Volume: 02, Issue: 02, Pages:581-596 (2010)

Intelligent Methods of Fusing the Knowledge During Incremental Learning via Clustering In A Distributed Environment

Dr. P. Nagabhushan
(On lien from the Department of Studies in Computer Science, University of Mysore, Mysore – 570 006)
Presently with Bangalore Technological Institute (BTI) and Bangalore Educational Society for Technology Advancement and Research (BESTAR), Bengaluru – 560 035
Email: pnagabhushan@hotmail.com
Syed Zakir Ali
Department of Studies in Computer Science, University of Mysore, Mysore -570006
Email: zakirsab@gmail.com
Pradeep Kumar R
Amphisoft Technologies Private Limited, Coimbatore
Email: pradeep@amphisoft.com

ABSTRACT
One of the ways of learning from the data which is physically distributed over multiple locations is to have a common learning mechanism at each of the source and knowledge of each of the learnt concepts has to be transmitted to a centralized location for assimilation. In this research, clustering is employed as a mechanism of learning and a cluster is viewed as a concept which is described by a set of variables. The set of variables which describes each of the clusters is being referred to as a *knowledge packet (KP)*. As histograms have the generic ability to characterize any type of data, a histogram based regression line has been used as one of the variable to describe a *KP*. For online monitoring of the progression in learning apart from achieving computational ease and efficacy, the *KP*s at the centralized location are fused incrementally to get the overall knowledge. If learning mechanisms employed are data sequence sensitive, different combinations of merging the thus generated *KP*s may result in altogether a different overall knowledge. Further, the distance measure employed to find distance between the *KP*s in obtaining the optimal sequence of merging, may also result in a different overall knowledge. This phenomenon is being referred to as the problem of *order effect*. To minimize or avoid the *order effect*, a density based spatial clustering of applications with noise (DBSCAN) algorithm, which is insensitive to the order of presentation of data samples is used to learn from the data chunks and a novel methodology of finding the distance between the batches of data and there by finding the more optimal sequence of merging the *KP*s is presented. A specially designed distance measure for histogram based objects (histo-objects) is employed to find distance between the *KP*s and the nearest *KP*s are merged incrementally till certain conditions are satisfied. The proposed methods provide a robust mechanism of avoiding order effects. Since it is difficult to get the real distributed datasets, effectiveness of the proposed approaches is demonstrated with a carefully designed synthetic dataset. Some of the bench mark datasets were modified to simulate the distributed environment and experimentations with some of them show an accuracy of up to 100%.

Keywords - Cluster analysis, Incremental augmentation of knowledge, Order effect, Regression Analysis.

Date of Submission: July 16, 2010 Date of Acceptance: August 14, 2010

I. Introduction

When there is requirement of learning from a huge mass of data which cannot be processed in one go, the best option is to divide the data into smaller manageable chunks and learn from each of the chunks in an incremental way i.e., the knowledge generated by learning from a chunk of data is updated after learning from each of the subsequent chunks. The process of learning in which the knowledge is updated or derived in a phased manner without recalling the original data corresponding to the knowledge already obtained is referred to as Incremental Learning [1-2]. The situation of learning could be identical in a distributed environment where the data to be processed has to be pooled in from different sources.

With the advancement in communication technology, the dream of learning from the data which is physically distributed over different locations has become a reality. However considering the fact of the large size of data, gathering all the data at a centralized location will be an over burden on the communication channel and also demands more resources at the centralized location to process the huge mass of data. Moreover, the privacy and security constraints also restrict the transfer of raw data from a source to an intermediary or a centralized location. For example, in the application of counting of electoral votes, the ballot boxes are retained at physically distributed locations and learning from the data available in ballot boxes is done at their respective source locations; after learning, the obtained knowledge from the learnt concepts is transmitted to the centralized location. Thus it is sensible to process the data at source and use communication channel to transmit the knowledge. For the compatibility of knowledge resulting from different sources of data, it is advisable to have identical learning mechanisms at each of the multiple locations.

While learning incrementally, if the learning mechanisms employed are data sequence sensitive, different ways of merging the generated knowledge will lead to different results and literature has termed it as the problem of *order effect* [1]. The other aspect of the order effect is that the knowledge packets generated at the intermittent stages instead of showing the trend would deviate at earlier levels. The conventional order effect problem is overcome by the merging policy adopted. It is pointed out in [1] that there exists at least three levels at which the order effects can occur - (i) at the level of *attributes* (ii) at the level of *instances* and (iii) at the level of *concepts*.

During incremental learning with multi-dimensional data, if all the dimensions or features of the data are not available, then learning has to begin with the available features and should be reviewed as and when the new features become available (of course without reusing the already processed features). Different orders of processing of the features will lead to different results. This phenomenon is referred to as order effect at the level of *attributes*. Learning with the available samples or rows of the data (all attributes of each row), will lead to order effect at the level of *instances*. Similarly, when learning has to be done with the available concepts, which are the statistical summaries of the data chunks, then different orders of processing the concepts will lead to order effect at the level of *concepts*. This is the one which is considered in this paper.

The learning mechanism employed in this research is *clustering*, which is an unsupervised classification method that aims to partition the dataset into subsets called clusters such that the degree of similarity is high among members of the same cluster and low between the members belonging to different clusters. Each cluster is viewed as a concept. A *concept* is considered as knowledge which is described by a set of variables and is being referred to as a *knowledge packet* (*KP*) in this paper.

Talavera et al [3] made an attempt to avoid ordering effect at the level of instances. Here the task is to classify the instances into various classes. The learning mechanism employed is clustering. To avoid order effects, whenever an instance cannot be added to a cluster, it is stored in a buffer with a hope of getting utilized in future progression. When the size of the buffer reaches a user defined threshold, the instances present in the buffer are reprocessed. By reprocessing the data, though the system no longer remains fully incremental, we can still say that only a limited data has been reprocessed for the sake of avoiding order effects. The process of buffering is similar to the idea of *partial instance memory* proposed by Michalski et al [4]. A similar strategy to learn from temporal data has been carried out by us in [5].It has been shown that the optimal size of the buffer cannot exceed 10% of the dataset. In another study [6], we have also shown that a fairly good learning is possible even by discarding the buffered data, which leads to the idea of *zero instance memory learning* [4].

Nicola et al [7] have also made an attempt to avoid *order effects* at the level of *instances* where the task is to learn a concept definition from instances. To avoid *order effect*, backtracking mechanism has been employed to go back to a previous knowledge level whenever a dead end is reached. Going back to a previous knowledge level implies maintaining more than one set of knowledge which violates the basic principles of incremental learning [1].

Fisher in [8], made an attempt to avoid order effects at the level of *concepts* using a classification tree (CT). Each node in the CT is a probabilistic concept that represents an object of a class. It is not mentioned explicitly as to how these concepts were learnt. As and when a new concept becomes available, it is placed into each of the existing nodes of the CT apart from creating a new node to place the incoming concept. Then an evaluation is performed to find a best partition. The node that results in the best partition is allowed to host the new concept. As pointed out by the authors, this process is very sensitive to the ordering of the initial input. Hence, whenever a concept is inserted into a CT, the node which has accepted the new concept is checked for merging and splitting. It is observed that since merging and splitting has inverse relation, the order effects could be eliminated automatically.

In [8], rather than placing the new concept in each of the existing nodes of concepts and checking each of the concepts for a best partition, had the similarity measure between the new concept and the existing concepts is obtained and the new concept is placed into the concept with nearest distance, the computational efforts could have got reduced. By doing so, even when more than one concept is available for insertion, we can easily find the concept to be inserted first and there by the ordering effects can be avoided. Further in [8], since it is unclear as to how the concepts are being represented, we may have to explore ways of standardizing the representation of concepts so that the distance between concepts can be computed easily.

In [9], a framework for learning from the data which is distributed over multiple sites has been proposed. It has been wisely advocated to extract the knowledge of the data at the respective sites and transmit the knowledge to a centralized location for further processing. In this case, knowledge from different sources is gathered at a centralized location and is *some how* fused rather than fusing them in a systematic way. For online monitoring of the progression of learning apart from achieving the computational ease and efficacy, the knowledge at the centralized location should have been be fused incrementally to get the overall knowledge. Since different orders of fusing the knowledge become possible, there is a strong need to engineer the fusion process which should avoid order effects. *Since incremental learners tend to favor the local optimum solution compared to global one, there should be some mechanism to insert global knowledge at each step of learning into the incremental learners.*

In [10], an attempt is made to uncover the reasons behind *order effects* in a supervised environment. It is shown that *optimality* and *storage criteria* are sufficient for ensuring order independence. It has been pointed out that achieving order independence in *unsupervised learning* scenario would be interesting and represents a choice for future work.

The work of Christoppe [2] lists many problems associated with incremental learning. Of the several problems, achieving order independence is of greater relevance, but has remained largely open [1-2], [10-12]. For example, to quote from Langley [1], *the field of incremental learning needs better measures for detecting order effects in incremental learners*. Further, it is pointed out in [1] that while learning incrementally, different samples or attributes or concepts may lead the learner down quite different paths, and later experiences may not be sufficient to counteract them. Hence there is a strong need to find ways to avoid *order effects* in incremental learning.

Though the work in [7] concentrates on avoiding order effects at the level of instances, it has been pointed out that the problem of *order effects* at the level of *concepts* requires further analysis and represents a future work issue.

From the above discussions, it is clear that avoiding order effects in incremental learning especially with respect to *concepts* has not been engineered to perfection till date and needs a thorough analysis. Further, the exact definition of a *concept* is not explicitly premeditated. In this research, we concentrate on (i) defining the *concept* explicitly (ii) identifying the challenges involved in avoiding order effects at the level of *concepts* and (iii) providing a robust solution to avoid order effects - especially with respect to the *concepts* that were generated at different sites and have to be transmitted to a centralized or an intermediary location for incremental augmentation.

Since DBSCAN (Density Based Spatial Clustering of Applications with Noise) [13] algorithm is insensitive to the order of presentation of data samples (i.e., irrespective of the order of presentation of data samples the resulting clusters will not change), we would like to employ it for the purpose of learning at all the sites of a distributed environment. This overcomes one aspect of order effect. To overcome the other aspect of order effect, two ways of inserting the global knowledge into incremental learning process is proposed. These two ways differ only in the way the batches are combined and are termed as Kruskal's like and Prim's like methods. These names are in line with the methods proposed by Joseph Kruskal and Robert C. Prim for the selection of a node to be inserted during the construction of a *minimum spanning tree* (mst) from a given weighted graph [14]. The data samples which are considered as outliers during the knowledge generation process at each of the sites were considered as single element clusters of that site. For the sake of simplicity, we presume that the data available at each of the sites can be easily processed in one step without the obligation of splitting, as order effects may creep in even at this level.

Since histograms have the generic ability to characterize any type of data and since the frequency distribution of elements is conveniently recorded in histograms, the knowledge of each of the obtained clusters is represented using histograms. Further, since the memory requirement of histograms depends on the number of bins, the histograms are transformed into first order regression lines which require only two variables to store slope and intercept.

II. The Proposed Approach

In a distributed environment, as it is sensible and desirable to transmit the knowledge of each of the learnt concepts, it has become necessary to sort out a set of variables which portray a concept proficiently.

2.1 Statistical variables to portray a *concept*

We have identified the following set of variables as a sufficient requisite to describe a *concept*.

(i) *Number of elements* in each cluster. We denote it by '*n*'. This could be useful in keeping the concepts in normalized form and can reflect the density as well.

(ii) *Centroid* is the most commonly used statistical variable to describe the center of a cluster, assuming that clusters are generally spherical in structure. *Centroid* is the arithmetic mean of all the elements in that cluster. We denote the centroid by 'μ' and is given by

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{--- (1)}$$

(iii) *Standard deviation* is the measure of distribution of objects around the centroid and is very much useful in finding the density of the cluster. We denote the standard deviation by 'σ' and is given by

$$\sigma = \sqrt{\left(\frac{1}{n-1}\right)\sum_{i=1}^{n}\left(x_i - \mu\right)^2} \quad \text{---- (2)}$$

(iv) Frequency distribution of elements can be easily recorded in histograms and only histograms have the generic ability to characterize most of the symbolic and

conventional data types. Hence we have considered histogram as a better representative of a *concept*. Though we require few bins to store a histogram, the memory required to store bins is far less than the actual data itself. However, as the number of bins of histogram varies, the memory requirement will also change. To keep the memory requirement stable, we have decided to utilize the idea of transforming the histogram into a regression line proposed in [15, 16]. Such a regression line requires memory to store only two variables (slope and intercept) irrespective of the number of bins of the histogram. We normally refer histogram based regression line as a *histo-object.*

Thus, in this research a *concept* is described as a *set* of *number of elements in each cluster*; *mean*, *standard deviation* and *histogram based regression line* of each of the dimension of the cluster. As brought out earlier, we refer to this set of variables as a *knowledge packet (KP)*. For example, for an *m*-dimensional data space, if the current source or current batch $[B]_i$ shows up 'k' clusters $[Cl_1]$, $[Cl_2]$ … $[Cl_k]$, then knowledge structure of $[k]_i$ is as shown in Table I. Each row in Table I is a *knowledge packet (KP)*.

Construction of histogram requires input about the number of bins and the bin width. In this research, through out the distributed sources, the number of bins is simply fixed at 10 and the bin width is decided by the prior knowledge [4] about the dataset. Since change in number of bins and the bin width may affect the performance, we would like to take up this issue in a separate study. However, as histograms are ultimately converted to cumulative histograms before converting them to regression lines, this may not pose a threat.

For the sake of completeness, a glimpse at the conversion of histogram to regression line through an intermediate stage of constructing its cumulative histogram as proposed in [15, 16] is reviewed here.

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 bi 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 1.

A cumulative frequency distribution is computed for each of these 10 centers resulting in the cumulative histogram (CH); CH = $\{ch_1, ch_2, ch_3, ch_4, ch_5, ch_6, ch_7, ch_8, ch_9, ch_{10}\}$ where ch_i = sum (ch_k) for k = 0 to i; for the histogram of Fig 1, CH becomes {10 40 80 130 170 200 220 240 270 280} and the cumulative histogram is as shown in Fig 2.

CH is then normalized by dividing ch_i for i = 1 to 10 by ch_{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 3. Even a small change in histogram is easily reflected in the cumulative histogram which changes the slope of the line and hence there is no chance of getting same slope for two different histograms.

2.2 Distance between *concepts*

If the *concepts* are of irregular shape (for example chain like), it may so happen that the *Centroid* may not lie within the boundaries of the *concept* itself and the *concepts* with nearest *Centroids* may not be the nearest *concepts* with respect to the internal distribution of elements within the *concepts*. Hence as we have a much stronger variable in the form of a histogram based regression line (histo-object) which also records the distribution of objects within a *concept*, it is sensible to find distance between the regression lines. For this purpose, we would like to utilize a specially designed distance measure for histo-objects called *regression distance measure* [15, 16]. Since the presence of histogram based regression line nullifies the requirement of μ and σ, we have retained them for deciding the merging criteria of *concepts.*

For the sake of clarity, the regression based distance measure [15, 16] is summarized in fig 4. Upper part of the fig 4 gives the different cases of regression lines that could become possible and lower part gives the method of finding distance between the regression lines by calculating the area and behavior of the regression lines.

When a *KP* is described by multiple dimensions, each of the dimensions of a *KP* is represented by a regression line. In that case, the distance between the *KP*s is the average of the distances between each of the dimensions.

2.3 Merging of nearest *concepts*

Once we can obtain distance between two *concepts* or *KP*s, we can extend it to find distance between every pair of the available *KP*s and can represent it as a distance matrix. From the distance matrix, we can easily find the nearest *KP*s. This helps to overcome the problem of order effect.

Criteria for merging *KP*s can be obtained by checking the density (ρ) of the proposed merge. *The formula used to calculate density should indicate the distribution of objects in a cluster (which is a KP in our case) around its centroid* [17]. The density of a typical knowledge packet 'i' can be obtained by

$$\rho_i = (n_i / \sigma_i) \quad \text{---- (3)}$$

where n_i is the number of elements and σ_i is the standard deviation of i^th *KP* .

Merging of the nearest *KP*s could cause uneven distribution of knowledge requiring the splitting of *KP*(s) which requires access to the raw data. Since we have only the knowledge about the raw data which is present at different locations and since the idea is not carrying the data along, there should be some mechanism at the time of merging itself which can avoid splitting of the *KP*s at a

later stage. Hence we have applied the following criteria for merging.
If the density of the proposed merge is greater than or equal to the minimum of the existing densities of the *KP*s under consideration, then the *KP*s are merged. Otherwise, the process of merging is terminated. Further details on merging have been made available in the following sections.

III. THE NEW COMPUTATIONAL MODEL

The basic model for generation of knowledge of an ith source or ith batch $[B]_i$ is:
$[Concepts]_i + [Outliers]_i \leftarrow [DATA]_i$ ---- (4)
$[Single\ element\ Concepts]_i \leftarrow [Outliers]_i$ --- (5)
Model for obtaining the updated knowledge upon merging of knowledge packets is:
If (n_1, μ_1) is number of elements and mean of knowledge packet 1, and (n_2, μ_2) is the number of elements and mean of knowledge packet 2, then upon merging of these two knowledge packets we obtain (*updated_n, updated_μ*) as the number of elements and the mean of the updated knowledge packet and is obtained as:

$$updated\_n = n_1 + n_2 \quad \text{--- (6)}$$

$$updated\_\mu = \left(\frac{(n_1 * \mu_1) + (n_2 * \mu_2)}{n_1 + n_2} \right) \quad \text{--- (7)}$$

In the same way, updated standard deviation ($updated\_\sigma$) is obtained as follows:

$$updated\_\sigma = \sqrt{((n_1*(\sigma_1^2+d_1^2))+(n_2*(\sigma_2^2+d_2^2)))/(n_1+n_2)} \quad \text{--- (8)}$$

Where,

$$d_1^2 = (\mu_1 - updated\_\mu)^2;$$

$$d_2^2 = (\mu_2 - updated\_\mu)^2;$$

Similarly, updated density can be calculated as

$$updated\_\rho = \frac{updated\_n}{updated\_\sigma} \quad \text{--- (9)}$$

Similarly, If (n_1, s_1, i_1) is the number of elements, slope and intercept of the regression line of knowledge packet 1, and (n_2, s_2, i_2) is the number of elements, slope and intercept of the regression line of knowledge packet 2, then upon merging of these two knowledge packets we get (*updated_n, updated_s, updated_i*) as the number of elements, slope and intercept respectively of the updated knowledge packet and is obtained as follows:

$updated\_n = n_1 + n_2$; (Same as Eq. 6 above)

$$updated\_s = ((n_1 * s_1) + (n_2 * s_2)) / (n_1 + n_2) \quad \text{--- (10)}$$

$$updated\_i = ((n_1 * i_1) + (n_2 * i_2)) / (n_1 + n_2) \quad \text{--- (11)}$$

Proofs for these formulations need not be elaborated here as they are well established results in statistics [17]. It is to be noted that in our earlier work [5, 6] on incremental learning in the context of data arriving temporally, we have used a similar model to update the existing knowledge with the knowledge derived from the new chunk of incoming data. However, with the data arriving temporally, one shall not be pressed with *order effect.*

Basic model to check the feasibility of merging of the nearest two knowledge packets:

Let ρ_1 and ρ_2 are the densities of the knowledge packets to be merged and $updated\_\rho$ is the density of the proposed updated knowledge packet, which can be computed as per eq. (9). If the $updated\_\rho$ is greater than or equal to $\min(\rho_1, \rho_2)$ then merging of knowledge packets is allowed.

It should be observed that, a suitable strategy is presented for merging the *KPs*, but not for splitting a *KP* for two reasons. (i) Since merging operation is prevented under the unfavorable conditions the question of subsequent splitting does not arise and (ii) splitting of *KP* requires access to the raw data which has been assumed to be inaccessible. The alternative for access to raw data could be re-constructing the data with the available knowledge, which is discouraged due to computational over burden [10].
The simplest way to avoid order effects is to find the distance between all the available *KP*s and merge the nearest *KP*s. Upon every merge, the distance between the remaining *KP*s has to be re-computed. Undoubtedly, the number of *KP*s are very much less than the actual data itself. However, if we could think of a heuristic which can reduce the complexity of constructing the distance matrix, it saves lot of space and time. Here we present a possible heuristic which significantly reduces the resources required for computation.

Model for generating the knowledge of a source or a batch:

Summation of all the *KP*s of a *source* or a *batch* is referred to as *batch knowledge*.
Number of elements that have contributed to the knowledge of a batch is given by $\sum_{i=1}^{j} n_i$, where *j* is number of *KP*s and n_i is the number of elements of the *KP* '*i*'. μ and σ can be calculated by repeatedly executing the eq. (7) and eq. (8) till all the *KP*s of the batch are processed.
Given two histograms H1 and H2 defined over a same variable '*i*' with '*N*' number of bins, the basic addition operation can be defined as:

$$H1 + H2 = \sum_{i=1}^{N} \left(H1(i) + H2(i) \right) \quad \text{--- (12)}$$

The above model of addition of two histograms can be extended for addition of all histograms of a batch. The resulting histogram can be transformed into a regression line as explained above to generate the *batch knowledge.*
Alternatively, regression line of each batch can also be constructed by repeated addition of the regression lines of each of the *KP*s of that batch using eq. (10) and eq.(11). To speed up the process, knowledge generation of each batch could be done in parallel. Then the distance between

the batch knowledge of each of the batches could be obtained as we obtained the distance between *KP*s. Then the nearest two batches to be selected for merging. This reduces the search space significantly.

Alternatively, we can obtain the *batch-knowledge* from all the elements of a batch before applying the learning mechanism at the source itself and send the *batch-knowledge* along with the *KP*s of the batch.

Basic model for merging of knowledge packets of the nearest two batches:

Let us assume that out of the available batches of data, the nearest two batches are B_i and B_j and the number of knowledge packets (k) in each batch could be different.

$$B_i = \{k_{i1}, k_{i2}, k_{i3} ... k_{ip}\} \quad \text{--- (13)}$$

$$B_j = \{k_{j1}, k_{j2}, k_{j3} ... k_{jq}\} \quad \text{--- (14)}$$

Let us assume that the knowledge packets of B_i are to be moved to B_j during the process of merging.

Let us assume that k_{i1} is nearest to k_{j3} and merging is agreed, then k_{j3} grows because of the merger of k_{i1} in it and its distance with the other packets within B_j could get reduced inducing further merging with B_j. The same process could happen in B_j also and the number of packets gets reduced after every merging. If merging is not agreed, k_{i1} is moved to B_j as k_{jq+1}. In either case, distance matrix of B_i and B_j has to be dynamically updated. Once all the packets of B_i are moved to B_j, number of batches gets reduced and the distance between the batches to be recomputed to find the next nearest two batches.

IV. ALGORITHM

A simple algorithmic form of the main procedure of kruskal's like method [14] of merging the *KP*s to avoid order effects is provided in the following section.

Algorithm: Avoiding concept level order effects during incremental learning through clustering in a distributed environment

```
1) Apply DBSCAN [13] clustering algorithm on the
   data available at each source. Extract knowledge
   of each of the resulting clusters, which is in the
   form of KPs and send it to a third party location.
   Also send the knowledge of the entire source
   which can be termed as batch knowledge.
2) At the third party location,
while (no-of-batches > 1)
    Find distance_between_batches using the batch-
    knowledge and pick the nearest batches;
    Among the nearest batches, identify one of the
    batch as source_batch and the
    other as destination_batch;
    while (no-of-KPs of the destination-batch > 0)
        Find distance between source-batch KPs and
        destionation-batch KPs
        Identify the nearest KPs;
        if (the nearest KPs can be merged as per the
          specified condition)
            Merge;
            Check for induced merging;
            while (no-of-KPs of the source-batch > 1)
                Find distance between the grown-up KP
                and the existing KPs of the source_batch;
                if (induced merging is possible as per the
                    specified condition),
                    Merge;
                else
                    break; /* go out of the present while loop */
                end  /* end of if-else */
            end  /* end of while loop */
        else
            break; /* go out of the while loop */
        end  /* end of if-else */
    end  /* end of while loop */
    Update the knowledge of source-batch;
    Delete the knowledge of destination-batch;
    Reduce the no-of-batches by one;
end /* end of outermost while loop */
3) stop
```

The other method differs only in the way the sources or batches are merged. After merging the nearest two batches to get a combined batch, distance between the combined batch and the remaining batches is computed. The batch which is nearest to the combined batch is merged next. As brought out in section 1, this method of merging can be seen as *Prim's* like and the earlier one as *Kruskal's* like methods [14].

4.1 Complexity analysis

(i) Time complexity of DBSCAN clustering mechanism is O($n$log$n$) [20] [21] where 'n' is the number of samples in each chunk; If there are 'k' number of sources, then the time complexity becomes k * ($n$log$n$);

(ii) Extracting knowledge of each cluster:
If there are m clusters in a source and if there are d dimensions of each cluster, then for extracting knowledge of each source, 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 (*KP*s):
  (a) If there are m_1 *KP*s obtained from one batch and if m_2 *KP*s are obtained from another batch, and if m_1 is almost equal to m_2 time required to find distance between them is $m_1 (m_1 - 1)/2$;
  (b) Re-computation of the distance matrix has to be done with a merge of each *KP*. At each step a *KP* gets reduced by one.

In the worst case, this procedure has to be repeated $m_1 (m_1-1)/2$ times.

The Overall complexity is:
$[k * (n\log n)] + [k * (md)] + [m_1 (m_1-1)/2] * [m_1 (m_1-1)/2]$;
$\rightarrow$ O $(m^4 n)$;
It should be noted that m is the number of knowledge packets and since $m \ll n$, we can argue that m is always bound by a constant. In practical scenarios it is also true that one could be interested in designing predefined number of *KP*s only. Thus the above expression may be simplified to O (λn) where λ is in terms of m.

4.2 An illustrative example

A synthetic dataset in a 2D space has been used for the sake of clear understanding of the proposed method although the method can be applied to any finite number of dimensions as can be seen in the experimentation part in section V.
The synthetic dataset is of 20,000 points in a 2D space distributed over five classes of unit radius and is as shown in Fig 5. A similar dataset has been used by Liaden O'Collagahan et al [22] to simulate a stream of data for the study of a clustering algorithm.
The initial dataset of 20,000 points is divided into ten batches by a random selection of samples from the initial dataset. Each batch of the thus obtained batches of data is assumed to be available at a different source or different location.
The DBSCAN parameters *Eps* and *Minpts* were set to 0.15 and 10 respectively. These values were obtained through trial and error mechanism. Upon executing the DBSCAN procedure on the batches of data at their respective locations, we obtained few clusters and outliers. As brought out earlier in section 1, the outliers of each stage are considered as single element clusters of that stage. The total number of clusters of each stage is as shown in the second column of Table II. The extracted knowledge (*batch-knowledge*) of each of the complete batches of data is as shown in the subsequent columns of Table II.
Distance between the *KP*s of these batches is obtained using regression based distance measure [15, 16]. For clear understanding of the distances between each of the batches of data, the obtained distances are presented in the form of a matrix in Table III.
Based on the distance matrix the nearest batches (here $\boldsymbol{B_5}$ and $\boldsymbol{B_6}$) are selected for merging, there by reducing the requirement of huge computational resources which was required had we considered finding distance between all the *KP*s (here 237 in total) available at the centralized location.
As can be seen from Table III, $\boldsymbol{B_5}$ and $\boldsymbol{B_6}$ are the two nearest batches. After merging of *KP*s between $\boldsymbol{B_5}$ and $\boldsymbol{B_6}$, some of the knowledge packets of $\boldsymbol{B_6}$ have got merged with the *KP*s of $\boldsymbol{B_5}$ and the remaining *KP*s have remained as additional *KP*s of $\boldsymbol{B_5}$ making $\boldsymbol{B_6}$ empty. As shown in Table III, $\boldsymbol{B_5}$ has 22 *KP*s and $\boldsymbol{B_6}$ has 14 *KP*s. Had none of the *KP*s of $\boldsymbol{B_6}$ got merged with the *KP*s of $\boldsymbol{B_5}$, it would have resulted in 36 (22+14) *KP*s in the source batch $\boldsymbol{B_5}$. The merging and induced merging brought down the number of *KP*s to 29 in $\boldsymbol{B_5}$.The overall procedure reduces the number of batches by one and the distance matrix is recomputed to find the next nearest batches. Table IV gives the sequence of merging the batches and the number of knowledge packets at each stage.
In Table IV, first column of each stage indicates the number of *KP*s of a batch prior to merging and second column indicates number of *KP*s of a batch after merging. Highlighted numerals indicate the *KP*s considered for merging at that stage. Highlighted and italicized numerals indicate the updated *KP*s of that batch because of merge. Highlighted, italicized and struck-off numerals indicate the deletion of *KP*s of the merged batch. After each merge (i) knowledge of the grownup batch is updated and the knowledge with respect to the merged batch is deleted and (ii) distance between batches is re-computed and the next nearest batches are selected for merging. This method of merging can be termed as *Kruskal*'s like method of merging. Instead of re-computing the distance matrix between batches, if the distance between the grownup batch and remaining batches is obtained, and if the nearest batch is merged with the grownup batch, the resulting sequence of merging can be viewed as *Prim*'s like merging. Table V gives the sequence of merging of batches in *Prim*'s like method. The description of Table V is similar to that of Table IV.
Since the algorithm employed for learning is insensitive to the order of presentation of data samples, we could not find any difference in the results obtained by Kruskal's like and Prim's like methods of merging. However, one can see the order effect through optimality at each stage, as it has been pointed out in [10], that *optimality* and *storage criteria* are sufficient for ensuring order independence. In the following paragraph, we can examine these issues from the results shown in Tables IV and V.
If m and n are the number of *KP*s of two batches to be merged, an ineffective combination of two batches is the one in which the resulting number of *KP*s is equal to $(m+n)$. A good combination should show the number of *KP*s to be less than $(m+n)$. A very good combination should be equal to $smaller(m,n)$; further it should be less than the $smaller(m,n)$. To measure the optimality, the percentage of reduction at each stage is calculated. For a comparative analysis with the proposed methods of merging the *batch-knowledge*, we have merged the batches at random and the percentage of reduction is shown in Fig 6 and Fig 7.
It is clear from the graph shown in Fig 6 that the percentage of reduction in the *KP*s in the initial stages of *Kruskal*'s like merging is more when compared to random merging. For random merging, percentage of reduction is higher at the last stage. Further it is also clear that the stability in reduction has reached at a stage prior to the last stage in *Kruskal*'s like merging. In fact, we would have obtained the stability at a much earlier stage; however because of the constraint of avoiding over merging, we

could see the stability at a later stage. Obtaining stability in earlier stages is quite significant in this case as we can terminate the process of learning and can save lot of time required to process the remaining batches.

A Similar comparison between *Prim*'s like method of merging and the random merging is depicted in the graph shown in Fig 7. The graph shown in Fig 8, shows the comparison between *Kruskal*'s like and *Prim*'s like methods of merging.

The advantages of the above methods of merging are (i) the computational resources required for merging the nearest batches is very less when compared with processing the complete set of *KP*s of all the batches at once (ii) reduction of KPs at the very beginning implies reduction in search space and the reduction of time required in the subsequent stages of merging (iii) process of assimilation of knowledge packets can be terminated once we notice that there is no change in the knowledge by the addition of subsequent stages of merging (iv) prediction of final knowledge could become possible by analyzing the trend at each stage of merging (vi) if any one is interested in the knowledge of the intermediate stages, most accurate intermediate knowledge can be obtained.

We are exploring further reduction of computational time by employing parallel processing.

Finally repeated merging of the nearest *KP*s gave five *KP*s. The actual values of each of the five initial *KP*s and the incrementally generated *KP*s and their difference in percentage are as shown in Table VI.

It is also to be noted that since there is no difference in actual and the obtained values of the parameter *n* of each of the *KP*s, it has been omitted from Table VI.

V. EXPERIMENTATION

We have conducted experiments on two bench mark datasets – (i) *700X3* dataset [14] which is a dataset of three dimensions with 700 samples distributed over five classes and (ii) Iris [23] which is a four dimensional dataset of 150 samples distributed over three classes.

5.1 *700X3* dataset [14]

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 features 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 [14]. As done in section 4, for the sake of simulating the distributed data, we have divided the original dataset into ten batches of 70 samples each by a random selection of samples and presumed that each batch of data is available at a different location. Knowledge of each batch of data is extracted to obtain the *batch-knowledge* of the respective batches. Learning is performed on each batch of data by setting the DBSCAN [13] parameters *MinPts* and Eps to 2 and 6 respectively. The number of clusters obtained by learning from each batch of data is shown in Table VII.

Knowledge of the resulting clusters is extracted to obtain the corresponding *KP*s. Distance between the *batch-knowledge* is obtained to find the nearest batches. The batches were merged as per the proposed *Kruskal*'s like and *Prim*'s like methods of merging. For a comparative analysis, batches were also merged randomly. The obtained comparative percentage of reduction at each stage is shown in Fig 9 to Fig 11.

From the graphs, it is very clear that the process of learning could be terminated after stage 6 in both *Prim*'s like as well as in *Kruskal*'s like merging; where as in random merging stability has reached only at stage 9.

5.2 Iris data [23]

The standard iris dataset [23] has150 points in a 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.

As informed earlier, the given dataset is divided into 6 batches of 25 elements each by selecting the samples randomly. With the DBSCAN parameters *MinPts* set to 2, *Eps* set to 1.4, the obtained clusters from each batch are as shown in Table VIII.

From the graphs shown in Fig 11 to Fig 14, it is clear that the percentage of reduction in *KP*s of the proposed methods dies down with the progression of learning, making way to stop the learning process.

VI. DISCUSSIONS

In our approach, we transmit only the knowledge and hence applicable to applications which demands privacy preserving. The outlier data which we transmit as a *KP* will not reveal the details of the data and in the worst case we can claim that the privacy of such data is unwarranted.

Our approach is similar to that of Diona et al [9] which preclude the transmission of data and allows the transmission of minimal sufficient statistics. However, the minimal sufficient statistics transmitted in [9] is suitable only to the Gaussian data where as the statistics that we transmit is more robust and should be applicable any distribution.

Our approach provide novel mechanisms to sequence the process of merging the concepts arrived from different sources where as in [9] the concepts are *some how* merged without bothering about the order effects [1].

VII. CONCLUSION

As a first step in avoiding or minimizing the order effect at the level of concepts during incremental learning, we have employed a learning mechanism which is insensitive to the order of presentation of data samples. The learning mechanism employed is density based spatial clustering of applications with noise (DBSCAN) and learning is performed on the data which is physically distributed over multiple locations. We advocate employing the learning mechanism at the venue of the data and transmitting the knowledge of each of the resulting clusters to a centralized location for incremental augmentation. The knowledge of each of the resulting clusters is being referred to as *concept* or *knowledge packet* (*KP*). To strengthen the *KP*s, a set of variables including a histogram based regression line is employed in describing each of the *KP*. The data available at each of the source is considered as one batch of data and the knowledge of each of such batches of data is referred to as *batch-knowledge*. To overcome the tendency of incremental learning mechanisms in preferring the local optimum in place of global one, distance between the *batch-knowledge* of the available batches is computed and at any stage the *KP*s of the nearest two batches are allowed to participate in the incremental knowledge generation process (Kruskal's like merging). To avoid order effect during the process of merging the *KP*s of the nearest two batches, a novel regression based distance measure is employed to find the distance between the *KP*s and the nearest *KP*s are merged subject to the satisfaction of certain merging conditions to avoid over merging. As an alternative to the method of Kruskal's like merging, distance between the *batch-knowledge* of the grownup batch with the *batch-knowledge* of the remaining batches is computed and the nearest batch is merged with the grown up batch (Prim's like merging). For a clear understanding, a synthetic dataset in a 2-D space is used for the illustration of proposed methods. The proposed methods are validated using two bench mark datasets. As a future work, we would like to apply the said methods on the datasets containing classes of variable density and chain like structures, in which case we may have to find an alternative to DBSCAN. We are also exploring the possibility of applying these methods on remotely sensed images in the framework of incremental learning.

REFERENCES

1 Langley P; Order Effects in Incremental Learning, P. Reimann & H. Spada (Eds); Learning in Humans and Machines: Towards an Interdisciplinary Learning Science; (Elsevier, Amsterdam, 1995).

2 Christophe G.C, A Note on the Utility of Incremental Learning, AI Communications, 13(4), 2000, 215-223.

3 Talavera L, Roure J, A buffering strategy to avoid ordering effects in clustering, Proceedings of ECML-98, 1998, 316-321

4 Maloof A.M & Michalski R.S; Incremental Learning with partial instance memory, Artificial Intelligence 154, 2004, 95-126.

5 P Nagabhushan, Syed Zakir Ali, Pradeep Kumar R, A new cluster-histo-regression analysis for incremental learning from temporal data chunks, International Journal of Machine Intelligence 2010, 53-73

6 Syed Zakir Ali, P Nagabhushan, Pradeep Kumar R, Regression based Incremental Learning through Cluster Analysis of Temporal data, International Conference on Data Mining (DMIN) 2009, 375-381;

7 Nicola Di Mauro, Floriana Esposito, Stefano Ferilli and M.A. Basile, Avoiding Order Effects in Incremental learning, S. Badani and S. Manzoni (Eds) LNAI 3673, Springer Verlag Berlin Heidelberg 2005, 110-121

8 Fisher D.H, Knowledge acquisition via incremental conceptual clustering, Machine Learning 2 (1987), 139-172

9 Doina Caragea, Adrian Silvescu, Vasant Honavar, A Framework for learning from distributed data using sufficient statistics and its application to learning decision trees, International Journal of Hybrid Intelligent Systems 1(2) 2004, 80-89.

10 Cornuejols A, "Getting Order Independence in Incremental Learning", in the Proceedings of the Sixth European Conference in Machine Learning, 1993, 196-212,

11 Fisher D, L. Xu and N. Zard, "Ordering effects in Clustering", Proceedings of the Eighth International Conference on Machine Learning, 1992.

12 J. MacGregor, The effects of order in learning classifications by example: Heuristics for finding the optimal order, Artificial Intelligence, 34, 1988, 361-370.

13 Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu; A Density Based Algorithm for Discovering Clusters in Large Databases with Noise; Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996, 226-231.

14 Thomas H. Cormen, Charles E.Leiserson, Ronald L. Rivest and Clifford Stein; Introduction to Algorithms; (Third Edition; Eastern Economy Edition, PHI, 2009), pp 631-633;

15 Pradeep Kumar R & P Nagabhushan, Multi Resolution Knowledge Mining Using Wavelet Transform Engineering Letters, 14:1;EL_14_1_30;2007, www.**engineeringletters**.com/issues_v14/issue_1/**EL_14_1_30**.pdf

16 Pradeep Kumar R, Wavelets for Knowledge Mining in Multi Dimensional Generic Databases, PhD Thesis of the University of Mysore, India, 2006.

17 Michael J Parik, Advanced Statistics from an elementary point of view; (Elsevier Academic Press; ISBN 13:978-0-12-088494-0, 2008)

18 Mohammad Ali Kadampur, Somayajulu D.V.L.N, S.S.Shivaji Dhiraj and Shailesh G.P.Satyam, Privacy preserving clustering by cluster bulging for information sustenance, International Conference on Information and Automation foe Sustainability IEEE-ICIAFS-08, 240-246

19 Tsai-Hung Fan, Dennis K.J Lin, Kuang-Fu Cheng, Regression Analysis for massive datasets, Data and Knowledge Engineering 61 (2007) - Elsevier; 554-562;
20 Yi-Pu Wu, Jin-Jiang Guo; and Xue-Jie Zhang; A Linear DBSCAN Algorithm based on LSH; International Conference on Machine Learning and Cybernetics; Vol 5, Issue 19-22; 2007; 2608-2614;
21 Martin Ester, Hans-Peter Kriegel, Jorg Sander, Michael Wimmer, Xiaowei X; Incremental Clustering for Mining in a Data Warehousing Environment; Proceedings of the 24th VLDB conference New York, 1998, 323-333
22 Liaden O'Collagahan, Nina Mishra, Adam Meyerson, Sudipto Guha and Rajiv Motwani; Proceedings of the ICDE-2002; pp 685-694
23 S. Hettich, C.L. Blake and C.J. Merz, "UCI Repository of Machine Learning Databases," Irvine, CA: University of California, Department of Information and Computer Science, 1998

Biographies and Photographs

Prof. P Nagabhushan (BE-1980, M.Tech–1983, PhD-1989) is Principal of Bangalore Technological Institute and is a Director of BESTAR. He is presently on lien from the department of studies in Computer Science, University of Mysore, India. He is an active researcher in the 1983, PhD-1989) is Principal of Bangalore Technological Institute and is a Director of BESTAR. He is presently on lien from the department of studies in Computer Science, 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 (FIE) and Institution of Electronics and Telecommunication Engineers (FIETE) India.

Syed Zakir Ali is a research scholar at the department of studies in Computer Science. He completed his Bachelor's degree in Computer Science and Engineering in 1993; and Master's degree in Computer Engineering in 1996 from University of Mysore, India. He is an academician and a budding researcher who has taught graduate level courses at International levels for the past 14 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 Science 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 9 years. His areas of interest include Data and Knowledge Engineering, Image and video processing and Computational Intelligence. He is a professional member of ACM. He is presently managing the Amphisoft Technologies Private Limited, Coimbatore. He has served as Head of Training and R&D sections at TCS Chennai.

TABLES AND FIGURES

Table I: Knowledge parameters for Incremental learning

[B $_i$]	No. of elements	f_1			f_2			…	f_m		
[Cl$_1$]	n_1	${}^{i}\mu_1^1$	${}^{i}\sigma_1^1$	${}^{i}L_1^1$	${}^{i}\mu_1^2$	${}^{i}\sigma_1^2$	${}^{i}L_1^2$	…	${}^{i}\mu_1^m$	${}^{i}\sigma_1^m$	${}^{i}L_1^m$
[Cl$_2$]	n_2	${}^{i}\mu_2^1$	${}^{i}\sigma_2^1$	${}^{i}L_2^1$	${}^{i}\mu_2^2$	${}^{i}\sigma_2^2$	${}^{i}L_2^2$	… …	${}^{i}\mu_2^m$	${}^{i}\sigma_2^m$	${}^{i}L_2^m$
…	…	..	…	…	..	…	…	…	..	…	…
[Cl$_k$]	n_k	${}^{i}\mu_k^1$	${}^{i}\sigma_k^1$	${}^{i}L_k^1$	${}^{i}\mu_k^2$	${}^{i}\sigma_k^2$	${}^{i}L_k^2$		${}^{i}\mu_k^m$	${}^{i}\sigma_k^m$	${}^{i}L_k^m$

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

Table II: Knowledge of each batch

Batch No	***No. of clusters***	***X – (Dimension 1)***		***Y – (Dimension 2)***	
		Slope	***Intercept***	***Slope***	***Intercept***
B_1	16	0	1	0.1812	0.0994
B_2	28	0.0814	0.7220	0.1776	0.1020
B_3	28	0.1697	0.4042	0.1761	0.1094
B_4	26	0.2378	0.1333	0.2056	0.0348
B_5	22	0.2759	-0.0483	0.2652	-0.1156
B_6	14	0.2996	-0.2204	0.2852	-0.1633
B_7	25	0.2929	-0.3159	0.2298	-0.0242
B_8	32	0.2655	-0.3336	0.1851	0.0837
B_9	28	0.2112	-0.2995	0.1761	0.1094
B_{10}	18	0.1397	-0.2132	0.1798	0.0982

Table III: Distance matrix for distance between batches

	B$_1$	***B$_2$***	***B$_3$***	***B$_4$***	***B$_5$***	***B$_6$***	***B$_7$***	***B$_8$***	***B$_9$***	***B$_{10}$***
B$_1$	**0**	4997.3	5000.6	5001.7	5002.4	5002.9	5003.0	5003.1	5003.8	5005.1
B$_2$	4997.3	**0**	3.3114	4.4595	5.1805	5.6638	5.8312	5.9213	6.5307	7.8438
B$_3$	5000.6	3.3114	**0**	1.1836	1.9049	2.3880	2.5552	2.6470	3.2194	4.5682
B$_4$	5001.7	4.4595	1.1836	**0**	0.7214	1.2044	1.3717	1.7318	2.4474	3.7337
B$_5$	5002.4	5.1805	1.9049	0.7214	**0**	**0.4834**	0.9460	1.5625	2.2785	3.5647
B$_6$	5002.9	5.6638	2.3880	1.2044	**0.4834**	**0**	0.5946	1.2114	1.9273	3.2136
B$_7$	5003.0	5.8312	2.5552	1.3717	0.9460	0.5946	**0**	0.6171	1.3327	2.6191
B$_8$	5003.1	5.9213	2.6470	1.7318	1.5625	1.2114	0.6171	**0**	0.7170	2.0024
B$_9$	5003.8	6.5307	3.2194	2.4474	2.2785	1.9273	1.3327	0.7170	**0**	1.3488
B$_{10}$	5005.1	7.8438	4.5682	3.7337	3.5647	3.2136	2.6191	2.0024	1.3488	**0**

Table IV: Merging the nearest batches (Kruskal's like)

	Stage 1		*Stage 2*		*Stage 3*		*Stage 4*		*Stage 5*		*Stage 6*		*Stage 7*		*Stage 8*		*Stage 9*		*Stage 10*	
B$_1$	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	***16***	**16**	***175***	
B$_2$	28	28	28	28	28	28	28	28	28	28	28	28	28	28	**28**	***159***	~~**159**~~	**159**		
B$_3$	28	28	28	28	28	28	28	28	28	28	**28**	***139***	**139**	***146***	**146**	~~***159***~~				
B$_4$	26	26	26	26	**26**	***55***	55	55	**55**	***120***	**120**	~~***139***~~	**18**	~~***146***~~						
B$_5$	**22**	***29***	29	29	**29**	~~***55***~~	**56**	***70***	**70**	~~***120***~~	18	18								
B$_6$	**14**	~~***29***~~	**25**	***56***	56	56	**28**	~~***70***~~	18	18										
B$_7$	25	25	**32**	~~***56***~~	28	28	18	18												
B$_8$	32	32	28	28	18	18														
B$_9$	28	28	18	18																
B$_{10}$	18	18																		

Table V: Merging the nearest batches (Prim's like)

	Stage 1		*Stage 2*		*Stage 3*		*Stage 4*		*Stage 5*		*Stage 6*		*Stage 7*		*Stage 8*		*Stage 9*		*Stage 10*	
B₁	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	**16**	***175***	***175***	
B_2	28	28	28	28	28	28	28	28	28	28	28	28	28	28	**28**	***159***	**159**	~~***175***~~		
B_3	28	28	28	28	28	28	28	28	28	28	**28**	***139***	**139**	***146***	**146**	~~***159***~~				
B_4	26	26	26	26	26	26	**26**	***106***	**106**	***120***	**120**	~~***139***~~	**18**	~~***146***~~						
B_5	**22**	***29***	**29**	***49***	**49**	***80***	**80**	~~***106***~~	**28**	~~***120***~~	18	18								
B_6	**14**	~~***29***~~	**25**	~~***49***~~	**32**	~~***80***~~	28	28	18	18										
B_7	25	25	32	56	28	28	18	18												
B_8	32	32	28	28	18	18														
B_9	28	28	18	18																
B_{10}	18	18																		

Table VI: Summary of deviation of Knowledge parameters

Knowledge Parameter	**Values**	***KP_1***		***KP_2***		***KP_3***		***KP_4***		***KP_5***	
		f_1	**f_2**	**f_1**	**f_2**	**f_1**	**f_2**	**f_1**	**f_2**	**f_1**	**f_2**
μ	Actual	1.00	1.00	4.00	4.00	4.00	1.00	2.50	2.50	1.00	4.00
	Computed	0.99	1.00	3.99	4.00	3.99	1.00	2.49	2.50	0.99	4.00
	Diff (%)	1.00	0.00	0.25	0.00	0.25	0.00	0.40	0.00	1.00	0.00
σ	Actual	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
	Computed	0.50	0.50	0.50	0.50	0.49	0.50	0.49	0.50	0.49	0.50
	Diff (%)	0.00	0.00	0.00	0.00	2.00	0.00	2.00	0.00	2.00	0.00
L (slope)	Actual	0.14	0.14	0.21	0.14	0.21	0.14	0.28	0.28	0.14	0.21
	Computed	0.14	0.14	0.21	0.14	0.21	0.14	0.28	0.28	0.14	0.21
	Diff (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
L	Actual	0.49	0.49	-0.28	-0.28	-0.28	0.49	-0.16	-0.16	0.49	-0.28
(Intercept)	Computed	0.49	0.49	-0.28	-0.28	-0.28	0.49	-0.16	-0.16	0.49	-0.28
	Diff (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table VII Number of clusters obtained from each batch of 700X3 data

Batch No	***No. of clusters***
B_1	06
B_2	06
B_3	05
B_4	06
B_5	05
B_6	06
B_7	05
B_8	06
B_9	06
B_{10}	05

Table VIII Number of clusters obtained from each batch of Iris data

Batch No	***No. of clusters***
B_1	04
B_2	04
B_3	03
B_4	03
B_5	02
B_6	03

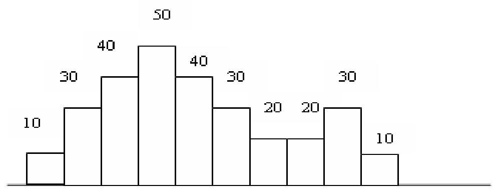


Fig. 1 A Sample Histogram

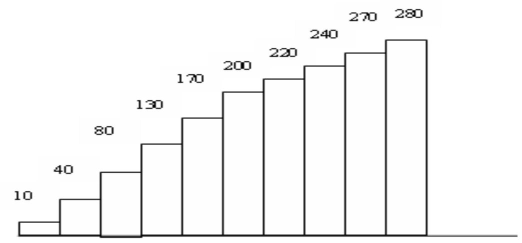


Fig. 2 Cumulative Histogram for the histogram of Fig. 1

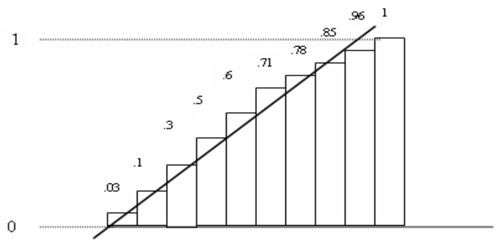
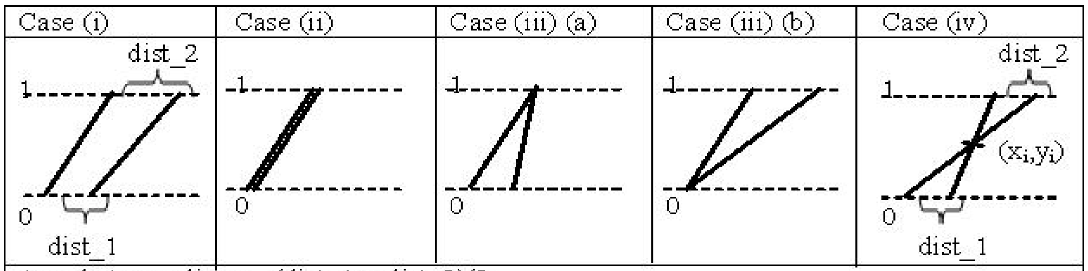


Fig. 3 Regression Line fitting on a normalized histogram



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. 4 Regression Distance Measure [15, 16]

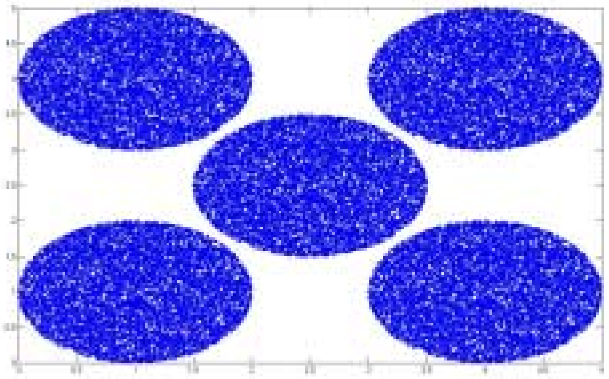


Fig. 5 A synthetic dataset of 20,000 points in a 2D space distributed over 5 classes

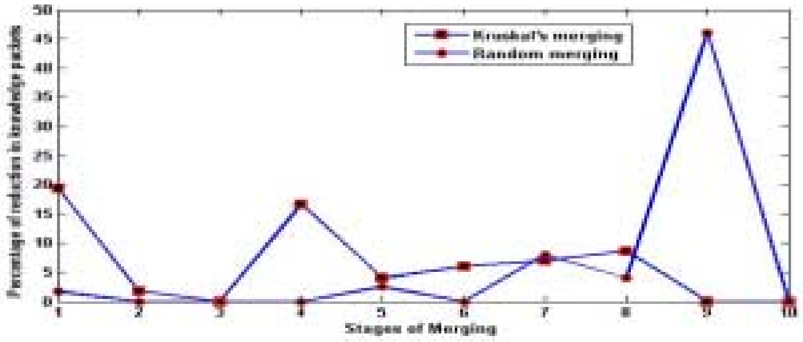


Fig. 6 Percentage of reduction in *KP*s by *Kruskal*'s like method and random method of merging

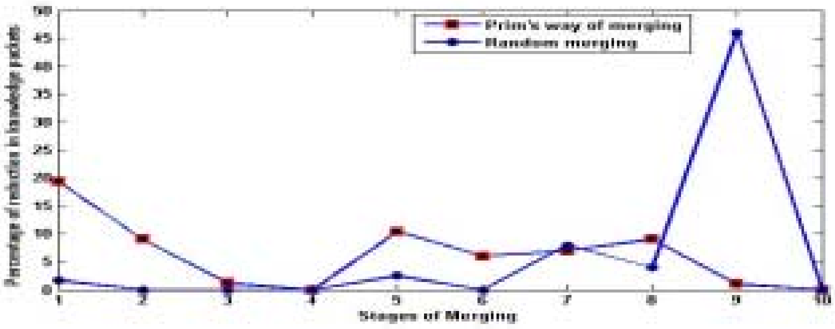


Fig. 7 Percentage of reduction in *KP*s by *Prim*'s like method and random method of merging

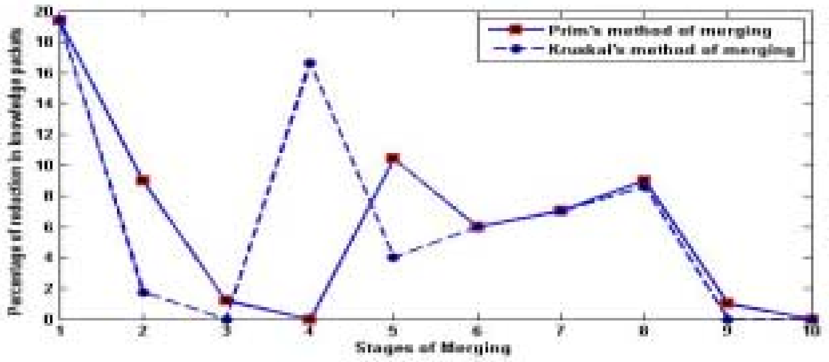


Fig 8. Percentage of reduction in *KP*s by *Prim*'s like and *Kruskal*'s like methods of merging

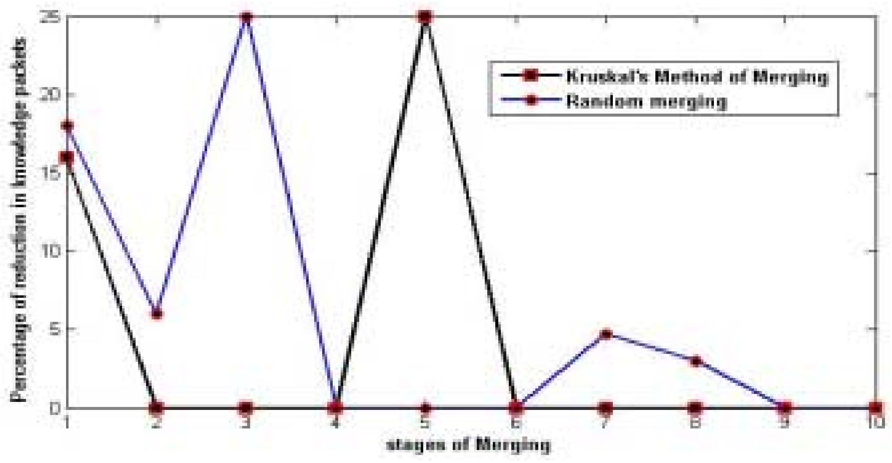


Fig. 9 700X3 dataset's percentage of reduction in *KP*s by *Kruskal*'s like method and random method

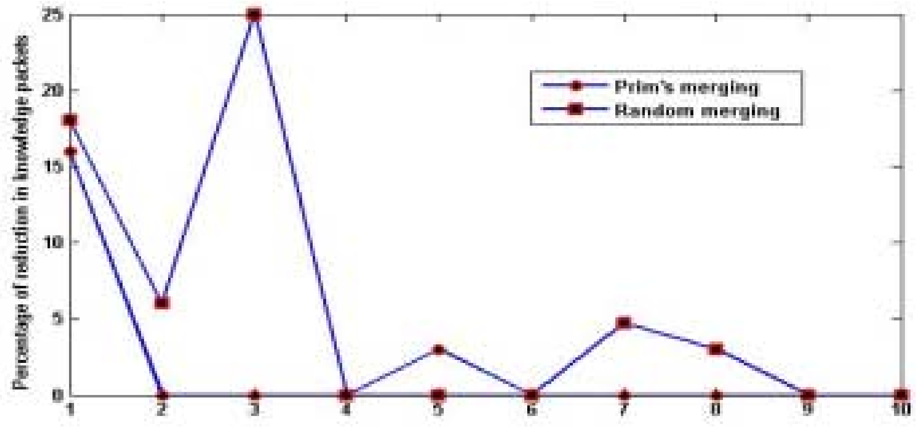


Fig. 10 700X3 dataset's percentage of reduction in *KP*s by *Prim*'s like method and random method of merging

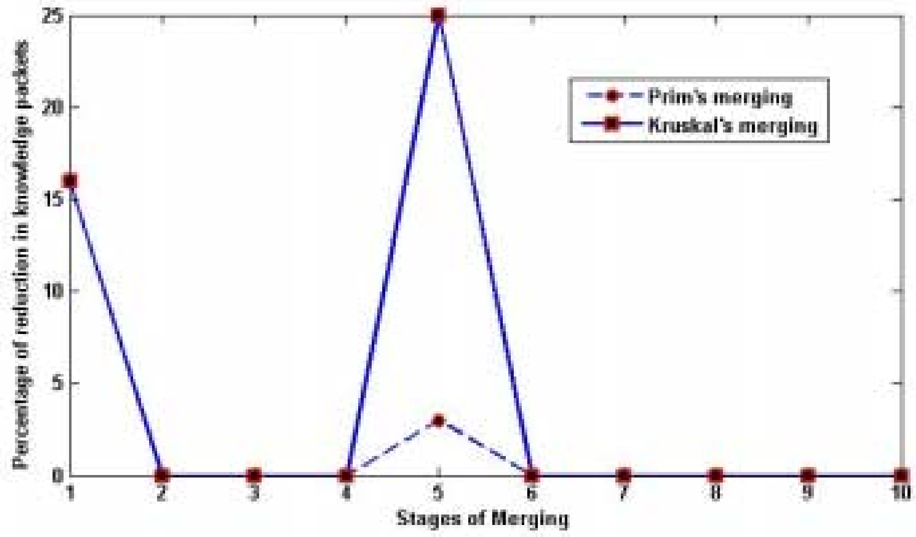


Fig. 11 700X3 dataset's percentage of reduction in *KP*s by *Prim*'s like and *Kruskal*'s likemethod of merging s

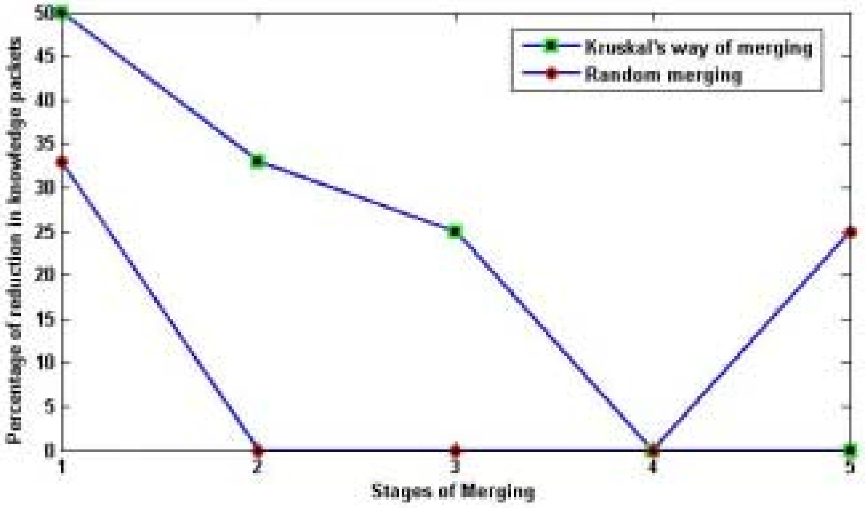


Fig. 12 Iris dataset's percentage of reduction in *KP*s by *Kruskal*'s like method and Random method of merging

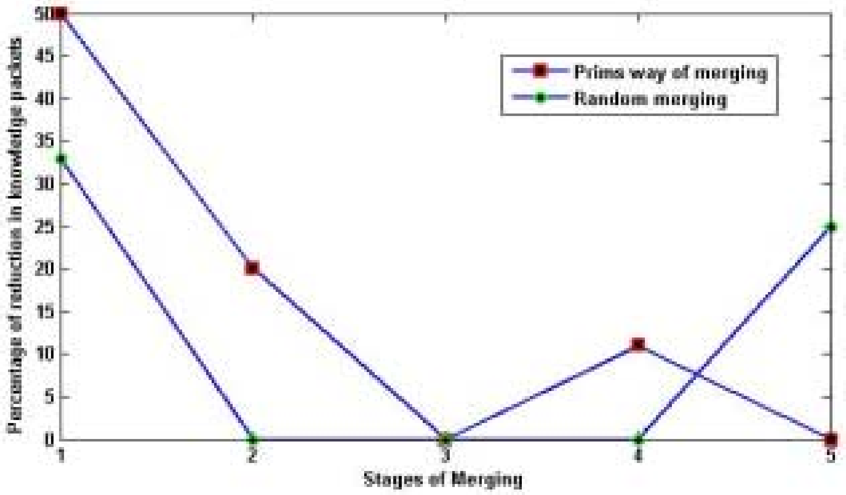


Fig. 13 Iris dataset's percentage of reduction in *KP*s by *Prim*'s and Random method of merging

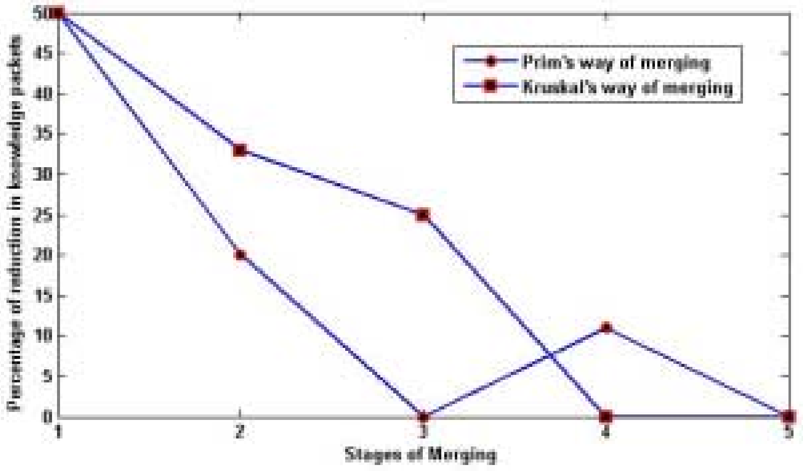


Fig. 14 Iris dataset's percentage of reduction in *KP*s by *Prim*'s like and *Kruskal*'s like methods of merging