 1993 from Vijayanagar Engineering College, Bellary, affiliated to Gulbarga University; and Master's degree in Computer Engineering in 1996 from Sri Jayachamarajendra College of Engineering (SJCE), affiliated to University of Mysore, India. He is an academician and a budding researcher who has taught graduate level courses at International levels for the past 12 years. His areas of interest include Data Mining, Knowledge Management, and Artificial Intelligence. He is a member of IEEE and ISTE.

Pradeep Kumar R (1977), has completed his bachelors degree in Electrical Engineering in 1999, Master's degree in Computer Engineering in 2001 and PhD in Computer Science from University of Mysore, India in 2006. He has been an active researcher and an academician for the past 8 years. His areas of interest include Data and Knowledge Engineering, Image and video processing and Computational Intelligence. He is a professional member of ACM