International Journal of Advanced Smart Convergence Vol.8 No.3 46-53 (2019)
http://dx.doi.org/10.7236/IJASC.2019.8.3.46

IJASC 19-3-7

The Effect of Bias in Data Set for Conceptual Clustering Algorithms

Gye Sung Lee

Department of Software, Dankook University, Korea
gslee@dankook.ac.kr

Abstract

When a partitioned structure is derived from a data set using a clustering algorithm, it is not unusual to have a different set of outcomes when it runs with a different order of data. This problem is known as the order bias problem. Many algorithms in machine learning fields try to achieve optimized result from available training and test data. Optimization is determined by an evaluation function which has also a tendency toward a certain goal. It is inevitable to have a tendency in the evaluation function both for efficiency and for consistency in the result. But its preference for a specific goal in the evaluation function may sometimes lead to unfavorable consequences in the final result of the clustering. To overcome this bias problems, the first clustering process proceeds to construct an initial partition. The initial partition is expected to imply the possible range in the number of final clusters. We apply the data centric sorting to the data objects in the clusters of the partition to rearrange them in a new order. The same clustering procedure is reapplied to the newly arranged data set to build a new partition. We have developed an algorithm that reduces bias effect resulting from how data is fed into the algorithm. Experiment results have been presented to show that the algorithm helps minimize the order bias effects. We have also shown that the current evaluation measure used for the clustering algorithm is biased toward favoring a smaller number of clusters and a larger size of clusters as a result.

Keywords: *COBWEB model, Data ordering, Clustering, Conceptual learning, Bias problem*

1. INTRODUCTION

The challenges of big data are prolific over all types of field that collects, stores and analyzes data. Advances of technologies on data collection and storage have evolved a variety of analysis methodologies[1-3]. It is not simply a matter of size, but rather a matter of quality and diversity of data[4]. Unsupervised learning methods such as conceptual clustering use an evaluation function as a criterion for categorizing data[3]. An important factor in setting up the evaluation function is the consideration of bias. The choice of the evaluation function renders the difference in final outcomes due to the fact that the bias involved in the evaluation function favors toward certain types of patterns [5-6]. For example, simplicity is one of widely used biases in data mining algorithms and helps simplify the complex and detailed relations among data objects and reduce the size of the clusters. A number of learning methods adopt the incremental property, especially for the construction of a

Manuscript Received: July. 4, 2019 / Revised: July. 7, 2019 / Accepted: July. 13, 2019
Corresponding Author: gslee@dankook.ac.kr
Tel: +82-10-3344-7529, Fax: +82-31-8005-3223
Dept. of Software, Dankook University, Korea

clustering hierarchy. The incremental property also creates a side effect that the input instances for classification are dependent on the existing structure. If the instances are entered in a different order, they would settle in different places of the hierarchy or even result in creating a totally different hierarchy. In this paper, we study the conceptual clustering algorithm, COBWEB [7,8], and its order bias issues[9,10]. In these researches, problems caused by the order of data were analyzed and several enhancements have been made to reduce the effect of the order problem. We revise these algorithms to minimize the effect of the order bias. We compare the new algorithm with previous algorithms.

2. COBWEB AND ITS DERIVATIVE

The COBWEB algorithm was developed based on the models of concept formation and human cognitive recognition. It accepts attribute-value pairs as its input and produces a concept tree as the output. It is called an incremental learner because when a data object is entered, nodes on the tree are restructured. In some cases it may change the entire structure of the tree considerably. COBWEB uses the category utility (CU) measure [7] as the criterion function for determining partitions in the hierarchy. The expected value of CU used in COBWEB is defined as $\mathrm{P}(A_i = V_{ij}|C_k)^2$. If the given data object belongs to a cluster C_k, the CU value implies the probability that A_i has the value V_{ij}, meaning the probabilistic match of the data object with the cluster. The role of category utility is to tradeoff between maximizing the intra-class similarity and inter-class dissimilarity. CU was used as a basis for incremental clustering. However, COBWEB also has several limitations in building its clustering tree like other hierarchical clustering algorithms:

1. The tree constructed by COBWEB is affected by the order that the data is entered.
2. CU includes the term $\mathrm{P}(C_k)$ that favors a bias toward larger size clusters. The consecutive, incremental presentation of a group or groups of highly similar data objects may result in a skewed partition structure.
3. Individual leaf nodes may not be useful with respect to the viewpoint of concept formation.
4. If the data set is known beforehand, there would be a chance to improve the order dependence by an analytical approach to re-arrangement of the data set.

With a certain data ordering, a data object could end up in an inappropriate place where the node is determined to be a meaningless concept or structurally redundant. To mitigate this problem, COBWEB introduced a few operators that restructure the internal nodes of the structure during the clustering process: split, merge, and promote. This approach has limitations in that it can enhance the quality of structure in a small section of the tree.

An alternative solution, called iterative optimization, was suggested in [9]. The basic idea in this solution is that the search strategy should consistently construct a clustering of high quality, but be computationally inexpensive as well. It partitions the search and constructs a tentative clustering for initial examination. Then several search operators are used to iteratively optimize the clustering. One of the operators that this method adopted is the redistribution operator that could shift the subtree containing a set of data objects from one place of the hierarchy to another. This method is different from [11] in that an individual data object (rather than a group of data objects) is redistributed for a better place. Both methods look for a possible reduction in the effect caused by the order bias. It could be done by redistributing wrongly placed data objects and reassigning them to better places. The ITERATE algorithm [11] claims that the best sequence of data objects could be obtained by arranging most dissimilar objects in neighboring places. This implies that the best sequence leads to a better clustering than COBWEB. It would be impossible to claim that the best sequence of data ordering could be found. However, as will be shown from the experiments, a data ordering could lead to a better clustering in terms of a specified measure (e.g., partition scores). ITERATE attempts to create an ordering in which each data object is most distinct from its neighbors. However, the limitation comes from the fact that the ordering scheme adopted in the algorithm, called an anchored dissimilarity ordering (ADO), does not take the information about the

possible number of clusters into account to the data ordering. If an object in the next order is picked in such a way that the object is most different from all the previously ordered data objects, it would turn out to favor more divisive clustering. Since the number of clusters is inherently not known in advance in unsupervised learning, it rather begins with a user selected number, n, for the expected number of clusters. It then finds the maximally different object from the previous n objects, and places it in order. Even if the number of clusters is known before clustering, there is a great chance that the selection of the first data object would result in a different data ordering, affecting the final clustering. In summary, it is hard to claim that the data ordering itself would guarantee best results. ITERATE introduced a redistribution operator to reduce the order bias effect more globally. Unlike other operators such as split, merge, and promote, the redistribution operator takes all the data objects and reassigns them to more appropriate clusters in order to optimize a chosen evaluation function. The category match (CM) [7] is used to measure the similarity between a data object and a cluster. ITERATE gives up the incremental property of hierarchical clustering. If the goal is to achieve data exploration and the data set is already given, it is not necessary to be incremental. ITERATE takes this approach and gets more balanced and stable clustering by exploiting the characteristics of data objects before clustering.

3. REIT ALGORITHM

In our algorithm to be presented in this section, the number of clusters implies important information in data object ordering. Our algorithm is based on the following observations. Firstly, the total distance among data objects is assumed to be maximal when the data objects are ordered according to the probable number of clusters. This means that the next data object to be selected is the one with the maximal difference from the previous n data objects, where n is the expected number of clusters of the final clustering, which is not known beforehand. Secondly, even if the final results are affected by the data order fed into a clustering algorithm, it is unlikely that the clustering would drastically change in terms of the number of clusters when clustering is attempted with different orders. From these observations, we devise the centric data sorting algorithm that produces a maximally distant data ordering, given the number of clusters. When the first clustering from the given data set is created, data objects within each cluster are similar together and data objects between clusters are distinct as the category utility is designed for. Data objects from the initial clustering are rearranged into a new ordering for better results. Given n clusters from the initial clustering, an ordering with the maximal overall distance can be obtained by picking up each center of the clusters in turn. The data objects of this order are used for the next clustering. The center of each cluster is determined to be the one that maximizes $\sum P(A_i = V_{ij})$ when an attribute, A_i, has a value V_{ij}.

```
for i=1,2, … {
  OCi = {Oij | cluster i, data object j}
  while ( ∃ OCkl in any OCk) {
     for Ck, k = 1,2, … {
        center = maxcenter = 0;
        for Okl, l = 1,2, … {
           center = ∑i P(Ai=Vij);
           if (center > maxcenter) {
              Okc = Okl ;
              maxcenter=center;
           }    // end of if
        }       // end of for
        output Okc ;
        OCk = OCk – {Okc};
       } // end of for
    }    // end of while
}    // end of for
```

Figure 1. Centric data sorting algorithm

The better ordering of data to be processed would be set up when data objects are ordered with respect to the size of the clusters. If all the data objects of a cluster have been used then the remaining clusters take turns for data selection. If only one cluster remains, the data objects compete with each other to be selected, which is maximally different from the previous data objects. The centric data sorting algorithm is described in Figure 1. $C = \{C_1, C_2, \cdots C_n\}$ denotes n clusters and O_{kc} denotes a data object c in cluster C_k. V_{ij} represents a value for attribute A_i of data object j.

The objects in each cluster are given from the previous clustering. The object in the center of the cluster is defined to be the most common object in the cluster. The similarity between two objects is computed and added up for all the pairs of objects. The object with the highest value is selected to be the most common object in the cluster. This value for an object can be easily computed by taking $P(A_i = V_{ij})$ from each object, instead of comparing each object with others. If a center object is selected, it is removed from the cluster and the same procedure is started for the next cluster.

1. initialize PS to be 0
2. construct a hierarchical structure by COBWEB algorithm
3. select initial partitions from the hierarchy
4. apply redistribution operator over partitions according to CM
5. update PS for each partition
6. if there is no improvement in PS, then stop
7. otherwise, rearrange the data objects by the centric data sorting
8. repeat the procedure from step 2

Figure 2. The REIT algorithm

Once the centric data sorting algorithm reorders the data objects, a new partitioning is constructed by the same clustering procedure. The partition score (PS) is defined as the average category utility of clusters in Equation (1).

$$\text{PS} = \sum_k CU(C_k)/(\#\ of\ \ clusters) \tag{1}$$

This partition score measures the quality of clustering and is used to determine whether the new clustering improves. If the PS value increases, the whole process repeats with the centric data sorting. The overall control structure of the revised algorithm, REIT, is shown in Figure 2. The category utility measure [7] is used to measure the similarity between a data object and a cluster. The partition score is initially set to zero. The first step of main loop is to construct a hierarchical structure by original COBWEB algorithm. From this hierarchy, the partitioning is determined. Redistribution procedure starts with the partitions. The partition score is computed over the partitioning. If there is no improvement in terms of partition score, the algorithm stops. Otherwise, the centric data sorting creates new data sequence by rearranging the data objects. Then the partitioning loop will begin with the new data sequence.

REIT, ITERATE, and COBWEB all use the category utility for clustering. Each algorithm has a different control structure and adopts different operators. Since both REIT and ITERATE reach the final clustering by repeating the core of the clustering algorithm, the efficiency issue will need to be addressed on how they quickly converge to the final result. The preprocessing (ordering) procedure in ITERATE requires of $O(n^2)$ where n is the number of data objects, but REIT can save the need of the preprocessing. REIT instead needs a hierarchical tree and then redistribution operation next for initial clustering, but the computational cost would not exceed $O(n \log n)$. Since the centric data sorting does not compare individual objects but rather computes $P(A_i = V_{ij})$, the computation could be done with $O(n)$ if radix sorting is applied.

4. EXPERIMENT RESULTS

In this section, the REIT algorithm is applied to the Balloon database[12], to show how it works. The data set is listed in Table 1. It has six attributes, id, inflation result, color, size, act, and age. There are four data sets representing different conditions for the experiment. One of them, the small-yellow data set is selected for experiment. The original data set has 20 data objects in the database; each of which consists of values for 4 attributes (color, size, act, and age) and 1 class label (inflated). The id attribute is just used for identification purpose and is removed from data set for the clustering process. The expected conclusion from the data set is that if the color is yellow and the size is small, then the balloon is inflated. Otherwise it is deflated. For the experiment to be simple, we collect only 12 objects and take the class label as another attribute for clustering. Five of the data objects are inflated and the rest are deflated.

Table 1. An inflated data set

id	inflated	color	size	act	age
D1	T	Yellow	Small	Stretch	Adult
D2	T	Yellow	Small	Stretch	Child
D3	T	Yellow	Small	Stretch	Child
D4	T	Yellow	Small	Dip	Adult
D5	T	Yellow	Small	Dip	Child
D6	F	Yellow	Large	Stretch	Adult
D7	F	Yellow	Large	Dip	Adult
D8	F	Yellow	Large	Dip	Child
D9	F	Purple	Small	Stretch	Adult
D10	F	Purple	Small	Stretch	Child
D11	F	Purple	Small	Dip	Adult
D12	F	Purple	Large	Dip	Child

As the first step of the algorithm, the data set is shuffled by randomly arranging the individual objects to avoid the possible order problem caused by the initial data arrangement. Before the data objects are clustered in the form of a hierarchy according to the COBWEB algorithm, the data sequence is ordered based on an anchored dissimilarity ordering algorithm, which adopts a complete-link approach in generating data orderings. The data object to be chosen next in the order is the one that is maximally distant from the previous data objects. The measure of maximal dissimilarity is defined as the sum of the Manhattan distances between the object and the previous n objects in the order. The window size, n, is user defined, and it is intuitively expected to be the actual number of clusters, which in most cases cannot be known in advance. It would be possible that the selection of the value of n leads to a variety of changes in the final clustering. Since the most appropriate value of n would not be available until the data set is fully investigated and analyzed, the selection of n for an unknown data set would not be possible. Therefore, the quality of the clustering cannot be guaranteed. We select a number in a range of 2 to 10 for the experiment. After the data sequence is determined by initially sorting the data objects, the original COBWEB algorithm generates a clustering hierarchy. The generated hierarchy of the balloon data set is shown in Figure 3.

Following the CU values at each node in a top-down manner, we notice that the values are changing from an increasing rate at the top levels, to decreasing at a certain point as it goes down through the hierarchy. This indicates that nodes in the lower part of the tree are not as effective as the upper part of the tree in terms of the category utility. Traversing the tree, when a node has the peak value along the path from the root to a leaf node it would be a candidate for base partitions. If any child node has a greater CU value, the entire nodes at the level will be chosen. For example, because one of the three nodes on the right side of the tree has a value greater than

its parent node, all its sibling nodes are chosen to represent the partitions. This way guarantees that the coverage is complete. In the current example, the shaded five nodes are selected as the base partitions.

As discussed in the ITERATE algorithm, initial partitions are more elaborated by redistributing the data objects in the initial partition to groups so as to optimize a chosen evaluation function. The category match (CM) is used to measure the similarity between a data object and a cluster description. In the Balloon data set, the redistribution operator does not move any data object to other partitions. Therefore, five clusters obtained from the initial partitioning remain as the final partitioning. All the inflated balloons (D1 through D5) are together in one node and deflated balloons are split into four different nodes. From this experiment, we notice that too many splits would not lead to quality clustering. It rather yields fragmented clusters. These initial partitions are used to form a new data sequence. When the size of the window for sorting by ITERATE is set to three, the initial random ordering and the anchored dissimilarity ordering produces a sequence: (D12, D1, D5, D9, D8, D6, D2, D11, D7, D3, D10, and D4). The initial clusters are (D12, D7, and D8), (D4, D2, D5, D1, and D3) and (D6, D11, D10, D9). Data objects from these clusters are reordered by the centric sorting algorithm. The new sequence is as follows: D8, D2, D6, D9, D7, D1, D10, D12, D5, D11, D4, and D3. As expected, data objects are selected from different groups by taking turns.

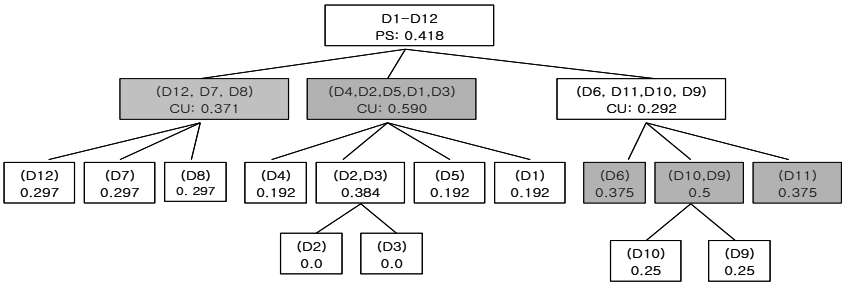

Figure 3. Initial clustering

From the new data sequence, the same procedure is repeated. The newly generated hierarchy by REIT is shown in Figure 4. The final result 4 indicates that the number of clusters is three, where the first and third groups are deflated balloons and the middle cluster consists of inflated balloons. The hierarchy has a different structure from the previous one. While nodes in the middle part of the hierarchy remain the same, the nodes on both sides have changes in the structure.

Unlike the previous clustering, the fragmented clusters are removed from the final partitioning. We can notice that the only difference between the two clustering hierarchies is the placement of D6, which has been moved from the third partition to the first partition, and the resultant change is much greater. While the partitions are collected from two different levels in the first clustering, the partitions from the second clustering are from the first level. The number of clusters is significantly reduced from five to three.

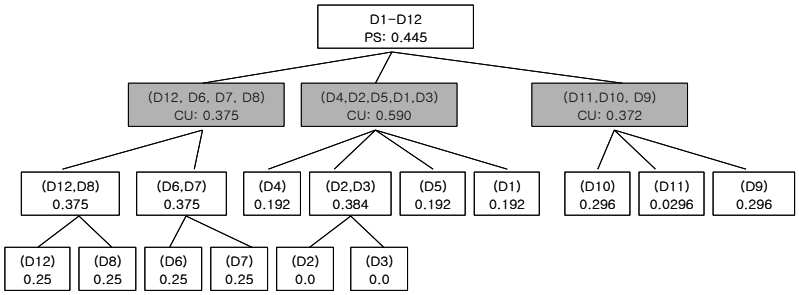

Figure 4. Clustering after centric sorting

Cluster 1 in Table 2 is well represented by the Size attribute, i.e., if the Size is Large, then the balloons can be determined as deflated. On the other hand, Cluster 2 and Cluster 3 are easily discriminated by the Color attribute.

Both clusters share the Size attribute (Small). The above example run shows that the different clustering hierarchies are resulted from different data orderings. The quality of clustering is clearly better in the latter clustering hierarchy. The CU values in the first level of the second clustering are greater than those in the previous hierarchy. The partition score of the first clustering, 0.383, is compared with 0.445 in the second clustering. This indicates that there is another ordering that leads to better clustering. We have shown that a new data sequence by the centric data sorting yields a better clustering for the example run in terms of partition score.

Table 2. Cluster Descriptions

cluster	# of objects	Data objects	Description $P(A_i = V_{ij}\|C_k)$
Cluster 1 (C1)	4	D12, D6, D7, D8	Class = F: 1.0 Color = Yellow: 0.75, Purple: 0.25 Size = Large: 1.0 Act = Stretch: 0.25, Dip: 0.75 Age = Adult: 0.5, Child: 0.5
Cluster 2 (C2)	5	D4, D2, D5, D1, D3	Class = T: 1.0 Color = Yellow: 1.0 Size = Small: 1.0 Act = Stretch: 0.6, Dip: 0.4 Age = Adult: 0.4, Child: 0.6
Cluster 3 (C3)	3	D11, D10, D9	Class = F: 1.0 Color = Purple: 1.0 Size = Small: 1.0 Act = Stretch: 0.67, Dip: 0.23 Age = Adult: 0.67, Child: 0.23

5. CONCLUSION

To mitigate the effect of order bias in clustering, a new efficient algorithm has been developed and its performance has been validated through the experiment. ITERATE is criticized by the fact that pre-ordering is an expensive procedure and that it is not always proven to produce better clustering. Even though it is not possible to completely avoid bias effect caused by specific data sequence, a method is presented in the paper that finds a way to reduce the effect of order bias. There are many different set of data ordering. The results from different sequence of data ordering can be compared with respect to a certain criterion which is defined as partition score. The partition score (PS) is computed as the average category utility of the clusters. We have shown that there would be a better data ordering than maximally differentiated pre-ordering. The better ordering can be obtained by applying the centric sorting to the initial clustering. We have also shown that the current evaluation measure, partition score, is found to be biased toward favoring smaller number of clusters and thus larger size of clusters. This could be problematic when the size of the data set is large, which is the case where more partitions are likely to happen. We need to investigate further how consistently the partition score measures the quality of clustering in relation to the size of the data set.

REFERENCES

[1] I.H. Witten, E. Frank, M Hall, and C. Palestro, "Data Mining: Practical Machine Learning Tools and Techniques," Elsevier Science & Technology, pp. 9-33, 2017.

[2] T. Mitchell, *Machine Learning*, McGraw-Hill Education, 1997.

[3] P. Berkhin, "A Survey of Clustering Data Mining Techniques," Grouping Multidimensional Data, Springer, Berlin, Heidelberg, pp. 25-71, 2006. DOI:https://doi.org/10.1007/3-540-28349-8_2

[4] U. Luxburg, "Clustering Stability: An Overview," *Foundations and Trends in Machine Learning*," Vol. 2, No. 3, pp. 235-274, 2010, DOI:10.1561/2200000008

[5] M.B. Zafar, I. Valera, M.G. Rodriguez, and K. Gummadi, "Fairness Constraints: Mechanisms for Fair Classification,"

[6] M. Hildebrandt, “Preregistration of Machine Learning Research Design. Against P-hacking,” in Being Profiled: Cogitas Ergo Sum, ed. E. Bayamlioglu, I. Baraliuc, L. Janssens, and M. Hildebrandt, Amsterdam University Press, 2018.

[7] D. Fisher, “Knowledge acquisition via incremental conceptual clustering”, *Machine Learning* Vol.2, pp. 139-172, 1987. DOI: https://doi.org/10.1007/BF00114265

[8] V. Kanageswari and A.Pethalakshmi, "A Novel Approach of Clustering Using COBWEB". *International Journal of Information Technology (IJIT),* Vol. 3 No. 3, pp 37-42, Jun 2017. DOI: https://doi.org/10.33144/24545414

[9] D. Fisher, “Iterative Optimization and Simplification of Hierarchical Clustering,” *Journal of AI Research*, Vol.4, pp. 147-179, 1996. DOI: https://doi.org/10.1613/jair.276

[10] F. Cao, J. Liang, and L. Bai, “A New Initialization Method for Categorical Data Clustering,” *Expert Systems with Applications*, Vol 35, Issue 7, pp. 10223-10228, Sep. 2009, DOI: https://doi.org/10.1016/j.eswa.2009.01.060

[11] G. Biswas, J.B. Weinberg, and D. Fisher, “ITERATE: A Conceptual Clustering Algorithm for Data Mining,” IEEE Tr. on Systems, Man and Cybernetics, Vol. 28, Part C No. 2. 1998. DOI: https://doi.org/10.1109/5326.669556

[12] UCI repository, https://archive.ics.uci.edu