Incremental Shared Nearest Neighbor Density-Based Clustering Algorithms for Dynamic Datasets

Panthadeep Bhattacharjee PANTHADEEP.EDU@GMAIL.COM
Amit Awekar AWEKAR@IITG.ERNET.IN
Indian Institute of Technology, Guwahati, Assam, India

Abstract

Dynamic datasets undergo frequent changes where small number of data points are added and deleted. Such dynamic datasets are frequently encountered in many real world applications such as search engines and recommender systems. Incremental data mining algorithms process these updates to datasets efficiently to avoid redundant computation. Shared nearest neighbor density based clustering (SNN-DBSCAN) is a widely used clustering algorithm, mainly for its robustness. Existing incremental extension to SNN-DBSCAN cannot handle deletions to dataset and handles insertions only point by point. We overcome both these bottlenecks by efficiently identifying affected parts of clusters while processing updates to dataset in batch mode. We present three different incremental algorithms with varying efficiency at elimination of redundant computation. We show effectiveness of our algorithms by performing experiments on large synthetic as well as real world datasets. Our algorithms are up to 2 Orders of Magnitude faster than non-incremental algorithm and up-to 50 times faster than existing incremental algorithm while guaranteeing exact same output.

1. Introduction

Many popular clustering algorithms work on static snapshot of dataset. However, many real-world applications such as search engines and recommender systems are expected to work over dynamic datasets. Such datasets undergo frequent changes where some data points are added and some data points are deleted. With any change in dataset, the clustering output might change. A naive method to get exact clustering over the changed dataset is to run the clustering algorithm again. However, if the changes in dataset are minor, then change in output is also expected to be small. These changes cannot be ignored as they might be significant for the datapoints and their neighborhood.

Most of the computation in reclustering is going to be redudant. This problem becomes more severe with increase in frequency of updates to dynamic datasets. For large datasets, rerun of the algorithm might not finish before next batch of updates arrive. Incremental algorithms target this fundamental issue of redundant computation yet obtain identical output to non-incremental counterpart.

SNN-DBSCAN is a widely used clustering algorithm for its robustness while finding clusters of varying densities and shapes even in high dimensional data(Ertöz et al., 2003). Existing incremental extension to SNN-DBSCAN cannot handle deletions from the dataset and it handles insertions only one point at a time(Singh & Awekar, 2013). We present three algorithms that overcome both these problems with varying efficiency at elimination of redundant computation.

2. Preliminaries and problem definitions

Incremental Clustering: *Given a data set D alongwith its initial clustering $f : D \rightarrow C$ where $C \subseteq D$ and an insertion or deletion sequence of B batches with 'n' points per batch. After $k \leq nB$ updates(inserts or deletes), such that k exactly divides nB and let D' be the new data set, then an incremental clustering is defined as a mapping $h : f,\ D' \rightarrow C'$, where $C' \subseteq D'$ is isomorphic to the one time clustering f(D') by the non-incremental algorithm.*

Similarity value: For two points p and q It is defined by the following equation:

$$similarity(p, q) = KNN(p) \cap KNN(q) \qquad (1)$$

Core, non-core and noise points: In SNN graph (Ertöz et al., 2003), if the number of strong links associated with a point exceeds a certain threshold, then the point is a core point, otherwise it is a non-core point. All the non-core points which do not share a link with any of the core points are called as noise points.

3. Proposed algorithms

Incremental algorithms for addition: We propose three incremental algorithms for addition: *Batch-Inc1*, *Batch-Inc2* and *Batch-Inc3*. In *Batch-Inc1*, the KNN list of the data points are computed incrementally. Amongst the existing points, the KNN list of only those points are updated which accomodates any of the new points in their updated KNN list. This type of points are categorized as *Type 1* points. The entry of one or more of the new points in the updated KNN list of *Type 1* points displaces some of the existing points from its KNN list. The rest of the points remain unaffected and retain their previous KNN list. The new similarity matrix(Ertöz et al., 2003) is constructed by finding the similarity values between every point $p \in D'$ and each point $q \in$ updated KNN list of p. Only the similarity values which are greater than or equal to a threshold value (*snn*) are accepted, given that p and q lie in each others' KNN lists. The number of non-empty cells in each row of the similarity matrix corresponds to the number of strong links for a point. All points $p \in D'$ that have the number of associated strong links greater than another threshold (*density*) qualify as core points. If two core points are connected by an edge in the SNN graph (Jarvis & Patrick, 1973), then they are placed in the same cluster. A non-core point is assigned to that cluster, which its *nearest* core point is a part of. The *nearest* point is decided by the strength of the shared link. If p, r are core points and q is a non-core point such that *similarity(q,r)* $>$ *similarity(q,p)*, then q is assigned to that cluster which contains r.

Batch-Inc2 computes both the KNN list and similarity matrix incrementally. For constructing similarity matrix, firstly every new point $p \in D'$ computes its similarity value with each point $q \in D'$, such that q lies in p's KNN list. If *similarity(p,q)* $\geq$ *snn*, and p and q lie in each others KNN list, a shared link between them is formed. Next compute the similarity values of each of the *Type 1* points from their respective updated KNN lists. The updated similarity values of the *Type 1* points may differ from corresponding values in the old similarity matrix. This is because the insertion of new data point(s) in the updated KNN list of each of the *Type 1* points forces one or more existing points to move out of the new KNN list. Only those points which are at a distance lesser than or equal to the Kth nearest distance from the affected *Type 1* point retain their place. This results in removal of any link between the *Type 1* point and its previously existing neighbors. The removed non-*Type 1* data points are labelled as *Type 2* affected points. Amongst the points which retain their place in the updated KNN list, all the non-*Type 1* points are also labelled as *Type 2* points. Next, compute the similarity values of each of the *Type 2* points. Although the KNN list of the *Type 2* points remain unaffected, these points may change their cluster membership due to possible presence of *Type 1* point(s) in their KNN list. The points which are niether *Type 1* nor *Type 2* retain their existing similarity values. The algorithm follows with the subsequent creation of core points and cluster formation similar to *Batch-Inc1*.

Batch-Inc3 computes all the properties: KNN list, similarity values and core/non-core property incrementally. The KNN lists and similarity matrix are computed similar to *Batch-Inc1* and *Batch-Inc2*. In *Batch-Inc3*, the *Type 1* and the *Type 2* affected points are categorized into core or non-core points based on the state of updated similarity matrix. The unaffected points retain their previous core or non-core property. The updated KNN list, similarity matrix, core and non-core points are used to find clusters incrementally for the next batch of incoming points.

Incremental algorithms for deletion: For deletion of data, we present three incremental clustering algorithms: *Batch-Dec1*, *Batch-Dec2* and *Batch-Dec3*. Although this class of algorithms compute the KNN list, similarity matrix and core point labelling incrementally, the computation method is fundamentally different from the incremental *addition* algorithms.

Batch-Dec1 computes the KNN list of the data points incrementally. Removal of points directly affects the KNN list of some of the existing data points. The existing KNN list of such affected points may contain one or more of the deleted points. Such points are categorized as *Type 1* points. The deleted points are removed from the KNN list of the *Type 1* affected points. As a result the size of the nearest neighbor list shrinks. However, the algorithm priorly maintains an extended KNN list for each data point with an additional space for (w-1)*K points where $w \in \mathbf{N}$. The points in the extended KNN list are placed in order of their increasing distance from the affected *Type 1* point. If $d \leq$ K points are removed from the existing KNN list within the boundary of K nearest points, then an additional d number of points shift that many places to form the updated KNN list. The formation of similarity matrix, core and non-core points and clusters are similar to *Batch-Inc1* of the *addition* algorithms.

Batch-Dec2 computes both the KNN list and similarity matrix incrementally. For constructing similarity matrix, firstly for every *Type 1* point $p \in D'$, the algorithm computes its similarity value with point $q \in D'$, such that q lies in the extended KNN list of p and q can at most be the Kth nearest point from p. The updated similarity values of the *Type 1* points may differ from corresponding values in the existing similarity matrix. This is because the deletion of data point(s) from the updated KNN list of each of the *Type 1* points forces one or more of the existing points from the extended *(w-1)**K space to enter within the distance of Kth nearest point from the respective *Type 1* point. The new points which enter the space within boundary of Kth nearest point from the *Type 1* point are labelled as *Type 2* points given that these points are non-*Type 1* points. As a result, a possible *Type 1-Type 2* link formation may take place. All the non-*Type 1* points which retain their place in the updated KNN list of the *Type 1* points, are also labelled as *Type 2* points. The shared link strength between such *Type 2* points and the *Type 1* point might increase with newly entered points becoming the contributory factor. Next, compute the similarity values of each of the *Type 2* points. Although the KNN list of the *Type 2* points remain unaffected, these points may change their cluster membership due to possible presence of *Type 1* point(s) in their KNN list.

Batch-Dec3 computes all the properties: KNN list, similarity values and core/non-core property incrementally. The KNN lists and similarity matrix are computed similar to *Batch-Dec1* and *Batch-Dec2*. In *Batch-Dec3*, the *Type 1* and the *Type 2* affected points are categorized into core or non-core points based on the state of updated similarity matrix. The unaffected points retain their previous core or non-core property.

4. Experimental results

First, we illustrate the superiority of our incremental algorithms over the non-incremental SNN-DBSCAN algorithm (Ertöz et al., 2003) across the three datasets: Mopsi2012 location based search engine, 5D synthetic dataset, Birch3 dataset. For the sake of our experiments, we define a new term called *algorithm components*. It consists of base dataset, KNN list, similarity matrix, core, non-core points and clusters based on the base dataset. In tables 1 and 2 we highlight the comparisons of our incremental algorithms with non-incremental SNN-DBSCAN for both addition and deletion of points.

Comparisons with pointwise insertion based incremental IncSNN-DBSCAN:
For comparing our algorithms with the IncSNN-DBSCAN algorithm, we performed experiments on Mopsi2012 and Birch3 dataset.
Mopsi2012: The initial parameters of the three incremental *addition* algorithms taken were: *knn*:10, *snn*:3, *density*:4. We inserted 3000 points upon the base dataset of 10000 points in multiple batches. The minimum number of added points/batch were 2, and we carried on the experiments till we added 50 points/batch. For a batch containing 2 points we ran each of the three algorithms independently 1500 times to insert 3000 points. The final set of clusters that were produced after 1500 batches were identical to the IncSNN-DBSCAN. For deletion, the algorithm parameters remain the same and w is chosen as 2. We set the *algorithm components* values by performing SNN-DBSCAN over 13000 points and then deleted 3000 points in multiple batches incrementally. Similar to addition, we started with a minimum of 2 points/batch to be deleted, and carried on the experiments upto 20 points/batch. *Batch-Inc3* achieved the highest speedup of 50 times and is about 98% faster than IncSNN-DBSCAN when 50 points/batch were inserted. Out of the three *deletion* algorithms *Batch-Dec3* was the most efficient . *Batch-Dec3* was about 81.77% speedier than its pointwise measure when it performed point wise deletion removing upto 20 points/batch. A maximum of around 0.30% of *Type1* and about 0.69% of *Type 2* affected points were identified while processing batch number 1229 and 898.

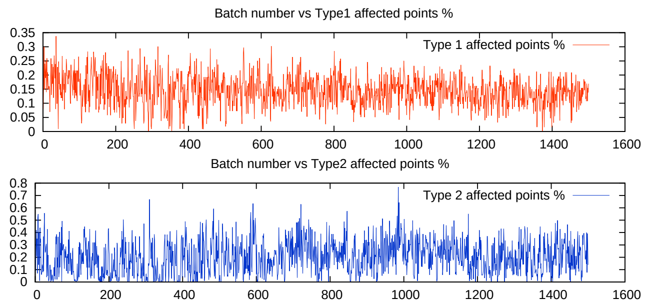

Figure 1. Type 1 and Type 2 affected points during addition for 1500 batches for Mopsi2012.

Table 1. Performance comparsion with SNN-DBSCAN for addition of data

Dataset	Size	Base dataset size	Added	SNN-DBSCAN(sec)	*Batch-Inc1* Speedup %	*Batch-Inc2* Speedup %	*Batch-Inc3* Speedup %
Mopsi2012	13000	10000	3000	128.95	77.52	78.06	78.94
5D points set	100000	80000	20000	7405.87	82.71	83.01	83.67
Birch3	100000	80000	20000	5876.93	82.39	82.83	83

Table 2. Performance comparsion with SNN-DBSCAN for deletion of data

Dataset	Base dataset size	Deleted	Remaining	SNN-DBSCAN(sec)	*Batch-Dec1* Speedup %	*Batch-Dec2* Speedup %	*Batch-Dec3* Speedup %
Mopsi2012	13000	3000	10000	81.19	29.65	30.27	31.02
5D points set	100000	20000	80000	4702.32	63.42	63.76	63.99
Birch3	100000	10000	90000	4874.74	99.52	99.54	99.55

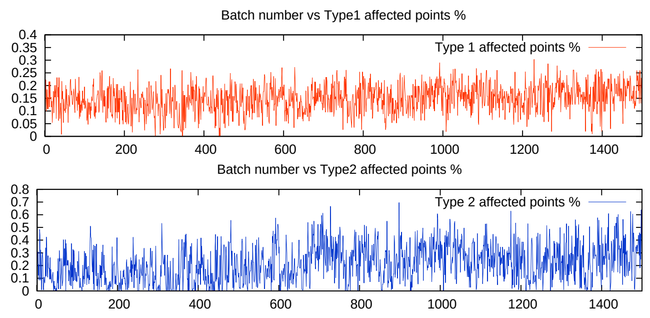

Figure 2. Type 1 and Type 2 affected points during deletion for 1500 batches for Mopsi2012.

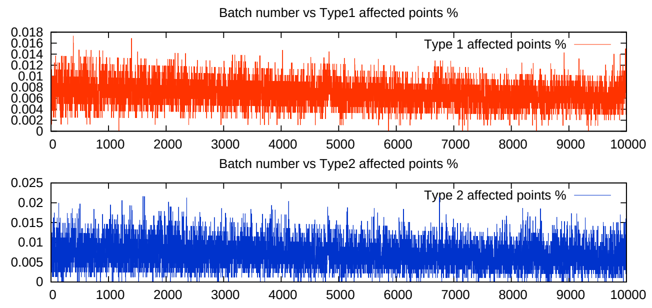

Figure 3. Type 1 and Type 2 affected points during addition for 10000 batches for Birch3 dataset.

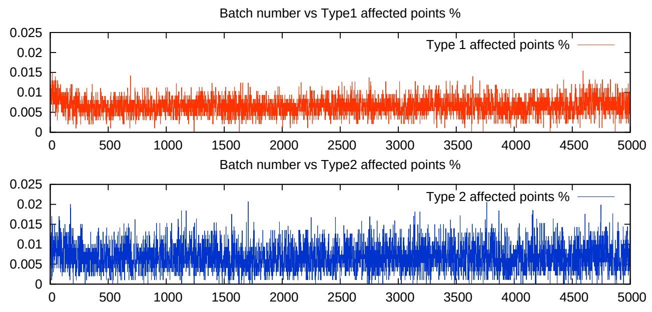

Figure 4. Type 1 and Type 2 affected points during deletion for 5000 batches for Birch3 dataset.

Birch3: The parameter values taken were *knn*:5, *snn*:2, *density*:2. The base dataset size was 80000 points and 20000 points were inserted upon it. The *algorithm components* were computed up based on the base dataset.We started with 2 added points/batch and continued till 50 points/batch were added. On entering upto 50 points/batch *Batch-Inc3* showed speedup of 40.67 times and was quicker than IncSNN-DBSCAN by about 97.50% For 10000 batches, maximum of about 0.0173% of *Type1* and about 0.0216% of *Type 2* affected points were identified(figure) while processing batch number 386 and 1599. Retaining same parameters with *w* is 2 we deleted 10000 points with 5000 batches and 2 points/batch from the base dataset size of 100000 points. *Batch-Dec3* was the most efficient method and was about 80.03% faster than its points wise measure when 20 points/batch were deleted. A maximum of around 0.0154% of *Type1* and about 0.0207% of *Type 2* affected points were identified while processing batch number 4593 and 1707 respectively.

References

Ertöz, Levent, Steinbach, Michael, and Kumar, Vipin. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *SIAM SDM*, pp. 47–58, 2003. doi: 10.1137/1.9781611972733.5.

Jarvis, R. A. and Patrick, E. A. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, C-22(11): Nov 1973. ISSN 0018-9340. pp. 1025–1034, doi: 10.1109/T-C.1973.223640.

Singh, Sumeet and Awekar, Amit. Incremental shared nearest neighbor density-based clustering. In *CIKM*, pp. 1533–1536, 2013.