A Density-Biased Sampling Technique to Improve Cluster Representativeness

Ana Paula Appel, Adriano A. Paterlini, Elaine P. M. de Sousa,
Agma J. M. Traina, and Caetano Traina Jr.*

Computer Science Department - ICMC
University of São Paulo at São Carlos – Brazil
{anaappel@,adriano@grad.,parros@,agma@,caetano@}icmc.usp.br

Abstract. The volume and complexity of data collected by modern applications has grown significantly, leading to increasingly costly operations for both data manipulation and analysis. Sampling is an useful technique to support manager a more sensible volume in the data reduction process. Uniform sampling has been widely used but, in datasets exhibiting skewed cluster distribution, biased sampling shows better results. This paper presents the *BBS - Biased Box Sampling* algorithm which aims at keeping the skewed tendency of the clusters from the original data. We also present experimental results obtained with the proposed BBS algorithm.

1 Introduction

Data reduction and sampling techniques have been employed to speed up data mining algorithms, and the more representative a dataset sample, the better the results obtained. Many of the data reduction techniques for multi-dimensional data rely on uniform sampling. However, several tasks have to deal with non-uniform data distribution, in particular clustering activities when the original clusters have distinct properties among themselves, such as the number of elements and/or the density. In such cases, density-biased sampling can provide better results, as the probability of a point to be added to the sample depends on the local density of its neighborhood.

Figure 1 shows nine clusters over uniform noise (fifteen percent), one containing 50,000 points and the others containing 1000 points. Extracting a 1% uniform sampling will produce a sample dataset containing one cluster with roughly five hundred points, and eight containg around ten points each as well as some noise. Since the number of clusters will be discovered only after the clustering has been finished, this information is not available to the sampling process. Therefore, in this example the clustering algorithm will not be able to spot the small cluster. The problem here is that when the representation of a cluster in the dataset is significantly lower than those of the other clusters, the clustering algorithm may miss the small clusters, mixing it with noise. Therefore, the question posed is:

* The authors thank CNPq, Capes and FAPESP for the financial support.

J.N. Kok et al. (Eds.): PKDD 2007, LNAI 4702, pp. 366–373, 2007.

"How to sample a multi-dimensional dataset without missing the clusters, even if they are unbalanced (that is, their number of points are quite different) and without any previous knowledge about the clusters?".

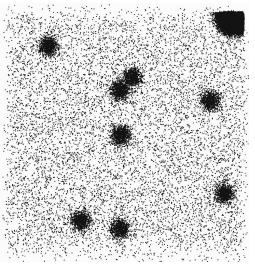

Fig. 1. A 20-dimensional dataset comprising 1 cluster with 50,000 points, 8 with 1000 points and 15% of noise

The problem of uniform sampling over skewed data was first treated in [1]. The authors divided the space into equal-sized cells, storing the points in a set of hash table. Thereafter, the cells with few points are oversampled and the cells with many points are under sampled. In [2], the hash table is substituted by uniform sampling to avoid collision problems. Another density-biased sampling technique was proposed in [3], which samples points according to the local density near each point. The authors use kernel-density based methods to estimate the local density. However, these functions cause a significant time overhead. All of the previously discussed techniques are sensitive to noise and dimensionality. In [4] a density-biased sampling method based on R-Trees that can do sampling in noisy datasets is presented. However, due to the R-Tree shortcomings for high-dimensional data, the authors advised that the technique aimed at sampling data in a Database Management System (DBMS) indexed by a spacial index, and it is also sensitive to the data dimensionality.

We present a new technique for sampling based on local density. It is less sensitive to noise and high dimensionality problems than the most of existing techniques. We also presents the **BBS - Biased Box Sampling** algorithm [5], that implements our technique with linear cost $O(N \cdot E)$ on the number N of dataset points and on the number of attributes E, and can also be integrated into a DBMS. The experimental results show that, even at a lower sampling rate, it can generate samples that allow clustering processes to find clusters more accurately. The remainder of the paper is organized as follows. Section 2 discusses the technique and the algorithm developed. Section 3 discusses experimental studies. Section 4 concludes the paper.

2 Biased-Box Sampling

This section details our main contribution: a technique to sample multi-dimensional datasets to reduce the amount of data to be submitted to clustering

processes based on a multi-resolution mapping of the data space, aiming at reducing the computational cost of such processes. We also present the BBS algorithm (**Biased Box Sampling**) based on this technique, which enables clustering processes to find clusters retaining their accuracy even at very low sampling rates.

The main idea of the proposed technique is to divide the data space into 2 regions, so that each attribute splits the space by half, and counting the number of points of the dataset that lies at each region. Each region holding more than a given threshold δ of points is recursively divided, generating a "hyper-quad tree" like structure, the compact multi-resolution grid tree - *MGc-tree*. Whenever a region does not hold the given threshold, it is represented in a leaf node and its points are sampled. The leafs can be at different levels, reflecting the density of the space at each region, so performing the sampling at a constant rate at every leaf will in fact retrieve more points from less dense regions, reflecting accurately the density variation over the full data space.

The BBS algorithm has three parts. The first one creates the multi-level grid, implemented as a tree. It is similar to the LiBOC algorithm [6], the main modification being the need to store the points at each leaf node. As this new requirement has a computational cost constant for each point, this step has the same complexity as LiBOC, which is $O(N \cdot E)$. The resulting structure is a tree (the *MG-tree*) where each non-leaf node stores a counter of the number of points lying on its corresponding region, an identification of the region, and from zero up to 2^E pointers to the next higher-resolution regions that divide the current one. A leaf node only stores the pointers for the data points lying on the corresponding region. Notice that at this step, the tree has every leaf node at the highest resolution $R - 1$. The value R defines the maximum number of resolutions the algorithm must try to obtain a good sample. Its minimum is bounded by how good the sampling must be, and its maximum is bounded by the highest density of the sampled dataset. If R is set too high, the algorithm will require more memory to operate (as the complexity of the memory required by the algorithm is $O(N \cdot E \cdot R)$), but after a threshold it will not improve the sampling anymore. Experimentally we observed that $R = 5$ is suitable for all the datasets evaluated. The first part of BBS is shown as Algorithm 1.

The second part of the BBS algorithm aims at reducing the deep of the multi-resolution grid tree at the less dense regions, transforming the MG-tree generated by Algorithm 1 into a condensed *MGc-tree*. This part, shown as Algorithm 2, looks for cells in the tree where the number of points is lower than the threshold δ. When such cells are found, the index lists from the children cells are concatenated, transforming their parent into a leaf-node. Thus, the resulting condensed *MGc-tree* has leaf nodes of approximately the same number of points.

The threshold δ is set $\delta = E * 100/(2 * Ratio)$, and is evaluated in Step 5 of Algorithm 2. The concatenation of points in Step 7 generates varying number of elements at each leaf node. To assure none of them are under-represented we double δ (using $2 * Ratio$). The next step is to perform the sampling. The third (and last) part of the BBS algorithm effectively performs the multi-resolution sampling. It retrieves each leaf node of the *MGc-tree* and the difference between

the level of the node and the maximum level of the tree is used to increase the number of points selected, performing another over sampling in lower resolution regions (Step 7 of Algorithm 3). Step 8 uses the concatenated list to choose points for the sample.

The complete BBS algorithm shown as Algorithm 4 just calls the three parts. It receives the set of points to be sampled and creates an array indexing the sampled points of the original dataset (Step 4). The δ employed in Algorithm 2 and Step 7 of Algorithm 3 contributes to enlarge the number of points in the sample. Therefore, Step 4 of Algorithm 4 generates the final sample, randomly dropping the points selected until the desired number is achieved.

Algorithm 1. Biased Box Sampling: Create Tree - BBSCT

Input: Dataset with N points with the E attributes, number of levels R
Output: Multi-resolution grid Tree - MG-tree
1: Normalize the dataset points to a unit cube;
2: **for** each point t in dataset **do**
3: **for** $r = 1/2^j,\ j = 1, 2, 3, .., R-1$ **do**
4: select the cell in the next level where t lies as i;
5: Increment the counter $C_{i,r}$;
6: **if** level $= R-1$ **then**
7: insert point t in the index list of cell i;

Algorithm 2. Biased Box Sampling: Join List - BBSJL

Input: MG-tree and sample ratio $Ratio$
Output: Grid structure Concatenated - MGc-tree
1: $\delta = E * 100/(2 * Ratio)$
2: **while** Cell $\neq$ NULL **do**
3: **if** Next Level Cell $\neq$ NULL **then**
4: **if** Actual Level $<$ R **then**
5: Call $BBSJL(ActualLevel + 1, Ratio)$;
6: **else if** $C_{r,i} < \delta$ **then**
7: Concatenate sibling cells index lists and make parent a leaf node;
8: Cell receives next level cell;

Algorithm 3. Biased Box Sampling: Extract Sample - BBSES

Input: Concatenated Grid structure - $MGc - tree$ and sample ratio $Ratio$
Output: vector with indexes of sampling points
1: **while** Cell $\neq$ NULL **do**
2: **if** Next Level Cell $\neq$ NULL and First $=$ NULL **then**
3: Call $BBSES(Actual\ Level + 1, Ratio)$
4: **else if** First $\neq$ NULL **then**
5: $samplesize = Ratio * C_{r,i}/100$;
6: **if** $Actual\ Level - R \neq 0$ **then**
7: $samplesize = samplesize * 2 * (R - Actual\ Level)$;
8: Selected a point in the index list and insert it into the result;
9: Cell receives next cell;

Algorithm 4. Biased Box Sampling

Input: Dataset with N points, E attributes, number of levels R and sample ratio $Ratio$
Output: Dataset with the selected points
1: Call $BBSCT(Dataset, R)$
2: Call $BBSJL(MG - tree, Ratio)$
3: Call $BBSES(MGc - tree, Ratio)$
4: uniformly selected $N * Ratio/100$ points from the sample obtained by BBSES()

3 Experimental Results

In this section we discuss the results of experiments performed using the sampling algorithm BBS on two synthetic and one real datasets. The "OneBig" dataset has twenty attributes and nine clusters, one containing fifty thousand points and the others containing one thousand points each. The remaining ten thousand points are randomly distributed noise (fifteen percent). The "UniformClusters" has two attributes and five clusters forming one big circle, two small ones, two ellipsoids connected by a chain of outliers and random outliers scattered over the entire space, as described in [3]. "UniformClusters" and "OneBig" present, respectively, uniform and Gaussian intra-cluster distribution. The third dataset is the real world dataset "Pendigits" [1] with sixteen attributes and ten thousand and nine hundred ninety two points.

The same experimental methodology was applied to every dataset. In the first step, we ran the BBS algorithm, and for comparison purposes, we also ran the DBS sampling algorithm presented in [3], the GBS presented in [1][2] and the Uniform Sample (US) algorithm, generating four sample sets for several sampling rates of each dataset. In the second step, we evaluated the quality of the samples regarding the preservation of original properties of the full dataset regarding cluster distribution. Every experiment was repeated 10 times, so 10 samples were created with the same parameters. The values presented are the averages of processing each set of 10 samples. We tuned the parameters of DBS according to the indications in [3]. The parameter a was set as follows: for datasets containing noise and clusters with various densities (including small clusters) a was set to -0.25. The other parameter is the number of kernels for DBS, set to 1000. In the GBS algorithm the only parameter is e, set to 0.5 as indicated in [1].

We applied the well-known clustering algorithm DBSCAN [7] to evaluate the precision of our techniques, over the original datasets and their corresponding samples. DBSCAN is based on local density to discover clusters, and it can detect noise as well. We used the WEKA[3] implementation of DBSCAN, which requires the following parameters: the minimum number n of points that a cluster should have, and the radius ϵ that defines the maximum distance to determine if two points are neighbors or not. The value of n was set as a proportion of the size of the smallest cluster in the dataset. The radius ϵ was experimentally set to 0.2 for every sample size for the "OneBig" and "UniformClusters" datasets, and between 0.4 and 0.5 for the "Pendigits" dataset.

Figure 2 shows a visualization of the "UniformClusters" dataset, both complete (Figure 2(a)) and sampled at a 0.5 percent rate by each algorithm (Figure 2(b): BBS, 2(c): DBS, 2(d): GBS and 2(e): US). As we can see, the samples are different but every one allowed DBSCAN to find four clusters. However, visually we can also see that the sample generated with BBS resembles more closely the original

[1] UCI http://www.ics.uci.edu/-mlearn/MLSummary.html

[2] We would like to thank also Chris Palmer for putting his biased sampling generator on the Web and Alexandros Nanopoulos for send to us his implementation of DBS.

[3] http://www.cs.waikato.ac.nz/ml/weka/

one data. DBSCAN found four clusters in every sample of "UniformClusters", in spite of the five original clusters, due to the chain of outliers between two of the clusters, which was preserved by all sampling methods.

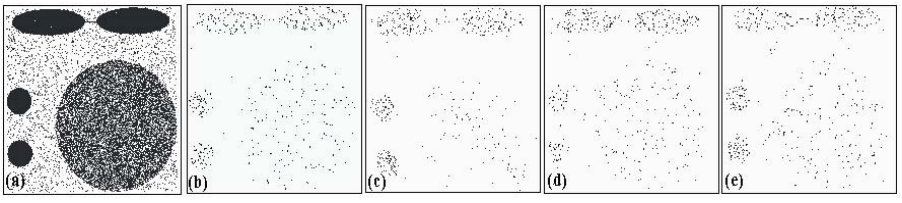

Fig. 2. Visualization of the "UniformClusters". (a) Original dataset, and samples from: (b) BBS, (c) DBS, (d) GBS; (e) US.

As the intra-cluster distribution of datasets "OneBig"(Figure 1) and "UniformClusters" (Figure 2) are distinct, the experiments with "OneBig" led to distinct results. Table 1 shows that the number of clusters found by DBSCAN varies for samples from "OneBig" generated by distinct algorithms. In particular, the DBSCAN was able to find the 9 clusters from the samples generated by BBS, even at a sampling rate as low as 0.5%. All the competing algorithms, on the contrary, were significantly affected by noise and dataset distribution. The DBS in special was strongly affected by the data distribution, and its samples allowed DBSCAN to find just one cluster at any sampling rate. The GBS algorithm is affected by noise and high dimensionality. The US algorithm is also affected by the noise.

Table 3 presents the average amount of noise (in number of points) identified by DBSCAN in samples from "OneBig". As we can see, BBS can filter the noise better than the competing algorithms. Table 4 presents the average error rates produced by DBSCAN when evaluating each sample. The error rates were measured using a class attribute (not submitted to the sampling algorithms nor to DBSCAN), which specifies either the class of the point or whether it is noise. As it can be noted, almost all error rates obtained by DBS is very low. However, looking at Tables 1, 3 and 4 we observe why DBS found just one cluster: DBSCAN correctly classified almost every point, but DBSCAN also found that every point in the DBS samples were from only one cluster, that is, DBS sampled points almost only from the big cluster. The GBS and US algorithms led to bigger error rates than DBS, they allowed finding more clusters than DBS. US and GBS algorithms retrieve points from all clusters in every sample, but several of the sampled clusters do not have enough points to allow DBSCAN to find them. In contrast, samples from BBS allowed DBSCAN to find every cluster in every sample, even at the lowest sampling rate, and always with the lowest error rate.

The last experiment was carried out on "Pendigits" a well-known dataset that illustrates the efficiency of BBS over real data. We applied a different experimental methodology for this dataset, as its number of clusters was not known

Table 1. Number of Clusters Found in the "OneBig" dataset

Algorithm	Sample Size (%)					
	0.5%	1%	1.5%	2%	3%	4%
BBS	9	9	9	9	9	9
DBS	1	1	1	1	1	1
GBS	6	6	7	7	8	7
US	5	7	7	6	8	5

Table 2. Number of Clusters Found in the "Pendigits" dataset

Algorithm	Sample Size (%)			
	3%	4%	5%	10%
BBS	7	7	7	8
DBS	5	5	6	6
GBS	3	4	7	6
Random	2	3	6	2

Table 3. Average noise in the "OneBig" dataset (number of misclassified points)

Algorithm	Sample Size (%)					
	0.5%	1%	1.5%	2%	3%	4%
BBS	3	9	23	50	92	176
DBS	193	292	4	6	10	7
GBS	52	113	194	257	367	479
US	70	113	179	241	320	584

Table 4. Average Error Percentage in samples of the "OneBig" dataset

Algorithm	Sample Size (%)					
	0.5%	1%	1.5%	2%	3%	4%
BBS	0.3	0	0.21	0.08	0	0
DBS	5	43	0.38	0.43	0.48	0.26
GBS	3.24	2.9	1.8	2.05	1.1	2.2
US	5.9	1.6	2.4	2.3	1.2	4.4

beforehand. We applied DBSCAN to the original data, setting the minimum number of points (n) in a cluster as ten percent of the dataset. For ϵ, several values were also evaluated, but DBSCAN was able to find a maximum of 8 clusters in the original dataset only when setting $\epsilon = 0.4$, so this is the value employed in the experiments. The results from the experiments are shown in Table 2, where we can notice that BBS allowed the best clustering accuracy. Although BBS required a sampling rate of 10% to allow DBSCAN to find 8 clusters, the other algorithms did not allow finding more than 7 clusters at any sampling rate.

As expected, almost every sampling algorithms generated the samples sets in low computational time in. Only the DBS took ≈ 15 minutes while every other algorithm took less than 8 seconds to generate a sample. On the other hand, DBSCAN spent 3 hours to process "UniformClusters", 11 hours to process "OneBig" and 18 minutes to process "Pendigits".

The experiments show that BBS is efficient for biased sampling. Furthermore, BBS is tougher to withstand high-dimensionality drawbacks and noise in the datasets than the other techniques. This fact supports us to conclude that the proposed sampling approach is efficient and effective to speed up clustering algorithms, yet having a small impact on their precision.

4 Conclusions

This paper presents a new technique and a corresponding algorithm, the **BBS - Biased Box Sampling**, to perform sampling based on local density. The

technique is based on a multi-dimensional multi-resolution grid structure whose depth depends on the local density of the points in the corresponding region. Therefore, even at low sampling rate, the points are selected such that the representativeness of each cluster occurring in the original dataset is preserved. Moreover, BBS is tougher to withstand high-dimensionality drawbacks and it is less sensitive to noise in the datasets than the competing techniques.

The dataset must be read only twice: one when preparing the grid structure, and thereafter to retrieve the points chosen to be in the sample. In fact, the whole process is linear on both the number of points N and on the number E of attributes in the original dataset. Therefore, the proposed technique can handle dimensionality higher than the other methods. We performed extensive experiments on both synthetic and real-world datasets. They highlighted the fact that the BBS algorithm is a very efficient technique to select samples for clustering algorithms with very little impact on their precision, always outperforming the existing techniques, particularly at very low sampling rate.

References

1. Palmer, C.R., Faloutsos, C.: Density biased sampling: An improved method for data mining and clustering. In: ACM SIGMOD, San Diego, pp. 82–92. ACM Press, New York (2000)
2. Kerdprasop, K., Kerdprasop, N., Sattayatham, P.: Density-biased clustering based on reservoir sampling. In: DEXA Workshops, Copenhagen, pp. 1122–1126 (2005)
3. Kollios, G., Gunopulos, D., Koudas, N., Berchtold, S.: Efficient biased sampling for approximate clustering and outlier detection in large data sets. TKDE 15(5), 1170–1187 (2003)
4. Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y.: Indexed-based density biased sampling for clustering applications. DKE 57(1), 37–63 (2006)
5. Appel, A.P., Paterlini, A.A., Sousa, E.P.M.: d., Traina Jr., C., Traina, A.J.M.: Biased box sampling - a density-biased sampling for clustering. In: ACM SAC, Seoul, Korea, pp. 445–446 (2007)
6. Traina, J.C., Traina, A.J.M., Wu, L., Faloutsos, C.: Fast feature selection using fractal dimension. In: Brazilian Symposium on Data Base - SBBD, João Pessoa, PB, Brazil, pp. 158–171 (2000)
7. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, pp. 226–231 (1996)