Data Dependent Dissimilarity Computation in Dynamic Datasets

ABSTRACT

The data dependent dissimilarity measure provides a better dissimilarity measure between the data points in a dataset compared to their geometric distance in case of algorithms in clustering, classification and anomaly detection. In these algorithms, computing data dependent dissimilarity values takes major computation time. In case of on-line setting where the size of data keeps on increasing, computing dissimilarity value between each pair from scratch is an expensive task in terms of time. In this paper, we present two methods to update dissimilarity value between each pair of data points efficiently under dynamic addition of a few data points. The updated dissimilarity values in the first method can be used by any algorithm in clustering, anomaly detection or multi-label classification. However, the second method is more efficient in terms of computation time but the updated dissimilarity values can only be used by algorithms in clustering.

KEYWORDS

Dissimilarity Measure, Incremental Algorithms, Clustering

ACM Reference Format:
. 1997. Data Dependent Dissimilarity Computation in Dynamic Datasets. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Dissimilarity measure between the data points in a dataset is a core component of any data mining algorithm. Geometrical distance based dissimilarity measure is often employed based on assumption that two objects which have less distance between them in the geometric model are more similar. Geometric models do not take context and nature of the data into consideration for calculating dissimilarity which is a key property of dissimilarity measure [3]. Research suggests that the shortcomings of distance-based measure may be overcome if spatial density is used along with interpoint distance i.e two instances with the same interpoint distance in a dense region is less similar than in a sparse region [1, 3]. In fig 1, black and blue colored points are the initial data points while dynamically added data points are indicated by green color. Two data points a and b in scenario 1 (sparse region) is more similar compared to the data points c and d in scenario 2 (dense region) with same inter-point distance. In [5], the author gives a data dependent dissimilarity measure and shows that it is a better dissimilarity measure than the standard distance-based measure for different algorithms. We represent dissimilarity value between each pair of data points by a matrix called mass matrix.

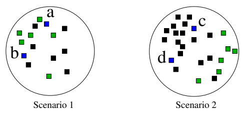


Figure 1: Incremental Scenario

In this paper, we present the incremental version of MBSCAN [5] algorithm that uses mass based dissimilarity measure in place of geometrical distance based dissimilarity measure. Under dynamic environment, the euclidean distance between any two points does not change. But, the dissimilarity value between the same two points may change due to change in density of the smallest region containing the two points as shown in fig. 1. In Scenario 1 of fig. 1, the density of the smallest region containing data points a and b changes under dynamic addition of a few data points (indicated by green color) and thus pair (a, b) is the affected pair whose mass value must be updated. But in scenario 2, the density of the smallest region containing data points c and d does not change and thus the mass value for this pair remain same under dynamic addition of few data points. Updating mass matrix using the straightforward method recomputes the mass value of unaffected pairs as well. This re-computation is redundant. In our proposed algorithm, we update the mass value of affected pairs of data points only. We first present a method to update mass matrix for the general case under dynamic addition of a few data points. The updated mass matrix can easily be used by any algorithm in clustering, multi-label classification, and anomaly detection. We then present a more efficient method to update mass matrix which can only be used for discovering clusters. We show that the proposed algorithm is much efficient in terms of computation time without compromising with accuracy compared to straightforward approach under dynamic addition of a few data points.

2 PRELIMINARIES

The generic name to data dependent dissimilarity measure is *mass based dissimilarity measure.* In this section, we discuss the straightforward approach [5] to compute mass based dissimilarity value of each pair in given dataset D of size n. The flowchart shown in fig. 2 gives the overall architecture to construct *mass matrix.* The mass matrix computation is done in three phases:

Phase 1: iForest Creation In this phase, we first construct an *iForest* (isolation forest) [4] which is a set of *itrees* (isolation trees). The number of *itrees*, $numTree$, to be built is a user parameter. Each *itree* is build independently using the sample of data points chosen randomly from the dataset D. The sample size is an another user parameter denoted by ψ. Each *itree* is basically a binary tree. The maximum height of each *itree* is $\log \psi$ and denoted by $maxHeight$. For each *itree*, we generate samples of size ψ from the dataset randomly and insert them into the $iTree$. For each node in the $iTree$

WOODSTOCK'97, July 1997, El Paso, Texas USA
© 2016 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

we choose a *split attribute* randomly and then randomly choose a *split point* from the range of minimum and maximum value of chosen *split attribute*. We then split the data points in the node to its child based on the *split attribute* and *split point* value. At particular node, we stop splitting if any of the following conditions are met:

- Node has only one instances left.
- Height of node reaches $maxHeight$.

We call these nodes *leaf nodes* for given $itree$. In this phase we record *split attribute* and *split point* of each node of each $iTree$ that gets split. These information are used to construct some generic arrays in phase 2 that further are used to construct mass matrix in phase 3.

Phase 2: Generic Array Update In this phase, we insert whole data points of size n into each $itree$ constructed in previous phase. Using the *split attribute* and *split point* value of each node of $itrees$ computed in previous phase, we partition the data points into its children. We stop splitting data points at some node if that node is a *leaf node*. We record the number of data points in each node of each $itree$ and denote it by $treeNodeMass$. We observe that each data point lies in exactly one *leaf node*. We record the leaf node corresponding to each data points in D for each $iTree$ and denote it by $treeNode$.

Phase 3: Computing Mass Matrix As per the definition in [5], mass of two data points is the number of data points lying in the smallest region (*node*) containing the two data points. The smallest region containing the data points i and j is the least common ancestor node, k_t, of nodes *treeNode(i)* and *treeNode(j)* where *treeNode(x)* denotes the leaf node where data point x lies in $t^{th} iTree$. The mass value of i and j denoted by $m(i, j)$ is computed using equation 1.

$$m(i, j) = \frac{\Sigma_{t=1}^{numTree} treeNodeMass(k_t)}{numTree} \quad (1)$$

Where, *treeNodeMass(k_t)* denotes the number of data points in node k_t for given $t^{th} itree$. We record the mass value of each pair of data points in dataset D using equation 1 and call it mass matrix.

In [5], the author uses this mass matrix in place of distance matrix in standard algorithms of clustering, multi-label classification and anomaly detection. The method to discover cluster using mass based dissimilarity measure MBSCAN is same as distance based DBSCAN [2] with distance matrix replaced by mass matrix. The parameter μ in MBSCAN play the same role as ϵ in DBSCAN. In

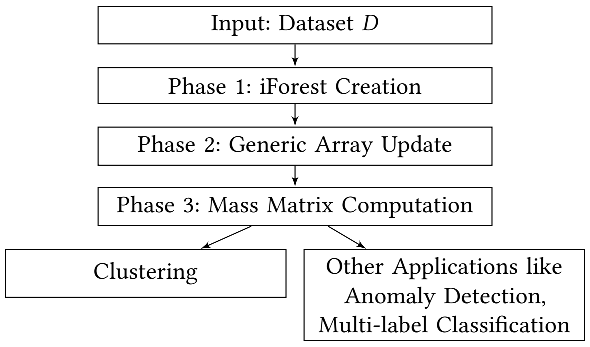

Input: Dataset D
Phase 1: iForest Creation
Phase 2: Generic Array Update
Phase 3: Mass Matrix Computation
Clustering
Other Applications like Anomaly Detection, Multi-label Classification

Figure 2: Architecture of Mass Based Dissimilarity Measure

next section, we describe our proposed incremental algorithm for updating *mass matrix* for general case under dynamic addition of a few data points.

3 PROPOSED ALGORITHM 1

Under dynamic addition of a few data points, instead of computing whole mass matrix from scratch, computing it efficiently using the information available from previous run is the main goal of the incremental algorithm. In this section, we discuss the incremental algorithm that updates mass mass matrix efficiently under dynamic addition of a few data points. The updated mass matrix can be used by any algorithm in clustering, anomaly detection and multi-label classification. Fig. 7 gives the overall idea of updating mass matrix under dynamic addition of data points. We update mass matrix incrementally in two phases described below.

Phase 1: Incremental iForest Update Before addition, we have an $iForest$ created from dataset of size n. We assume that the size of new data points, $n^{'}$, being added is very small compared to n. So, this new data points will not affect the spatial density of data points much. Since, $iForest$ is the representation of the data points in D. Hence, we do not change the existing structure of $iForest$. Thus, at each node of each $itree$, we leave the *split attribute* and *split point* value as it is. This $itrees$ are constructed from the sample of size ψ which was generated from the original dataset D of size n.

In this phase, we select same proportion of data points from new set of data points. We insert these new samples into each $itree$. We partition these sample points into its left or right child based on the previously chosen *split attribute* and *split point*. Every points from these samples definitely reaches to the leaf node of each $itree$. If the leaf node is at its maximum height, we do nothing. A leaf node whose height is less than the $maxHeight$ contains a single old data points. After insertion of new samples, if new data points enters in such leaf nodes then we merge that old single point with these new sample points. We choose *split attribute* and *split point* randomly for such nodes as we did in straightforward approach. We then partition the sample data points into its left and right child accordingly. We stop splitting if it meets the stopping criterion defined in previous section. In this phase, we iterated each $itree$ using the sample points only from new set of data points of size $n^{'}$ which otherwise would have used samples from whole updated data points of size $n + n^{'}$. We use this updated *iForest* in next phase.

Phase 2: Incremental Generic Array Update For each $itree$, we first insert all the old data points together with the new data points into root node. We follow the partition of points as usual using the *split attribute* and *split point*. From previous run, we already have *treeNodeMass* and *treeNode*. At any node, if number of data points is not equal to the number of data points stored in $treeNodeMass$, we update mass of each pair whose least common ancestor is this current node. We observe that the current node is the least common ancestor node of all pairwise elements in its left child and right child. We update the mass value of pairwise element in its child node. Note that the mass value of a pair is averaged over all $itrees$ in previous run. For any pair, we get the old mass contribution from the current $itree$ using the old $treeNode$ and old $treeNodeMass$. New mass contribution from the current $itree$ is

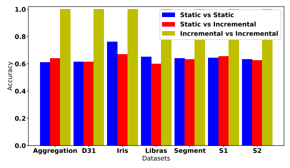


Figure 3: Accuracy

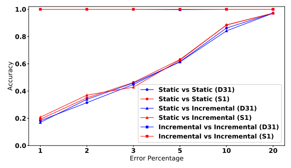


Figure 4: Error Vs Accuracy

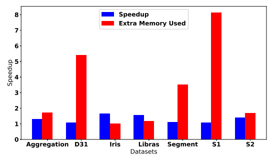


Figure 5: Speedup

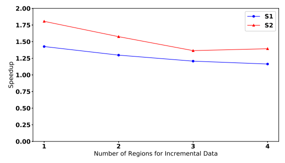


Figure 6: Region Vs Speedup

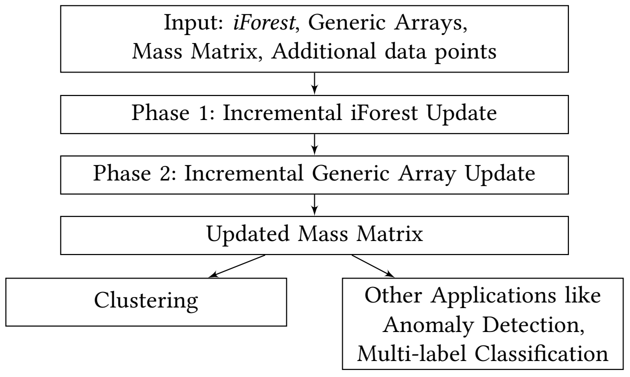
the updated number of data points in current node. We divide both old and new contribution by *numTree*. We get the final mass value by subtracting old contribution and adding new contribution to the old mass value. The old mass value for a pair having new data points is zero. For such pairs, we only add the new contribution to compute their mass.

We stop partitioning data points further down to a node if number of data points after split is equal to the previous value in *treeNodeMass*. In this case, no new data points entered into this node and thus the mass value of each pair having this node or nodes in its subtree as their least common ancestor node does not get change.This way we update mass value of all affected pairs due to dynamic addition of new data points from one *itree*. We do this for all *itrees*. We get the updated mass matrix in this phase itself.

4 RESULTS OF ALGO 1

In this section we present experimental results for the algorithm described in Section 3. We use $\psi = 256$ and $numTree = 100$ as parameters for both static and incremental algorithm. We evaluate our algorithm on 5 real world and 2 synthetic datasets as described in table 1.

$S1$ and $S2$ are synthetic datasets. $S1$ has 10000 data points having 4 Gaussian clusters with mean at coordinates $(150, 150)$, $(150, -150)$, $(-150, 150)$, and $(-150, -150)$ with different standard deviation of each cluster. $S2$ has 1000 data points having 4 Gaussian clusters with mean located at coordinates $(80, 80)$, $(80, -80)$, $(-80, 80)$, and $(-80, -80)$ with different standard deviation of each cluster. We use Jaccard similarity technique as a measure of accuracy. Due to randomization involved in creating *iForest*, we report the results averaged over 10 runs. Across multiple runs, a pair of points may have different mass values. If the difference in the mass values


Input: iForest, Generic Arrays, Mass Matrix, Additional data points
Phase 1: Incremental iForest Update
Phase 2: Incremental Generic Array Update
Updated Mass Matrix
Clustering
Other Applications like Anomaly Detection, Multi-label Classification
Figure 7: Flow Diagram of Algorithm 1

Dataset	Size	Dimensions	Clusters
Aggregation	788	2	7
D31	3100	2	31
Iris	150	4	3
Libras	360	90	15
Segment	2310	19	7
S1	1000	2	4
S2	10000	2	4

Table 1: Properties of different datasets used

across multiple runs is less than a certain percentage, termed as *error percentage*, we take it as accurate. The default *error percentage* and *incremental percentage* are 5 % for both.

Fig.3 shows the accuracy of updated mass matrix on multiple runs in static and incremental algorithms for different datasets. We see that the incremental algorithm has accuracy similar to static algorithm. It is worth noting that the multiple runs of incremental algorithm gives consistent results. If we increase the error percentage the accuracy of mass matrix on multiple runs by static as well as incremental algorithm increases in-tandem and reaches close to 1 at 20 percent as shown in fig. 4. Fig. 5 shows the speedup (incremental time/ static time) for different datasets. The data points added dynamically for this experiment belongs to all region uniformly. We observe that the incremental algorithm is always better than the static algorithm. The save in computation time costs more memory as indicated in the figure. Fig. 6 analyzes a special case in which dynamically added points are coming from a small region, vis. from one, two, three, or four region. We observe that for data points coming from small region, the speedup is large.

5 PROPOSED ALGORITHM 2

In this section, we describe a more efficient method to update mass matrix under dynamic addition of a few data points. This updated mass matrix can only be used for algorithms in clustering.

Preprocessing: In this preprocessing step, we make a list of all possible pairs of data points that may get affected due to addition of new data points of size n'. Let x be the mass value of any pair in original dataset. Under dynamic addition, a pair whose normalized value was above μ can have updated normalized value below μ if no new points gets inserted in the minimum region where this pair lies, i.e.,:

$$\frac{x}{n + n'} \leq \mu \Rightarrow x \leq \mu * (n + n') \qquad (2)$$

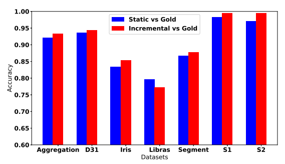


Figure 8: Accuracy

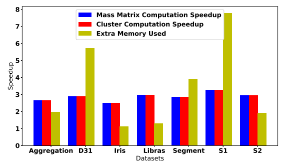


Figure 9: Speedup

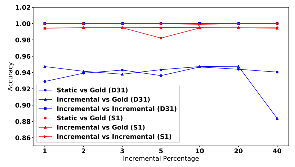


Figure 10: Inc % Accuracy

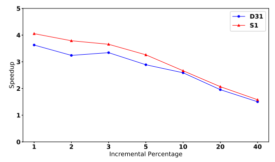


Figure 11: Inc % Vs Speedup

Where, μ is the parameter for MBSCAN which acts as a radius to find the neighborhood. Thus, we get *upperlimit* as $\mu * (n + n^{'})$. Similarly, a pair whose normalized value was below μ can have updated normalized value above μ if all new points gets inserted into their minimum region, i.e.,:

$$\frac{x + n^{'}}{n + n^{'}} \geq \mu \Rightarrow x \geq \mu * (n + n^{'}) - n^{'} \tag{3}$$

Thus, we get *lowerlimit* as $\mu * (n + n^{'}) - n^{'}$. Hence, we create a list of all pairs from old data points whose mass value lies in this range. After addition of new data points, we just update the mass value of these pairs from old data points. However, we must have to compute the mass value of new pairs formed by old data points with new data points as well as by two new data points using the naive method. Formally, we update mass matrix for clustering in three phases.

Phase 1: Incremental Forest Update Here, we update *iForest* with the same approach as we discussed in our method for general case above.

Phase 2: Incremental Generic Array Update We observe that we have two type of nodes after phase 1. Non-leaf nodes before addition of new points together with the leaf nodes that has not been split in previous phase belongs to first category. Nodes that has been split in previous phase together with the new nodes due to splitting constitutes a second category. In this phase, we only insert new data points of size $n^{'}$ into the root of each *itree*. If current node belongs to first category, the size of node increases by number of new data points in current node. If current node belongs to second category, we first merge the old data points lying in the node with the current set of new data points entered into the node after partitioning. The size of current node will be the size of updated list after merging. We follow the general partitioning method from this node downward with all the merged points. We update *treeNodeMass* accordingly. When we reach leaf node, we update that node as the minimum region node for all its data points in *treeNode*. Thus, in this phase, we have *treeNode* and *treeNodeMass* matrix updated correctly and efficiently.

Phase 3: Incremental Mass Matrix Update With straightforward approach, we update mass value of all pairs in dataset. Instead, we update mass value of pairs belonging to the list we created in our preprocessing step. First, we iterate through each pair in the list and update their mass value using eqn. 1. Under dynamic addition, we have new pairs in the mass matrix. We compute mass value of these pairs using straightforward approach.

6 RESULTS OF ALGO 2

In this section, we present results of incremental mass matrix computation algorithm for clustering. Results are prepared according to experimental methodology described in Section 4. Fig. 8 shows the accuracy of clusters by static and incremental algorithm in comparison with original clusters for different datasets. We observe that the incremental algorithm performs as par with the static algorithm. Proposed incremental algorithm saves computation time significantly. Fig. 9 shows that the incremental algorithm is around 3 times faster compared to the straightforward approach with the use of extra memory. The saving is achieved for all the datasets under observation, showing that the algorithm is applicable for different datasets. The saving achieved in time is not cost free. Our algorithm takes more amount of space than static algorithm, which is indicated in the figure. In fig. 10, we see that the accuracy of clustering by static and incremental algorithm with respect to original clusters is almost same with different percentage of additional data points. With increase in size of dynamically added data points, the time to update mass matrix increases as shown in fig. 11.

7 CONCLUSION

In this paper we proposed an incremental algorithm to update mass based dissimilarity value between each pair of data points under dynamic addition of a few data points. We see that the mass matrix computation takes major time compared to total time taken to discover clusters. The time o update mass matrix for general case is independent on the size of additional data points. It only depends on the size of region in which the new data point belongs.

REFERENCES

[1] S. Aryal, K. M. Ting, G. Haffari, and T. Washio. 2014. Mp-Dissimilarity: A Data Dependent Dissimilarity Measure. In *2014 IEEE International Conference on Data Mining*.

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press.

[3] Carol Krumhansl. 1978. Concerning the applicability of geometric models to similarity data: The interrelationship between similarity and spatial density. (1978).

[4] F. T. Liu, K. M. Ting, and Z. H. Zhou. 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE.

[5] Kai Ming Ting, Ye Zhu, Mark Carman, Yue Zhu, and Zhi-Hua Zhou. 2016. Overcoming Key Weaknesses of Distance-based Neighbourhood Methods Using a Data Dependent Dissimilarity Measure. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.