Co-Clustering Algorithm: Batch, Mini-Batch, and Online

Hyuk Cho and Min Kyung An

***Abstract*—Unlike traditional one-way K-means clustering, co-clustering simultaneously cluster both data points and features of a two-dimensional data matrix. It is a powerful data analysis technique that can discover latent patterns hidden within particular rows and columns. Accordingly, co-clustering has been successfully applied to varied domains, including, but not only limited to, text clustering, microarray analysis, speech and video analysis, and natural language processing. Assuming a whole data matrix is available, usual co-clustering algorithm updates all row and column assignments in batch mode. In this paper, we develop an online incremental co-clustering algorithm that can update both row and column clustering statistics on the fly only for each available data point; thus, the proposed algorithm can handle stream data collected from sensor networks or handheld devices. Characteristics among batch, mini-batch, and online clustering and co-clustering algorithms are discussed and future work is provided.**

***Index Terms*—Batch, mini-batch, incremental, online co-clustering.**

I. Introduction

Nowadays the amount of data from scientific and commercial sources continues to ever evolve at an unprecedented rate. Extracting latent patterns from a given data becomes the first and the necessary step to explore the nature of data under study. However, processing the explosive amount of data is very challenging because of massive storage requirement as well as demanding computational complexity. In an effort to remedy storage and/or computing demand, researchers have proposed ideas such as efficient indexing using locality sensitive hashing [1] and parallelization of machine learning and data mining algorithms [2]–[4] using map-reduce [5]–[6] and/or using graphics processing units (GPUs) or multicores [7]–[9], to name a few.

In general, clustering has been employed to conduct the exploratory data analysis and to summarize large quantities of high-dimensional data. Traditional "single-sided" clustering focuses on only either row or column dimension, while co-clustering targets both object and feature dimensions and seeks "blocks" (or "co-clusters") of inter-related objects and features by alternating object clustering into feature clustering, and vice versa. Co-clustering is a novel exciting unsupervised data analysis paradigm that utilizes the duality of both the dimensions, discovers latent patterns, reduces dimensionality implicitly, generates compact representation, and reduces overall running time [10]–[12]. Although relatively newly developed, co-clustering is considered more desirable than traditional clustering. In addition, because of these desirable characteristics and the theoretical maturity, it has shortly become popular for diverse applications in web mining, image retrieval, recommendation, bioinformatics, pervasive/ubiquitous computing [10], [13]–[17] and more applications are expected. Other interesting applications of co-clustering algorithms can be found in the survey [18]. The authors of this paper have led the co-clustering research by developing state-of-the-art co-clustering algorithms and further highlighted the potential of co-clustering framework as an indispensable data exploratory analysis tool with varied advanced algorithmic strategies [10], [14], [15], [19].

This paper aims to further improve currently existing co-clustering algorithm so that it can efficiently handle streaming data. The answer is to adopt the "online" learning strategy, where "online" in this context means that the whole data is not available at the beginning. Therefore, an online algorithm does not keep the whole data in storage or in main memory, but it processes the available data one by one incrementally, only updating the affected statistics and keeping aggregates or necessary information. Ideally it also does this in a single pass, i.e. it just processes each instance (i.e., data point) at a time. The theory then goes that if enough data is available over some period of time, the characteristics/patterns of the data may still be captured; thus, grouping/class information latent in the whole data processed over time can be identified. Accordingly, this online update/learning fits well for many realistic scenarios in incremental data mining and machine learning problems.

The rest of this paper is organized as follows: In Section II, we discuss the paradigm-shift from batch to online learning, where we briefly compare offline learning with online learning. In Section III, we describe batch, mini-batch, and online clustering algorithms. Then, we provide details of batch, mini-batch, and online co-clustering algorithms, which are main topics, in Section IV. We discuss the experimental details with simulated data in Section V. Finally, the paper is concluded with some remark in Section VI.

II. Background

A. Paradigm-Shift from Offline to Online Learning

Traditional machine learning algorithms process and analyze the data in batch (or offline) mode, where the whole data is assumed to be available and processed at once; thus, the aforementioned challenges in storage and computation demand need to be addressed. To help mitigate this difficulty, online machine learning has been developed to operate the data in online, or one-pass streaming settings, rather than in offline. Although the main idea of online learning goes back

Manuscript received April 12, 2014; revised June 12, 2014.

H. Cho and M. K. An are with Sam Houston State University, Huntsville, TX 77341, USA (e-mail: hyukcho@shsu.edu, an@shsu.edu).

to the 50s [20], online machine learning [21] has become of great interest to researchers and practitioners due to the rapid surge of large scale practical applications in data mining and ubiquitous/pervasive computing. Some applications include, but not limited to, webpage ranking, online advertisement, stock prediction, recommendation system, spam E-mail filtering, and so on. The goal of online learning is to design light-weight algorithms in terms of both memory and speed, dealing with inputs coming over time with no further information available at once and thus resulting in accurate predictions over the streaming data.

B. Offline vs. Online Learning

In offline learning (also called as epoch learning), all the data are stored and can be accessed repeatedly. Batch learning is always offline. With offline learning, the objective function for any fixed set/dimension of statistics can be computed at each batch update and thus the progress of the learning can be observable, upon computing the (local) minimum/maximum of the objective function to any desired precision. In online learning, each object/case is discarded after it is processed and the affected statistics are updated, and thus online training is always incremental. With online learning, the progress of objective function values and the computation of the minimum/maximum objective function value do not make much sense, because the whole data is not considered in batch, but just each instance available at a time is processed. Accordingly, online learning is generally considered to be more difficult and unreliable than offline learning. However, notice that incremental learning can be done either offline or online. Offline incremental learning does not have the aforementioned difficulties with online learning.

III. One-Way K-Means Clustering

TABLE I: K-MEANS

(a) **Offline Batch K-means** *(Assume whole data is available.)*	
Step 1	Initialize row assignment and statistics.
Step 2	Repeat till convergence:
Step 2.1	Update ***all row*** assignments.
Step 2.2	Update ***all row*** statistics.
(b) **Offline Mini-batch K-means** *(Assume whole data is available.)*	
Step 1	Initialize row assignment and statistics.
Step 2	Repeat till convergence:
Step 2.1	Update **(randomly selected)** ***row*** assignments.
Step 2.2	Update ***affected row*** statistics.
(c) **Offline Incremental K-means** *(Assume whole data is available.)*	
Step 1	Initialize row assignment and statistics.
Step 2	Repeat till convergence.
	Do the following for every row:
Step 2.1	Update ***a row*** assignment.
Step 2.2	Update ***affected row*** statistics.
(d) **Online Incremental K-means** *(Assume new row is streamed.)*	
Step 1	Initialize row assignment and statistics to be empty.
Step 2	Repeat for ***each data available*:**
Step 2.1	Update ***a row*** assignment.
Step 2.2	Update ***affected row*** statistics.

One-way clustering, also known as K-means, targets one dimension and concentrates on grouping "similar" objects (at rows) or features (at columns), based on similarities amongst their features or objects. Different types and steps of one-way K-means clustering are described in Table I.

A. Offline Batch K-Means

Assuming the entire data are available throughout the whole learning process, K-means aims to partition n objects (at rows) into k clusters. Every object is assigned to one of k clusters and then initial statistics (i.e., centroids) are obtained (Step 1 of Table I(a)). Note other elaborate initialization heuristics can be applied. Based on distances between each object to all cluster centroids, all objects are re-assigned to the nearest cluster (Step 2.1) and then all statistics are updated (Step 2.2). This one full cycle (epoch) is repeated until satisfying one of the two convergence criteria (i.e., maximum iteration number or/and objective function value change).

B. Offline Mini-Batch K-Means

Every object is assigned to one of k clusters and then initial cluster centroids are calculated (Step 1 of Table I(b)). Based upon distances between each of randomly selected objects (i.e., mini-batch) [22], the selected objects are re-assigned to the cluster with minimum distance (Step 2.1) and then the affected cluster centroids are calculated (Step 2.2). This process is repeated until satisfying one of the aforementioned convergence conditions.

C. Offline Incremental K-Means

Every object is assigned to one of k clusters and initial cluster centroids are calculated (Step 1 of Table I(c)). Then, for every object, its cluster assignment and the update of affected statistics are performed incrementally, which is the one whole round of incremental update (i.e., Steps 2.1 and 2.2 for every object). This process is repeated until it satisfies one of the convergence conditions.

D. Online Incremental K-Means

Online refers to continuous stream of data; thus, it is always incremental as aforementioned. Parameter k is set to be 0 at the beginning [23], [24] so as to specify no cluster yet (Step 1 of Table I(d)). The initial data point or the first object that comes in itself is assigned as the first cluster and the data itself becomes the first centroid. When a new data point or object comes in, the distance between the recent object and the existing cluster centroids are calculated. The object is then re-assigned to the existing cluster if its distance to the cluster is smaller than the distance threshold value (predefined at the beginning). Otherwise, it is assigned as a new cluster. The gist of online mode is that the algorithm only needs to remember the statistics (e.g., mean and size) of each cluster. Once these variables are updated, each data point from a stream does not need to be kept in main memory. Furthermore, the hidden parameter k can be controlled by adjusting the distance threshold value. For example, if the threshold value is set to be higher than earlier one, the algorithm will generate less k.

IV. Co-Clustering

Co-clustering targets both the row and column dimensions

and concentrates on grouping by rows and columns that are similar, respectively. The most important step is that the row cluster centroid and the column cluster centroid are calculated from the co-cluster centroid, which is in fact the main difference from the usual K-means algorithm.

TABLE II: CO-CLUSTERING

	(a) **Offline Batch Co-clustering** *(Assume whole data is available.)*
Step 1	Initialize row/column assignment and statistics.
Step 2	Repeat till convergence:
Step 2.1	Update ***all row*** assignments.
Step 2.2	Update ***all cocluster*** and ***row*** statistics.
Step 2.3	Update ***all column*** assignments.
Step 2.4	Update ***all cocluster*** and ***column*** statistics.
	(b) **Offline Mini-batch Co-clustering** *(Assume whole data is available.)*
Step 1	Initialize row/column assignment and statistics.
Step 2	Repeat till convergence:
Step 2.1	Update **(randomly selected)** ***row*** assignments.
Step 2.2	Update ***affected cocluster*** and ***row*** statistics.
Step 2.3	Update ***all column*** assignments.
Step 2.4	Update ***all cocluster*** and ***column*** statistics.
	(c) **Offline Incremental Co-clustering** *(Assume whole data is available.)*
Step 1	Initialize row/column assignment and statistics.
Step 2	Repeat till convergence:
	Do the following for every row:
Step 2.1	Update ***a row*** assignment.
Step 2.2	Update ***affected cocluster*** and ***row*** statistics.
	Do the following for every column:
Step 2.3	Update ***a column*** assignment.
Step 2.4	Update ***affected cocluster*** and ***column*** statistics.
	(d) **Online Incremental Co-clustering** *(Assume each row is streamed.)*
Step 1	Initialize row/column assignment and statistics to be empty.
Step 2	Repeat for ***each data available***:
Step 2.1	Update ***a row*** assignment.
Step 2.2	Update ***affected cocluster*** and ***row*** statistics.
	Do the following for every column:
Step 2.3	Update ***a column*** assignment.
Step 2.4	Update ***affected cocluster*** and ***column*** statistics.

A. Offline Batch Co-Clustering

With the assumption of whole data availability at the beginning, currently existing offline batch co-clustering approaches [15]–[17] are composed of two main processes at each iteration, including the cluster assignment of whole rows and the update of affected row statistics (Steps 2.1 and 2.2 of Table II(a)), and the cluster assignment of whole columns and the update of affected column statistics (Steps 2.3 and 2.4). The iteration repeats until stopping criteria are satisfied.

B. Offline Mini-Batch Co-Clustering

Like offline mini-batch K-means, k and ℓ are given as inputs. It starts with initial k row clusters, ℓ column clusters, and the corresponding co-cluster statistics (Step 1 of Table II(b)). Based on distances between each of selected objects (i.e., mini-batch) and the row cluster centroids, each of the selected objects is re-assigned to the row cluster with minimum distance (Steps 2.1 and 2.2). Similarly, the column cluster update step is followed (Steps 2.3 and 2.4). This process is repeated until convergence.

C. Offline Incremental Co-Clustering

Like offline incremental K-means, k and ℓ are assumed at the beginning. It starts with initial k row clusters and ℓ column clusters, both of which are calculated from the corresponding co-cluster statistics (Step 1 of Table II(c)). Each row is re-assigned to the row cluster with minimum distance and both the affected cocluster and row statistics are incrementally updated (Steps 2.1 and 2.2). After one row update, the corresponding column is updated (Steps 2.3 and 2.4). This process is repeated until convergence.

(Initialization Step)
Set row and column thresholds, *rowThreshold* and *colThreshold*.
Initialize co-cluster centroid with the first data.
Compute row centroid from co-cluster centroid.
Set both row and column cluster sizes to be 1 (i.e., k=1 and ℓ=1).
(Online and Incremental Update)
For each data point from stream do:
  (Online Row Assignment)
  Compute *minDistance* to row centroid.
  If *minDistance* > *rowThreshold*:
    (Introduce New Row Cluster)
    Update co-cluster centroid with new row centroid.
    Increase row cluster size by 1 (i.e., k++).
  Else:
    (Assign into Old Row Cluster)
    Update co-cluster centroid from row assignment.
  (Incremental Column Assignment)
  Compute column centroid from co-cluster centroid.
  For each column of all the rows do:
    Compute *minDistance* to column centroid.
    If *minDistance* > *colThreshold*:
      (Introduce New Column Cluster)
      Update co-cluster centroid with new column centroid.
      Increase column cluster size by 1 (i.e., ℓ++).
    Else:
      (Assign into Old Column Cluster)
      Update co-cluster centroid from column assignment.
      Compute column centroid from co-cluster centroid.
  Compute row centroid from co-cluster centroid.

Fig. 1. Online row clustering followed by incremental column clustering.

D. Online Incremental Co-Clustering

The proposed work is inspired by online incremental K-means (Table I(d)), where online K-means does not require whole data, letting each data come and go away. To be more specific, the statistics for each new instance are updated at each time, rather than those for all data are updated at once. This is particularly desirable when the instances are gradually obtained over some period of time; however we want to start learning (i.e., clustering) data before we have seen all of the instances. Therefore, the concept of "online" learning offers us the flexibility of updating only the affected statistics (e.g., centroids, cluster sizes, etc.) as each new data point arrives. In this paper, we extend this online learning idea to Minimum Sum-Squared Residue Co-clustering (MSSRCC) [10], which belongs to the general co-clustering framework, called Bregman Co-clustering (BCC) [11], as shown in Table II(d).

Motivated by the heuristic used in [23], [25]–[27], we can elaborate Steps 2.1 and 2.2 in Table II(d) so as to find the row cluster that a new row belong to as follows. If the similarity between a row and the closest row cluster is greater than the specified row threshold (denoted as *rowThreshold*) as described at **(Online New Assignment)** of Fig. 1, create a new row cluster for the data **(Introduce New Row Cluster)**, which will be a singleton cluster with the new row only. Otherwise, update the affected statistics of the closest row cluster that already exists, accordingly **(Assign into Old**

Row Cluster). The idea behind this heuristic is that if a data is not close (or similar) enough to belong to one of existing clusters, then we create a new cluster since the data different from other data in terms of the specified threshold. Therefore, we can control number of row clusters as well as number of column clusters by adjusting both the row and column thresholds. The same scenario can be applied to find column clusters at Steps 2.3 and 2.4 in Table II(d).

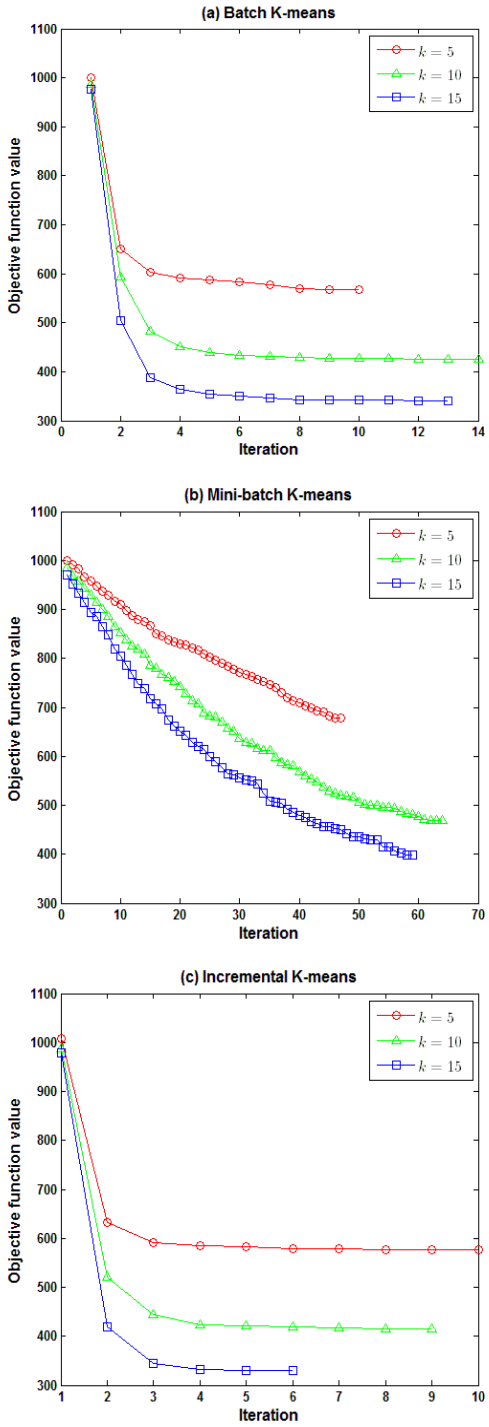

Fig. 2. Monotonicity of batch, mini-batch, and incremental K-means algorithms. Three clusters (i.e., k=5, 10, and 15) are considered. Note that mini-batch size of (b) batch K-means is set to be 20.

The proposed co-clustering algorithm (Fig. 1) performs online row clustering, followed by incremental column clustering. As in [23], [26], both the number of row clusters (i.e., k) and the number of column clusters (i.e., ℓ) are not determined; thus, $k = \ell = 0$ at the beginning. Both row and column threshold values have been set to be very small real values, respectively. Remember that online learning assumes that each instance comes in over each time. The instance (or feature) that comes first in itself is assigned as the first row (or column) cluster and the data itself becomes the first co-cluster centroid. Note that both the row and column cluster centroids are calculated from the co-cluster centroid.

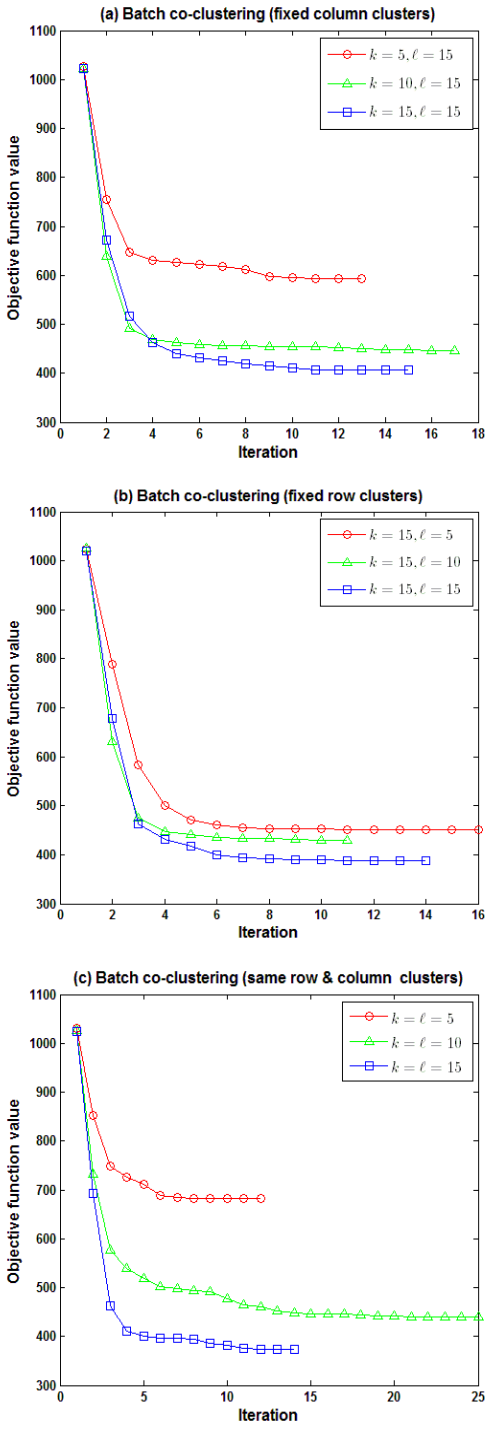

Fig. 3. Monotonicity of batch co-clustering algorithm. Three cases are considered: (a) fixed row cluster, (b) fixed column cluster, and (c) same row and column cluster numbers.

After then, the distance between the recent instance and the cluster centroids are calculated. The instance is re-assigned to the existing cluster if its distance to the closest cluster is smaller than the threshold value. Otherwise, it is

assigned as a new cluster. Like online K-means, the algorithm only needs to remember the mean and the size of affected row (or column) clusters and co-clusters. Once these statistics are updated, each data point will not be kept in main memory. Furthermore, as explained before, the resulting k and ℓ can be exploited by adjusting row and column threshold values, respectively. For example, if high threshold values are set, less numbers of row and column clusters will be generated.

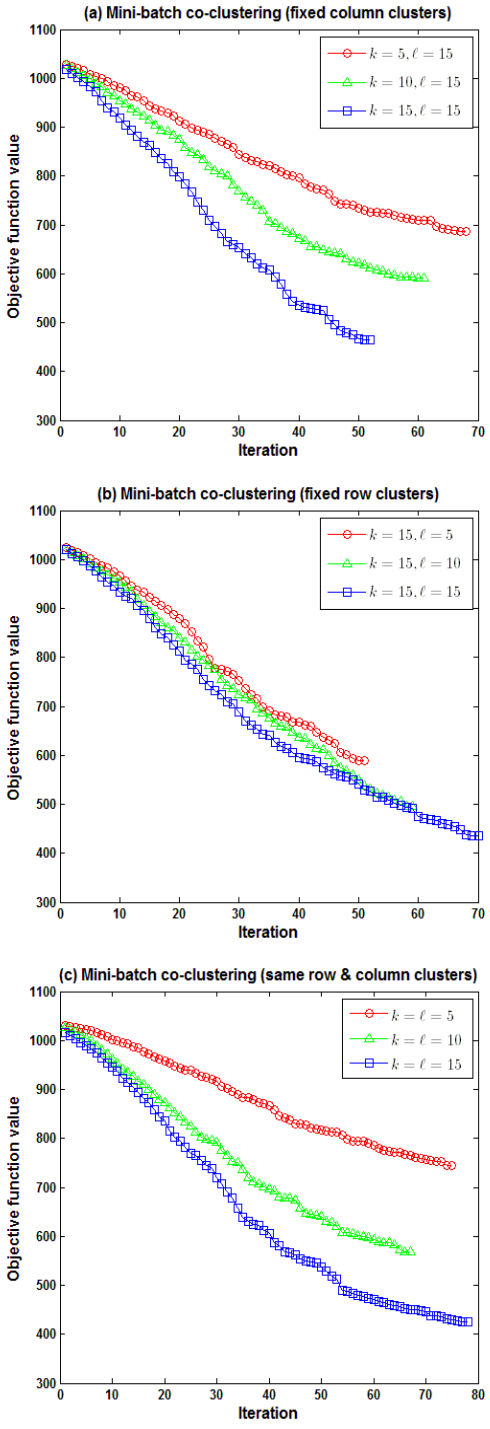

Fig. 4. Monotonicity of mini-batch co-clustering algorithm. Three cases are considered: (a) fixed row cluster, (b) fixed column cluster, and (c) same row and column cluster numbers.

V. Experimental Result

The aforementioned algorithms, including (offline) batch, (offline) mini-batch, (offline) incremental, and online (incremental) algorithms for K-means and Co-clustering, respectively, are implemented in Python. Their correctness was validated with various randomly generated real-valued data matrices. However, our discussion targets one specific real-valued dataset, 'LIBRAS Movement Dataset,' which contains 360 instances and 90 features. There exist 15 classes of hand movement types and each class has 24 instances.

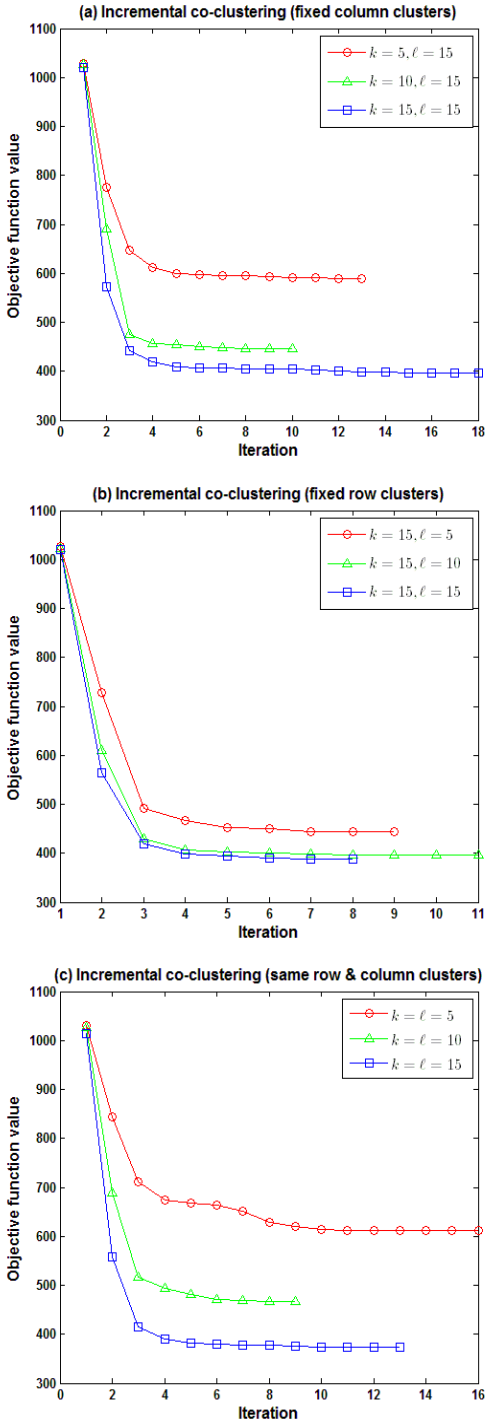

Fig. 5. Monotonicity of incremental co-clustering algorithms. Three cases are considered: (a) fixed row cluster, (b) fixed column cluster, and (c) same row and column cluster numbers.

The dataset was originally generated for classification of the official Brazilian sign language [28] and is publicly available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Libras+Movement).

Fig. 2 depicts the change of objective function values with 5, 10, and 15 row clusters for the three K-means algorithms. To be specific, Fig. 2(a) shows that the objective function value of batch K-means monotonically decrease as iteration increases for all the considered row clusters. Similarly, Fig. 2(b) and Fig. 2(c) illustrate the monotonicity of objective function value change for mini-batch K-means and incremental K-means, respectively.

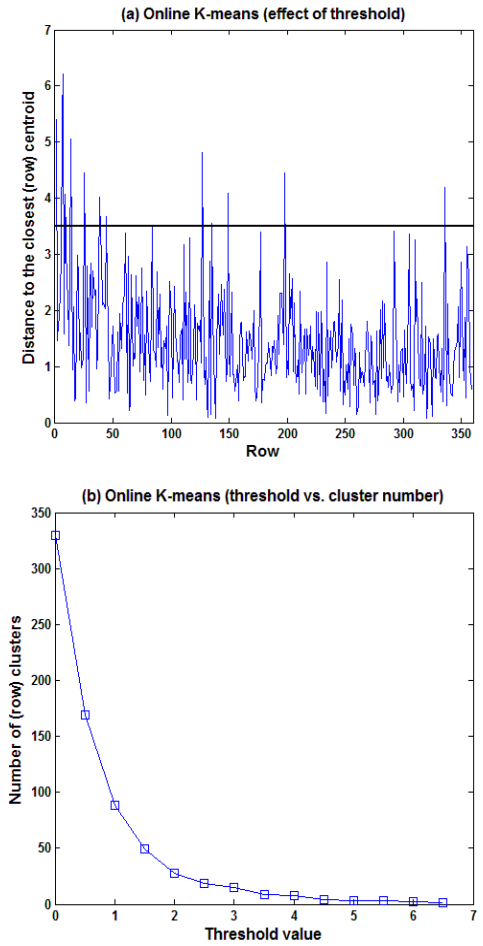

Fig. 6. Threshold value vs. resulting cluster number. Note that the horizontal line in (a) denotes the row threshold value (i.e., 3.5) for the experiment.

Fig. 2(b) shows the trace of objective function values with the specific mini-batch size of 20; thus, iteration number can be changed with varied mini-batch sizes. Each K-means algorithm goes through a different number of iterations to converge, however we have witnessed that mini-batch (Fig. 2(b)) generally requires more iterations than the other two algorithms (Fig. 2(a) and Fig. 2(c)). In fact, direct comparison of iteration number does not make much sense, because each algorithm processes different number of data points. To be more specific, batch, mini-batch, and incremental algorithm handles whole data points, selected data points (i.e., mini-batch), and only one data point, respectively. Therefore, we can expect that batch algorithm takes longest time and incremental one takes least time for each iteration.

Interestingly, mini-batch K-means slowly converges without much change of objective function values over the whole process, while both batch and incremental ones have much change of objective function values during the a couple of initial iterations and converge slowly as iteration number increases. The initial change seems to be critical to overcome local optima. It is well known that usual clustering algorithms improve their objective function values more at early stages because of many new cluster assignments or changes.

Fig. 3 – Fig 5 also display the monotonicity of objective function value change with the three co-clustering algorithms. The monotonicity of objective function value change for the incremental update with co-clustering (i.e., Figs. 3(c), 4(c), and 5(c)) was already proved in [29]. For the three co-clustering algorithms, the objective function value changes were measured with the following three scenarios: (1) $k = 5$, 10, and 15, with fixed ℓ=15; (2) $\ell = 5$, 10, and 15, with fixed $k = 15$; and (3) $k = \ell = 5$, 10, and 15. For all the scenarios, the monotonicity of objective function value change is observed. This observation is consistent with the proof in [29], where this property was also suggested to remove the degeneracy problem (i.e., empty clusters) in usual clustering algorithms. Like the min-batch K-means in Fig. 2(b), mini-batch co-clustering in Fig. 4 requires more iteration to converge, slowly converges over the whole iteration, and results in worse local minima than the ones that batch and incremental co-clustering algorithms achieve.

Fig. 6 demonstrates the effect of (row) threshold value on the number of (row) clusters for online K-means. Note that the thick horizontal line in Fig. 6(a) denotes the specific row threshold value of 3.5, with which the algorithm results in the actual 15 row clusters in the dataset. As discussed before, rows are far (i.e. more than the row threshold value) to the closest row cluster will introduce new singleton clusters, while rows are close enough (i.e., less than the row threshold value) to the closest row cluster will belong to one of exising row clusters. As shown in Fig. 6(b), by increasing (or deceasing) row/column threshold value to violate (or satisfy) the condition, "If *minDistance* > *rowThreshold*:," one can adjust the resulting row/column cluster number.

VI. Conclusion and Remark

The contribution of the proposed approach can be summarized as follows. First, we compare and contrast characteristics of batch, mini-batch, incremental, and online algorithms for both K-means and co-clustering. As the online co-clustering algorithm avoids repeated distance or similarity comparisons between every data point and every cluster centroid, it has a desirable property of the handling online stream data generated from sensor networks. In particular, online incremental co-clustering algorithm is able to identify latent local patterns among particular contexts and different conditions for each data point. In addition, once the row and column threshold values are set, no explicit user input for the resulting numbers of row and column clusters is necessary, since both the numbers can be automatically determined.

In this paper, we emphasize more on algorithm implementation than on theoretical analysis; thus, rigorous theoretical analysis on storage and complexity requirement for the considered algorithms is necessary. Comprehensive experimental study should be performed with actual datasets with ground truth class labels and the clustering performance should be validated with explicit performance measures.

The proposed framework is particularly based upon Minimum Sum-Squared Residue Co-clustering (MSSRCC) [10]. Therefore, the approach can be extended to the general co-clustering framework of Bregman Co-clustering (BCC) algorithms [11]. To do so, we need to investigate what summary statistics of each instance are affected and how they can be efficiently updated for each instance. Furthermore, it is interesting to see if the resulting online co-clustering algorithms can be used to improve other learning processes.

REFERENCES

[1] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. VLDB'99*, pp. 518-529, 1999.

[2] I. S. Dhillon and D. S. Modha, "A parallel data-clustering algorithm for distributed memory multi-processors," *Large-Scale Parallel Data Mining, LNAI*, vol. 1759, pp. 245–260, 2000.

[3] C. Pizzuti and D. Talia, "P-AutoClass: scalable parallel clustering for mining large data sets," *IEEE TKDE*, vol. 15, no. 3, pp. 629–641, 2003.

[4] J. Zhou and A. Khokar, "ParRescue: scalable parallel algorithm and implementation for biclustering over large distributed datasets," presented at ICDCS'06, 2006.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. OSDI'04*, 2004, pp. 137–150.

[6] C-T. Chu, S. Kim, Y-A. Lin, Y. Yu, G. R. Bradski, A. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *Proc. NIPS'06*, 2006, pp. 281–288.

[7] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating map-reduce for multi-core and multiprocessor systems," in *Proc. HPCA'07*, 2007, pp.13-24.

[8] X. Qiu, G. Fox, H. Yuan, S.-H. Bae, G. Chrysanthakopoulos, and H. Nielsen, "Parallel Data Mining on Multicore Clusters," in *Proc. the Seventh International Conference on Grid and Cooperative Computing*, 2008, pp. 41–49.

[9] J. Eastep, D. Wingate, and A. Agarwal, "Smart data structures: An online machine learning approach to multicore data structures," in *Proc. ICAC'11*, 2011, pp. 11–20.

[10] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra, "Minimum sum squared residue based co-clustering of gene expression data," in *Proc. SDM'04*, 2004, pp. 114–125.

[11] A. Banerjee, I. S. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha, "A generalized maximum entropy approach to bregman co-clustering and matrix approximations," *JMLR*, vol. 8, pp. 1919–1986, 2007.

[12] B. Kwon and H. Cho, "Scalable co-clustering algorithms," *ICA3PP'10*, C.-H. Hsu *et al.*, ed., pp. 32–43, Part I, LNCS, vol. 6081, 2010.

[13] Y. Cheng and G. M. Church, "Biclustering of expression data," in *Proc. ISMB'00*, 2000, pp. 93–103.

[14] H. Cho and I. S. Dhillon, "Co-clustering of human cancer microarrays using minimum sum-squared residue co-clustering algorithm," *IEEE/ACM TCBB*, vol. 5, no. 3, pp. 385–400, 2008.

[15] M. J. Deodhar, G. Gupta, J. Ghosh, H. Cho, and I. S. Dhillon, "A scalable framework for discovering coherent co-clusters in noisy data," presented at IEEE ICML, 2009.

[16] S. Jeong, S. Kalasapur, D. Cheng, H. Song, and H. Cho, "Clustering and naïve Bayesian approaches for situation-aware recommendation on mobile devices," presented at IEEE ICMLA, 2009.

[17] H. Cho, D. Mandava, Q. Liu, L. Chen, S. Jeong, and D. Cheng, "Situation-Aware on mobile phone using co-clustering: Algorithms and extensions,"in *IEA/AIE'12*, H. Jiang *et al.*, ed., pp. 272–282, LNAI, vol. 7345, 2012.

[18] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: A survey," in *Proc. IEEE/ACM TCBB*, vol. 1, no. 1, pp. 24–45, 2004.

[19] H. Cho, "Data transformation for sum squared residue," in *PAKDD'10*, M. J. Jaki *et al.*, ed., pp. 48–55, Part I, LNAI, vol. 6118, 2010.

[20] J. Hannan, "Approximation to Bayes risk in repeated play," *Contributions to the Theory of Games*, vol. 3, pp. 97–139, 1957.

[21] S. Shalev-Shwartz, "Online learning: Theory, algorithms, and applications," Ph.D. Dissertation, The Hebrew University, 2007.

[22] D. Sculley, "Web-scale K-means clustering," in *Proc. WWW'10*, 2010.

[23] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "Incremental K-means algorithm," in *Proc. the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 218, pp. 783–795, 2003.

[24] S. Chakraborty and N. K. Nagwani, "Analysis and study of incremental k-means clustering algorithm," *High Performance Architecture and Grid Computing, Communications in Computer and Information Science*, vol. 169, pp. 338–341, 2011.

[25] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "Selection of K in K-means algorithm," in *Proc. the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, pp. 103–119, 2005.

[26] D. Bollegala, Y. Matsuo, and M. Ishizuka, "Measuring the Similarity between Implicit Semantic Relations from the Web," in *Proc. WWW'09*, 2009, pp. 651–660.

[27] D. Bollegala, Y. Matsuo, and M. Ishizuka, "Relational duality: Unsupervised extraction of semantic relations between entities on the web," in *Proc. WWW'10*, 2010, pp. 151–160.

[28] D. B. Dias, R. C. B. Madeo, T. Rocha, H. H. Biscaro, and S. M. Peres, "Hand Movement Recognition for Brazilian Sign Language: A Study using Distance-based Neural Networks," in *Proc. IJCNN'09*, pp. 697-704, 2009.

[29] H. Cho, "Co-clustering algorithms: Extensions and applications," Ph.D. dissertation, Dept. Computer Sciences, The University of Texas at Austin, TX, 2008.

Hyuk Cho received the B.E. degree in computer engineering from Chonbuk National University, Korea, the M.A. degree in computer science from Korea University, Korea, and both the M.S. and the Ph.D. degrees in computer sciences from the University of Texas at Austin.

He is an assistant professor in the Department of Computer Science at Sam Houston State University, Huntsville, Texas, USA. He is known for his work on co-clustering algorithms, their extensions, and their applications to various practical tasks in real world problems. His research interests include data mining, statistical pattern recognition, machine learning, pervasive computing, bioinformatics, and data science. Previously he worked on linear matrix inequality and soft computing, including neural networks, evolutionary computation, genetic algorithm, and fuzzy/rough set theory.

Prof. Cho is a member of IEEE/ACM and SIAM.

Min Kyung An received her Ph.D. in computer science from the University of Texas at Dallas in August 2013, and her M.S. in computer science from the University of Texas at Arlington in August 2007. During her M.S. studies, she received the Graduate Studies Abroad Program Scholarship funded by the Korean government.

She is currently an assistant professor in the Department of Computer Science at Sam Houston State University, Huntsville, Texas, USA. Her major research areas include wireless ad hoc and sensor networks, social networks, design and analysis of approximation algorithms, graph theory, and program analysis.

Prof. An is a member of IEEE/ACM.