Density Biased Sampling: An Improved Method for Data Mining and Clustering

Christopher R. Palmer Christos Faloutsos

May 26, 1999

CMU-CS-99-113

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was partially funded by the National Science Foundation under Grants No. IRI-9625428 and DMS-9873442. Also, by the National Science Foundation, ARPA and NASA under NSF Cooperative Agreement No. IRI-9411299, and by DARPA/ITO through Order F463, issued by ESC/ENS under contract N66001-97-C-851. Additional funding was provided by donations from NEC and Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the United States Government.

Keywords: Random sampling, spatial clustering, data mining

Abstract

Data mining in large data sets often requires a sampling or summarization step to form an in-core representation of the data that can be processed more efficiently. Uniform random sampling is frequently used in practice and also frequently criticized because it will miss small clusters. Many natural phenomena are known to follow Zipf's distribution and the inability of uniform sampling to find small clusters is of practical concern. Density Biased Sampling is proposed to probabilistically under-sample dense regions and over-sample light regions. A weighted sample is used to preserve the densities of the original data. Density biased sampling naturally includes uniform sampling as a special case. A memory efficient algorithm is proposed that approximates density biased sampling using only a single scan of the data. We empirically evaluate density biased sampling using synthetic data sets that exhibit varying cluster size distributions. Our proposed method scales linearly and out performs uniform samples when clustering realistic data sets.

1 Introduction

Uniform sampling is often used in database and data mining applications and Olken provides an excellent argument for the need to include sampling primitives in databases [17]. Whether or not uniform sampling is the "best" sampling technique must be evaluated on an application by application basis. Some records may be of more value in the sample than others. If we knew the value of each record, we could sample by assigning a probability proportional to the importance of the record. It is unlikely that we can define the value of each record in the database and, worse yet, it may be difficult to generalize results obtained from such a sample because the sample is no longer representative of the database. Instead, we'll consider applications in which it is possible to define sets of equivalent records and use the size of these sets to bias our sample while ensuring that the sample is still representative. Data mining applications on spatial data are a natural application because we have a simple notion of equivalent points: points that are close. To show the applicability of using groups of equivalent points to bias the sample, we will concentrate on clustering a database.

Clustering can be generally defined as the following problem. Given N points in d dimensional feature space, find interesting groups of points. There is no definitive way to quantify "interesting" but many algorithms assume that the number of clusters, k, is known a priori and find the k clusters that minimize some error metric. Other algorithms look at areas of space that are denser than some threshold parameter and then form clusters from these dense regions. Clustering is of practical importance in many settings. For example, clustering can be used for classification problems in machine learning [16], in information retrieval to identify concepts [4] or to improve the presentation of web search results [22], by physicists to find the spatial grouping of stars into galaxies [15] and in general to find relationships in the data and to succinctly model the data distribution. Interesting problems for all of these applications involve data sets that have at least a million points.

A typical clustering algorithm will initialize the parameters of the model (randomly or based on a sample) and iteratively use the model to assign the data to group(s). According to this assignment, a new model is constructed. This iterative process involves the entire data set at each step and take an unbounded number of steps to converge. It's essential that we reduce the data size. One solution is to summarize the data and create a new representation that is more compact. The best algorithms based on data summarization use the current model to summarize a subset of the data [2, 3]. As such, it's imperative that the initial model for a summarizing clustering algorithm be representative of the data.

Alternatively, many people use a p-uniform sample (a sample in which each element has probability p of being selected). A sample is selected from the database and clustered. Provided that the sample was representative of the data, the clustering is expected to generalize to the entire data set. Once the sample has been clustered, a single pass over the database to correct for small errors (due to sampling)

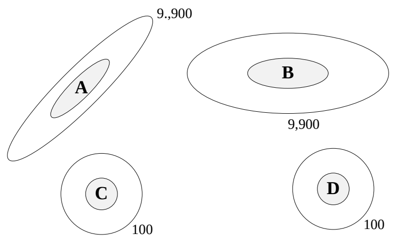

Figure 1: Example: four clusters with very skewed sizes

is recommended. To see why uniform sampling is not necessarily ideal, consider the example in figure 1. This example and much of the discussion is based on data generated by the "mixture model." The mixture model assumes that the data is generated by a mixture of k Gaussian distributions. Each distribution has a corresponding mean and covariance matrix and points are assumed to have been generated by one of these Gaussians. Our example contains 4 clusters and the distribution of points between clusters has been dramatically skewed: clusters A and B each contain 9,900 points while clusters C and D each contain only 100 points. The shaded area contains most of the points of each cluster (the dense core of the Gaussian distribution). A 1% sample of this data set would be expected to draw around 99 points from each of A and B and a single point from each of C and D. For any given sample, if one or more points are actually selected from the C and D clusters, they will likely be treated as noise by the clustering algorithm. That is, we expect that clusters C and D will be completely missed!

Let us consider what has happened with this uniform sample to see what properties are needed by a good sampling technique. First, it's important that the sample contain many points from the shaded region because they will be the best representatives of the cluster. Uniform sampling has this property since the shaded area is the dense core of the Gaussian. But, as we saw, the uniform sample fails because it is not representative of all the groups of points. We want to sample more evenly from all the different groups. For example, if we already knew the clusters, we'd rather randomly pick 50 points from each cluster to form the sample. Using the size of the groups to bias the sample is the heart of our proposed method and we call this a *Density Biased Sample*. The density biased sample here will still contain more points from the shaded regions. It is not necessarily a good sample because it is no longer representative of the data (it makes A and C appear to be the same size). Instead, we notice that each sampled point from clusters A and B is representing $\frac{9,900}{50} = 198$ points, while each sampled point from cluster C and D is only representing 2 points. Augmenting the sample with a weighting of the points is called a *Weighted Sample*.

Clusters sizes are not actually expected to be skewed as dramatically as was shown in the example. Instead, it seems more likely that cluster sizes will follow a Zipf distribution. Zipf distributions occur extremely frequently in practice: they have been found in the frequency distribution of vocabulary words in text (English and Latin works of literature [24]; Bible [6]); the distribution of city populations [24]; distribution of first and last names of people [5]; sales patterns [6]; income distributions (the "Pareto law" [20]); and distribution of web-site hits [13].

The main contribution of this paper is to introduce a new sampling technique and an efficient algorithm that improves on uniform sampling when cluster sizes are skewed. The rest of the paper is organized as follows. First, we present density biased sampling in general terms, parameterized to form a set of sampling techniques that includes uniform sampling as a special case. We then comment on related work. Next, we develop a one-pass algorithm that produces an approximate density biased sample and informally characterize its behaviour. Experimental results follow which demonstrate that density biased sampling is more effective than uniform sampling when the size of the clusters is skewed.

2 Density Biased Sampling

Suppose that we have N values $x_1, x_2, \cdots, x_N$ that are partitioned into g groups that have sizes $n_1, n_2, \cdots, n_g$ and we want to generate a sample with expected size M in which the probability of point x_i is dependent on the group sizes (particularly dependent on the size of the group containing x_i). Our example from figure 1 suggested the criteria that we want our sampling to satisfy. We will define a probability function and a corresponding weighting of the sample points that satisfies:

i) Within a group, points are selected uniformly.

ii) The sample is density preserving.

iii) The sample is biased by group size.

iv) Expected sample size is M.

Density preserving means that the expected sum of the weights of the points in the sample from a group is proportional to that group's size. That is, if group i contains the points $\{x_1, x_2, \cdots, x_{n_i}\}$, point x_j is included in the sample with weight w_j with probability $P(x_j)$, then

$$\sum_{j=1}^{n_i} w_j \cdot P(x_j) = \kappa n_i$$

for some constant κ. This formalizes the notion of "representative of the data distribution."

To satisfy criterion i), we define $\mathtt{P}$(selecting point $x \mid x$ in group $i) = f(n_i)$. Each point in the group then has the same probability of being selected and we assign each point from the group equal weight

$w(n_i) = 1/f(n_i)$. The expected weight of the points in group i is:

$$\sum_{j=1}^{n_i} P(\text{selecting point } x_i) \cdot w(n_i) = \sum_{j=1}^{n_i} f(n_i) \cdot 1/f(n_i) = n_i$$

which satisfies property ii). To bias the sample by group size, we define $f(n_i) = \frac{\alpha}{n_i^e}$ for any constant e. Notice that for $e = 0$, we have simply defined a uniform sample (independent of group assignments) and for $e = 1$ we expect to select the same number of points per group (as in the example). We define α such that the expected sample size is M (requirement iv):

$$\begin{aligned} E(\#\text{ points in the sample}) &= \sum_{i=1}^{g} E(\#\text{ points in the sample from group } i) \\ M &= \sum_{i=1}^{g} n_i f(n_i) = \sum_{i=1}^{g} n_i \frac{\alpha}{n_i^e} \\ \Rightarrow \alpha &= \frac{M}{\sum_{i=1}^{g} n_i^{1-e}} \end{aligned}$$

The following observations apply in general and will be useful when we discuss an implementation of density biased sampling. First, if there are g groups of size $\frac{n}{g}$ then every point is assigned the same probability and weight. That is, we have implemented uniform sampling. The second observation is that if each point is randomly assigned to a group, we will approximate uniform sampling. Since each point is randomly assigned to a group, the expected group size is n/g. That is, the expected behaviour will be to approximate a uniform sample when points are randomly assigned to groups.

If a density biased sample is being used for clustering purposes and the data truly was generated by a mixture model, then the ideal definition of groups appears to be the clusters themselves. The parameter, e, controls the sampling balance based on cluster sizes. For the special case that all clusters are of equal size, the ideal groups are the cluster and so, because the groups are equal sized, ideal density biased sampling is equivalent to uniform sampling. This observation will be useful in our experiments to quantify the effects of using a simple set of groups.

3 Related Work

Sampling has attracted much interest in databases: Olken et al. give algorithms for uniform sampling from hash tables and index trees [17]; Hellerstein et al. use sampling to give approximate answers to aggregation queries [12]; Haas et al. use sampling to make estimates for the number of distinct values of an attribute for query optimization [10].

Sampling is also used extensively for data mining: Commercial vendors of statistical packages (eg., SAS, at `http://www.sas.com/`) typically use uniform sampling to handle large datasets.

Clustering is one of the typical operations in data mining. There is a *huge* literature on clustering for Information Retrieval (see [18] for a recent survey), with additional interest in social and biological sciences (see [11]). Clustering for large datasets has attracted a lot of interest in the database field.

Zhang et al. proposed the BIRCH algorithm which was the first to explicitly use a data summarization step [23]. A tree of spherical groups of points (a CF-tree) is built and the size of spheres is grown as memory is exhausted. The assumption that the points may be summarized as spheres is often criticized and more recently Bradley, Fayyad and Reina have used the current model of the data to select points that should be summarized by their sufficient statistics [2, 3]. They show that this model based summarization is more effective than the BIRCH CF tree summarization. To produce good clustering results, they assume that the data is randomized (or at least the initial portion of the data is a random sample). It seems that a density biased sample would be ideal to "seed" their initial model.

Uniform sampling has also been used directly for clustering. CURE uses a uniform random sample and a new hierarchical clustering algorithm that out-performs BIRCH in experiments using non-spherical clusters that are unevenly sized with noise in the data [9]. They ensure that the sample is large enough to adequately cover all clusters. Salem directly compares uniform sampling against CF-tree summarization finding the uniform sampling is as good a representation for sufficiently large samples [14]. Sampling in both papers is done using Vitter's reservoir sampling [21].

Density Biased Sampling (DBS) is related to previous sampling techniques. In particular, Probability Proportional to Size (PPS) sampling has similarities to DBS. PPS sampling is a multi-stage sampling technique. The data is grouped and then some subset of the groups are chosen. From the chosen groups, elements are added to the sample. In PPS sampling, the selection of groups is biased proportionally to their size. DBS will be inversely biased by group size and is a one stage sampling technique. PPS sampling would be difficult if not impossible to implement as a one pass algorithm. Stratified sampling is used for spatial analysis (for example, [19]). Stratified sampling is another form of two stage sampling.

4 Approximating Density Biased Sampling

Density biased sampling requires that the data be partitioned into "groups." We have no a priori knowledge of how the data will be distributed and adopt the obvious technique for grouping the points. Numerical attributes are divided into G bins and categorical attributes have a bin for each category. When dealing entirely with d-dimensional numerical data, the space is divided into bins by placing a d-dimensional grid over the data. If the data is drawn from a low dimensional space and the number of occupied bins, B, is small, we can compute the bin counts using $O(d \times B)$ bytes. We call this an *exact density biased sample*. If too many of the bins are occupied, an approximate histogram algorithm can be used [8]. But, in higher dimensions, it becomes prohibitively expensive to merely represent the

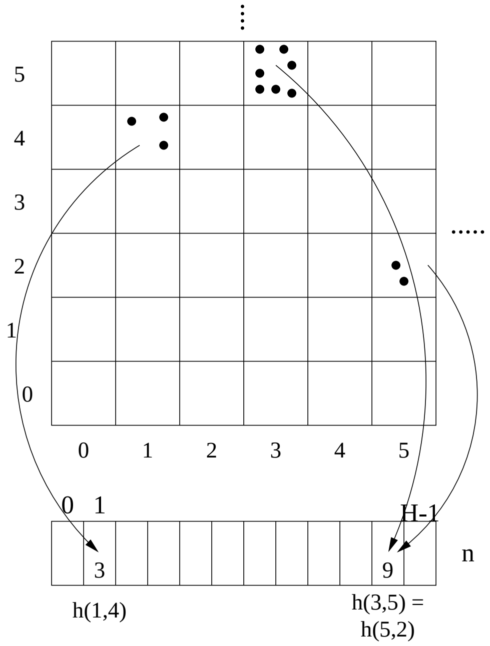

Figure 2: Density approximation by hashing

occupied bins.

Those potential implementations suffer from the lack of available memory. We propose a hashing based approach where all available memory is used to create an array of bin counts. Call this array n (then $n[i]$ corresponds to n_i in the previous section) and assume that it has H entries (indexed from 0). To index into this array, we will use a hash function from the bin label to array index (see figure 2). The bin labels are integers (either in the range $0, \cdots, G-1$ for numerical data or $0, \cdots, c-1$ for categorical data with c categories). Hashing bin labels should be similar to hashing strings since each element is expected to be drawn from a relatively small range and we expect values to frequently differ in only one position (adjacent bins). Aho, Sethi and Ullman suggest that the hash function shown in figure 3 is appropriate for the symbol table of a compiler [1]. For simplicity, assume that $h(x)$ is a function that takes value x, quantizes it and then invokes *hash* on the quantized version. Then the two pass algorithm using a hash function to approximate density biased sampling is trivial and shown in figure 4.

The second pass over the data makes this an unappealing algorithm. If $0 \leq e \leq 1$, this algorithm can be converted into a one pass reservoir style algorithm. The following lemma is needed to produce a correct algorithm:

Lemma 1 If, when the data is restricted to the first j records, the probability of outputting some record x is P_j. Then for $j \leq j'$, $P_j \geq P_{j'}$.

Proof. *Sketch* n^e and n^{1-e} are monotone increasing functions and the probability function that we use

```
hash((v1, ..., vd)) =
  for i = 1 to d do h = h*65599 + vi
  return h MOD H
```

Figure 3: Hash function

```
FOR each input point x DO n[h(x)] = n[h(x)]+1
Reset the input and compute alpha
FOR each input point x DO
   with probability alpha/n[h(x)]^e, output (n[h(x)]^e/alpha, x)
```

Figure 4: Two-pass hash approximation to density biased sampling

is of the form

$$P = \frac{M}{n_x^e \sum_{i=1}^{g} n_g^{1-e}}$$

As more data is processed the number of terms in the summation will never decrease nor will any value of n decrease and consequently the denominator of the probability function will be monotone increasing and the probabilities will be monotone decreasing. $\square$

A buffer of points that have some chance of being in the sample will be maintained. The buffer contains elements $\{< P_i, x_i >\}$ to indicate that x_i was added to the buffer with probability P_i. Suppose that at some later point, x_i would have probability P_i' of being output. We can convert the current output buffer into a buffer that is a density biased sample of the currently processed data. The lemma tells us that $P_i' \leq P_i$ and consequently we will never erroneously discard a point due to an underestimate of its probability. If we keep $< P_i', x_i >$ in the buffer with probability P_i'/P_i (otherwise, remove this entry from the buffer), then x_i is in the buffer with probability P_i'. The weight of a point is just $1/P_i'$ which means that we can output the weighted sample from the reduced buffer. The one pass algorithm is shown in detail in figure 5

Assuming that `reduce` always removes at least one entry, this algorithm is equivalent to the 2-pass version. It is equivalent because the current output buffer is always a superset of a density biased sample and the reduce operation converts it to a density biased sample of the data process to this point. When reduce fails to remove any entries, we randomly select an entry to evict. This happens quite rarely in practice[1]. In our experiments, the output buffer is of size $1.1 \times M$ to generate a sample of expected size M. The two lines marked with `(*)` compute the current denominator of α in constant time (instead of time proportional to the number of bins).

[1]This approach, of course, creates a small bias toward points later in the database. It is trivial to correct this problem by recording the number of times that each point in the buffer "survived" one of the random evictions and using this to weight the selection of the point to evict. But, since this does appear to be insignificant, it is not developed further.

```
alpha_den = 0
FOR each input point x DO
     IF n[h(x)] != 0 THEN alpha_den = alpha_den - n[h(x)]^(1-e)       (*)
     n[h(x)] = n[h(x)] + 1
     alpha_den = alpha_den + n[h(x)]^(1-e)                             (*)
     WITH probability P = min{ M/(alpha_den * n[h(x)]^e), 1 }  DO
          IF the output buffer is full THEN reduce()
          add <P, x> to the output buffer
reduce()
FOR each output buffer entry <Pi, xi> DO output <1/Pi, xi>

reduce() is
    FOR each output buffer entry <Pi, xi> DO
         Let P'i = min{ M/(alpha_den*n[h(x)]^e), 1 }.
         WITH probability P'i/Pi replace this entry with <P'i, xi>
         OTHERWISE remove this entry
```

Figure 5: One pass hash approximation to density biased sampling

Obviously this one pass algorithm can be used for any representation of the bin densities. Hashing is only used to map from input point to group size and any other mapping could be used here instead.

Collisions are a possible problem for this algorithm. It seems that the ideal value of e would be 1 because the sample will always be density preserving. If bins g and $g+1$ collided to form bin g then the expected weight of the points from bin $g+1$ in the sample is:

$$n_{g+1} \cdot P(\text{selecting a point from } g+1) \cdot \text{weight of a point}$$

$$= n_{g+1} \cdot \frac{\frac{M}{g}}{n_g + n_{g+1}} \cdot (n_g + n_{g+1}) = \frac{M}{g} n_{g+1}$$

This computation is particularly interesting because it illustrates why collisions will not be a serious problem in practice. If $n_g >> n_{g+1}$ then any points in bin $g+1$ will be heavily over-weighted in the sample. But, the probability of selecting one of the incorrectly weighted points will be very small. This means that with high probability we will not output any points from bin $g+1$. If two bins of about equal size collide, an exact density biased sample would be expected to output more points from each of these bins. Collisions perturb the sample but we see that a relatively small number of collisions will not tend to dramatically change the sample.

Finally, a few words about the sensitivity of this algorithm to the parameters. For very small values of H (the hash table size), the bins will be essentially randomly distributed to the various elements of n and this will generate an approximation of a uniform sample (by our observation of random group assignments). Similarly, if G is too large and each bin has occupancy 1 or 0 then this algorithm outputs a uniform sample (because each occupied bin will be of equal size). For very poor choices of the hash

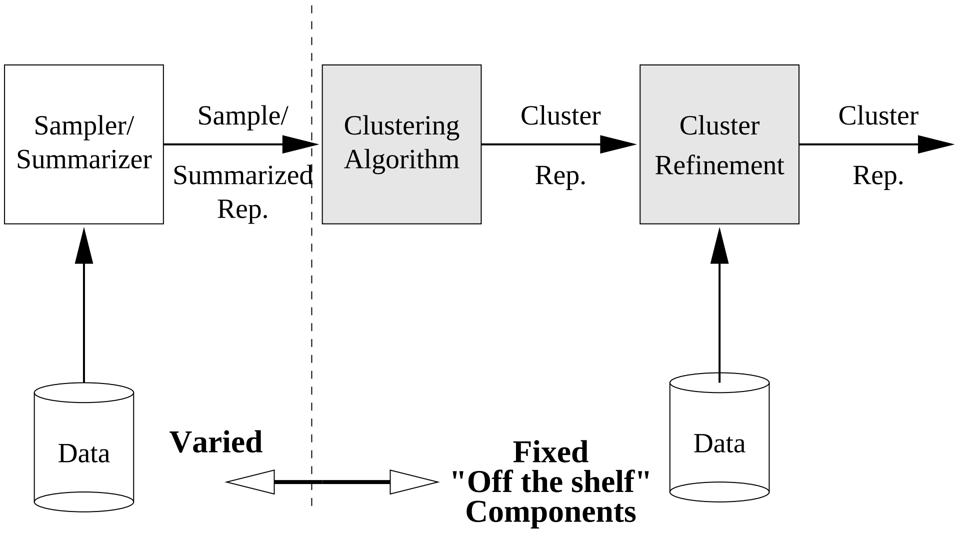

Figure 6: Clustering framework

table size or the number of bins per attribute, hash based density biased sampling will reduce to a uniform sample. That is, the algorithm is expected to be quite robust to poor parameter choices.

5 Experiments

There are several unknowns that will be explored in our experiments. First, and foremost, we wish to see that density biased sampling provides better clustering results than uniform sampling and BIRCH when the cluster sizes follow a Zipf distribution. The Zipf data set will constitute an average case and we explore a very skewed distribution of cluster sizes and a data set in which cluster sizes are all equal to observe a range of behaviours.

We have chosen to represent groups by binning the data. This will have some effectiveness implications. Using equal sized clusters makes uniform sampling equivalent to ideal density biased sampling. We can measure the effect of the binning by looking at this extreme case. We will find that binning introduces a small error that is acceptable given the significant improvements seen for realistic cluster size distributions.

Our approximation uses hashing to map bins to their respective counts. This introduces an error resulting from collisions. We will measure the effects of collisions and find that they make little to no difference in the effectiveness of the sample.

5.1 Methodology

Several sampling or summarizing algorithms are compared experimentally. An experiment consists of selecting a data distribution, a clustering algorithm and then varying the amount of available memory to measure performance for various sample sizes. All the contending methods use a single pass over the data to generate a sample, weighted sample, or a summarized representation of the data. The algorithms used are:

i) BIRCH. Summarization is done with CF-trees and the maximum available memory will be limited to 2x the space needed to hold the sample [23].

ii) Uniform random sampling. A reservoir sampling algorithm is used and requires only the amount of memory needed to represent the sample [21].

iii) Hash based approximation to density biased sampling. The amount of memory used is twice the amount of memory needed to represent the sample. We will use two values for e for density biased sampling. Using $e = 1$ is *Inverse Biased Sampling* (IBS) and $e = .5$ is *Inverse Root Biased Sampling* (IRBS).

iv) Exact density biased sampling. The occupied bins are represented explicitly and the memory needed is not restricted. The only difference between iii) and iv) are collisions in the hash table. For $e = 1$, call this *Exact IBS* and, for $e = .5$, call this *Exact IRBS*.

BIRCH is included because it is extensively studied and has recently been directly compared to uniform sampling for equal sized clusters in low dimensional space [14]. Our experiments extend the cases in which BIRCH and uniform sampling have been directly compared. The hash based algorithm requires auxiliary memory. The hierarchical clustering algorithm uses about twice the memory needed to represent the sample and thus the decision to allow BIRCH and the hash based algorithm the opportunity to use this memory is reasonable.

BIRCH is provided with a default configuration that uses the framework shown in figure 6. The sampling and the refinement step will be those used by BIRCH and are considered to be "off the shelf" components that are beyond our control. In BIRCH, the sampling step builds the CF-trees, the clustering step uses a simple hierarchical clustering algorithm and the refinement step implements a single iteration of the k-means algorithm (the output clusters are the center of mass of all the points that are included in the cluster). This framework is used by all our competing algorithms and consequently any differences in performance are directly attributable to the sampling/summarizing technique.

5.2 Evaluation Metrics

The natural evaluation metric for the BIRCH algorithm is the root mean square (RMS) distance to cluster centers. If we assign each point $x_i = (x_i^1, \cdots, x_i^d)$ to the closest cluster center, $c_i = (c_i^1, \cdots, c_i^d)$,

Param.	Value(s)	Interpretation
N	200,000	Number of points
d	20 or 50	Number of dimensions
k	500	Number of clusters
σ	.05	Measure of standard deviation
$p_1, \cdots, p_k$	Varies	Probability of cluster membership

Figure 7: Parameters for data generation

then the distance from the center is the standard Euclidean distance

$$\parallel x_i - c_i \parallel_2 = \sqrt{\sum_{a=1}^{d} (x_i^a - c_i^a)^2}$$

and the root mean square error is defined to be

$$\sqrt{\frac{\sum_{i=1}^{n} \parallel x_i - c_i \parallel_2^2}{n}}$$

RMS distance does not provide all of the information that we need. We are particularly concerned with the number of clusters that are actually found by the respective algorithms. RMS distance does not provide this information and we introduce a very simple metric to count the number of clusters that are "found."

Suppose that we knew that the true cluster centers were $\{c_1, c_2, \cdots, c_k\}$ and we wish to evaluate a system that found cluster centers $\{\hat{c}_1, \hat{c}_2, \cdots, \hat{c}_{k'}\}$. We say that cluster c_i is found if $\exists\ \hat{c}_j$ with $\parallel c_i - \hat{c}_j \parallel_2 < \epsilon$. We select $\epsilon = 0.001$ and define the metric *Number of Clusters found* (NC) to be the number of the true clusters that are "found." Notice that algorithms are not rewarded for finding the same cluster more than once nor are they rewarded for merging clusters.

5.3 Data Generation

We randomly generate data based on the parameters in figure 7 using a mixture model. There are k clusters and N points in the d-dimensional unit hypercube. Each of the d attributes for a cluster center are generated in the range $[.1, .9]$. A diagonal covariance matrix is generated by computing a variance in the range $[0, \sigma^2]$ and using the square root of the variance. Covariances computed in this fashion are in the range $[0, \sigma]$ but will have very few small values. Since $\sigma = .05$ and the centers are generated in the range $[.1, .9]$, the majority of the points will be in the unit hypercube. For simplicity we discard the few points that are outside the hypercube. To generate the data, each point is randomly assigned to a cluster using the the probability distribution $P(\text{cluster } i) = p_i$. Once assigned to a cluster, the appropriate mean and covariance is used to generate a point according to a Gaussian distribution.

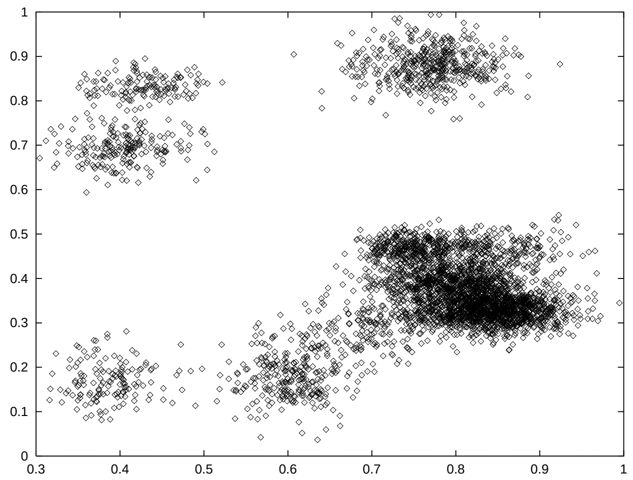

Figure 8: Sample synthetic data: $n = 4,000$, $d = 2$, $k = 10$, $\sigma = .05$

The center of mass of each cluster is recorded for future use in computing NC. By using the center of mass and not the randomly generated mean, any clustering that correctly classifies all the points will be guaranteed to "find" the cluster.

Three different cluster membership distributions are used:

i) *Even*: All clusters are equally likely ($p_i = 1/k$).

ii) *Zipf*: Cluster sizes follow a Zipf distribution ($p_i = 1/(H_k \cdot i)$ where H_k is the kth harmonic number).

iii) *OneBig*: One cluster has most of the points, all other clusters have 100 points ($p_1 = (n - 100(k - 1))/n$ and $p_i = 100/n$ for $2 \leq i \leq k$).

Figure 8 shows a very small example of the data that is generated. This data was generated with Zipf sizes, $k = 10$, $n = 2000$ and all other parameters unchanged. The clusters are fairly well separated but we see that several clusters overlap in the bottom right quadrant. It will be very difficult to exactly find every cluster in data generated by this procedure.

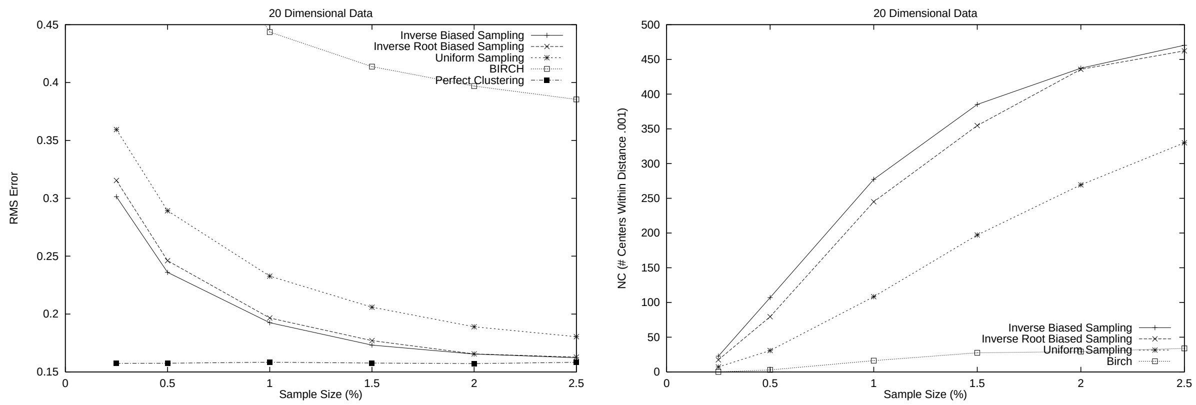

Figure 9: Zipf sizes (d=20): RMS Error predicted and NC for the same experiment

6 Results

Figure 9 shows performance for the algorithms for various sample sizes (equivalently memory size) for 20-dimensional data with Zipf cluster sizes. We see that the relative orderings of the contending algorithms is identical for the two metrics and we can see similar relative performance. This relationship is true for all the experiments reported. Either metric could be used to draw similar conclusions. Due to space limitations, we can only present one metric and use NC because it is easier to unambiguously interpret.

BIRCH performs quite poorly in these experiments. BIRCH appears to require memory that is 10% of the total database size to perform well in our 20 and 50 dimensional experiments. BIRCH tends to do slightly better under the RMS distance metric than NC metric but is generally still quite poor.

In the average case, IBS and IRBS are much better than uniform sampling. Figure 11 shows NC for various sample sizes for 20 and 50 dimensional data with Zipf cluster sizes. For 1% samples, IBS and IRBS find approximately 2.3 times as many clusters as uniform sampling in 20 dimensions and more than twice as many clusters in 50 dimensions. A 2.5% IRBS or IBS sample finds 90% of the clusters while a 2.5% uniform sample finds fewer than 70% of the clusters.

As the cluster sizes become more skewed, this difference in performance increases. Figure 10 shows the same information for data sets in which the OneBig cluster sizes are used. For 1% sample sizes, IBS and IRBS find between 4 and 6 times as many clusters as uniform sampling. A 2.5% IBS or IRBS sample finds more than 95% of the clusters while a 2.5% uniform sample still finds fewer than 70% of the clusters.

Figure 12 shows that binning is a good approximation to the ideal groups for IRBS but not as good for IBS. In 20 dimensions, IRBS is typically within 7.5% of uniform and in 50 dimensions generally within 16%. On the other hand, IBS is only within 20% and 43% in 20 and 50 dimensions respectively.

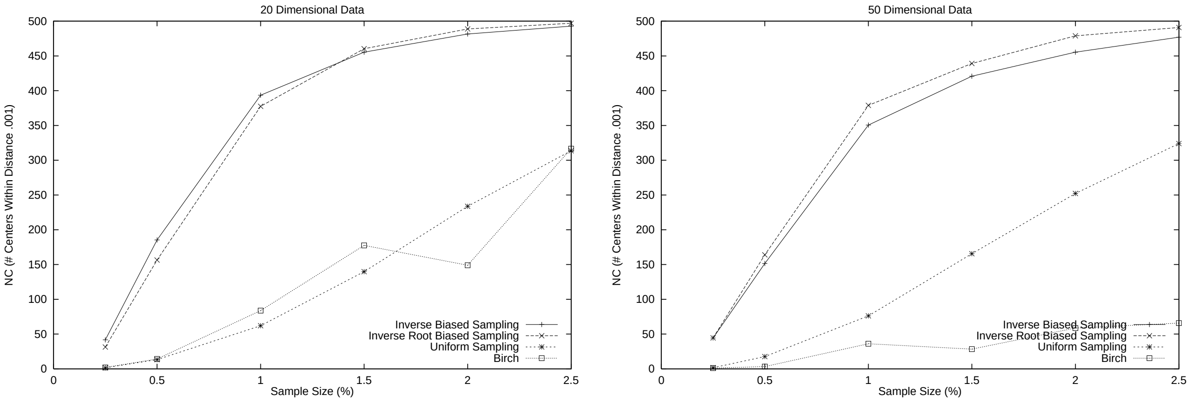

Figure 10: OneBig sizes (d=20/50): Ideal case for density biased sampling

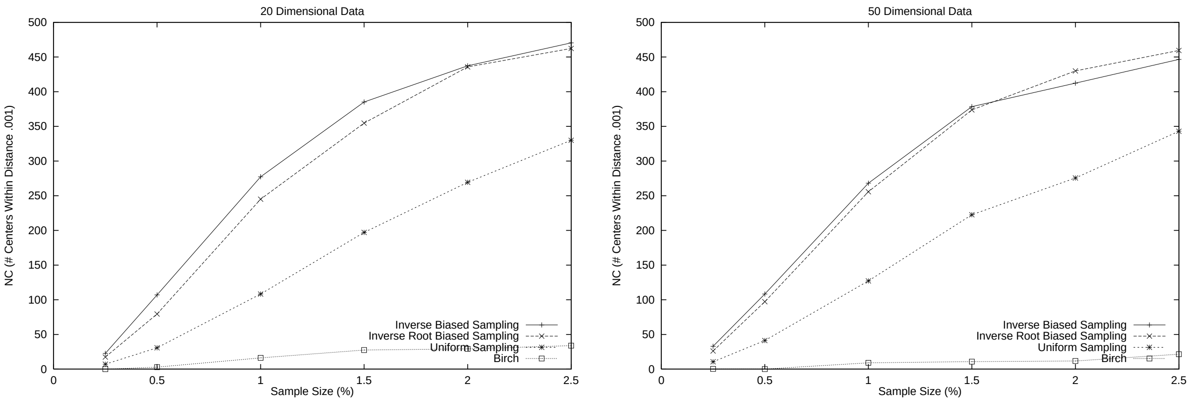

Figure 11: Zipf sizes (d=20/50): Data moderately skewed, density biased sampling excellent

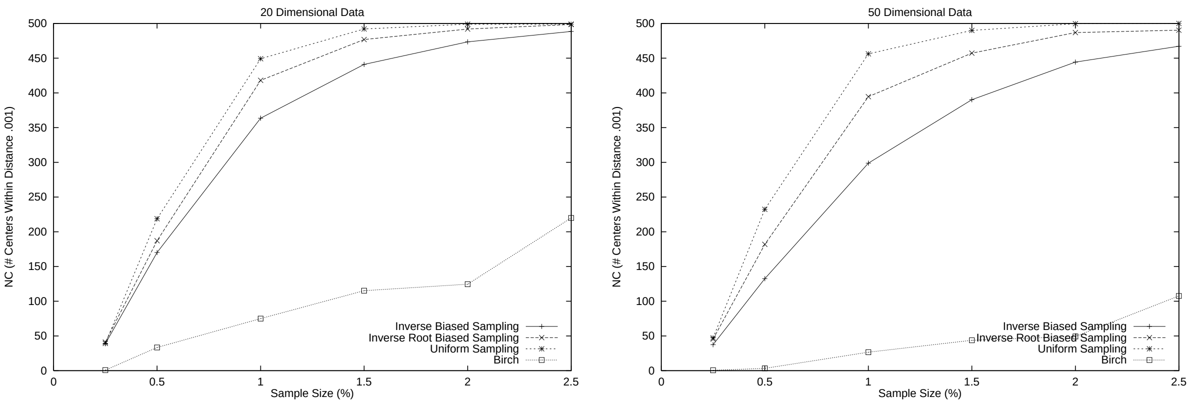

Figure 12: Even sizes (d=20/50): Ideal case for uniform sampling

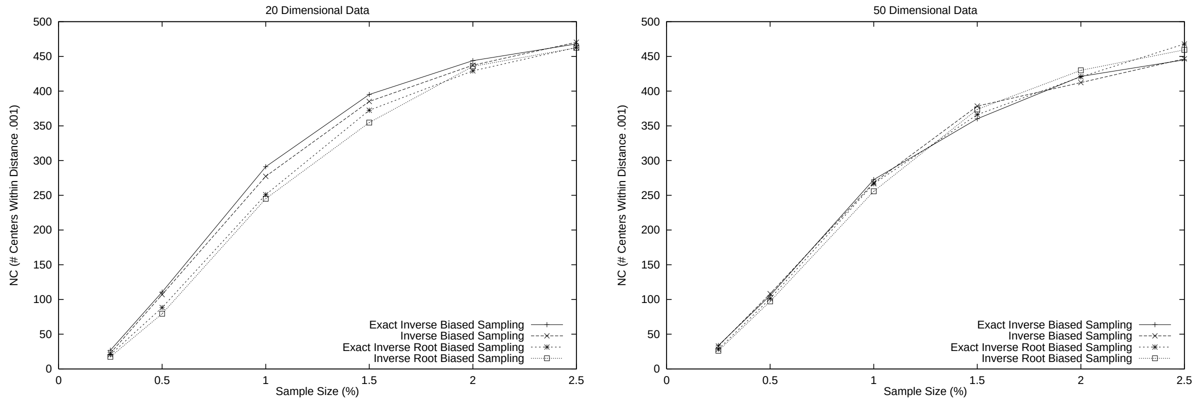

Figure 13: Zipf sizes (d=20/50): Collisions only cause minor differences

We see that IBS is sensitive to the quality of the groupings generated by binning the data but that IRBS is hardly affected by our grid-based choice of group assignments.

Finally, figure 13 shows that collisions have essentially no effect on clustering the Zipf cluster sizes data set. In both 20 and 50 dimensions, the approximation is typically within 10 clusters of exact IBS and exact IRBS.

To summarize:

- IRBS and IBS are much better than uniform sampling for clustering data sets with skewed cluster sizes.
- Using bins is a reasonable choice for IRBS.
- Collisions do not reduce the effectiveness of IBS or IRBS.

We generally conclude that IRBS gives the best performance of any of the algorithms considered.

The running time of all the algorithms is linear in the database size and completely dominated by the cost of reading the data. Figure 14 shows the wall clock running time to generate a 1% sample for a 20 dimensional data set with Zipf cluster sizes. "Read-only" is the time is takes to read the data and perform no other processing. The second row is the wall clock time less the time that it takes to read the data. All algorithms are quite efficient.

7 Applications and Further Research

We have used density biased sampling as a preprocessing step for clustering. Good summarization algorithms assume that the data appears in a random order and that the first component is representative

Method	Read-only	Sample	BIRCH	IRBS
Time (secs)	16.7	17.7	18.1	18.1
Time - “Read-only”		1.0	1.4	1.4

Figure 14: Zipf sizes (d=20): Wall clock execution times

of the data [2, 3]. Using a density biased sample is more likely to satisfy these assumptions than a uniform sample of equal size (in bytes). So, not only can IRBS be used to cluster, it can also be used to improve summarization based algorithms (such as [2, 3]) and to develop better initial models (as done with uniform sampling in [7]).

More generally, density biased sampling offers a representative sample of the data that includes more of the unexpected points. Any algorithm that does not require that all inputs be distinct can be trivially extended to support a weighted sample. Many statistical algorithms use multiple samples to reduce variability. Density biased samples should reduce the variability of the algorithms because we can include more of the “unusual” points (ie, the points that are likely to induce variability) while ensuring a representative sample.

Finally, it appears that it should be possible to efficiently construct a density biased sample using an R-tree index by descending in the R-tree only as far as needed to compute the bin sizes. Using an existing index may make it possible to construct samples without reading the entire database.

8 Conclusions

We proposed a new sampling technique: *Density Biased Sampling*. Density biased sampling naturally includes uniform sampling as a special case. We implemented density biased sampling using a hashing function to map bins in space to a linear ordering allowing it to work with very limited memory. The hash based approximation to density biased sampling with $e = .5$ (IRBS) is more effective for clustering than either a uniform sample or a CF-tree summarization (for realistic data). We found that binning is particular appropriate for IRBS and that collisions had little to no impact on the effectiveness of the sample generated by IRBS and IBS.

The method favours clusters containing fewer points. Uniform sampling tends to miss these smaller clusters. These clusters are the most likely to contain interesting results because the domain experts are likely aware of the very large clusters. Using a Zipf distribution of cluster sizes (an “average” case) and taking a 1% sample, IBS and IRBS find more than twice as many clusters as uniform sampling. As the cluster sizes become even more skewed this increases to between 4 times and 6 times for 20 and 50 dimensions respectively.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers. Principles, Techniques and Tools.* Addison–Wesley, 1986. Pages 433–438.

[2] P.S Bradley, Usama Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD–98)*, pages 9–15, New York City, New York, August 1998. AAAI Press.

[3] P.S Bradley, Usama Fayyad, and Cory Reina. Scaling EM (expectation maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research, Redmond, WA, November, 1998.

[4] Chris Buckley, Mandar Mitra, Janet Walz, and Clarie Cardie. Using clustering and superconcepts within SMART: TREC 6. In *Sixth Text REtrieval Conference (TREC-6)*, Gaithersburg, Maryland, November 1997. National Institute of Standards and Technology (NIST), United States Department of Commerce.

[5] Christos Faloutsos and H.V. Jagadish. On B-tree indices for skewed distributions. In *18th VLDB Conference*, pages 363–374, Vancouver, British Columbia, Aug. 23-27 1992.

[6] Christos Faloutsos, Yossi Matias, and Avi Silberschatz. Modeling skewed distributions using multifractals and the '80-20 law'. *VLDB*, September 1996.

[7] Usama M. Fayyad, Cory A. Reina, and Paul S. Bradley. Initialization of iterative refinement clustering algorithms. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD–98)*, pages 194–198, New York City, New York, August 1998. AAAI Press.

[8] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, volume 27,2 of *ACM SIGMOD Record*, pages 331–342, New York, June1–4 1998. ACM Press.

[9] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, volume 27,2 of *ACM SIGMOD Record*, pages 73–84, New York, June1–4 1998. ACM Press.

[10] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. of VLDB*, pages 311–322, Zurich, Switzerland, September 1995.

[11] John A. Hartigan. *Clustering Algorithms.* John Wiley & Sons, 1975.

[12] Joseph M. Hellerstein, Peter J. Haas, and Helen Wang. Online aggregation. In *SIGMOD Conference*, pages 171–182, 1997.

[13] Bernardo A. Huberman, Peter L. T. Pirolli, James E. Pitkow, and Rajan M. Lukose. Strong regularities in world wide web surfing. *Science*, 280(5360):95–97, April 3 1998.

[14] Najmeh Joze-Hkajavi and Kenneth Salem. Two-phase clustering of large datasets, 1998. Manuscript in preperation.

[15] Jeremy Kepner, Xiaohui Fan, Neta Buhcall, James Gunn, Robert Lupton, and Ghohung Xu. An automated cluster finder: the adaptive matched filter. *The Astrophysics Journal*, 517, 1999.

[16] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[17] Frank Olken, Doron Rotem, and Ping Xu. Random sampling from hash files. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, volume 19,2 of *ACM SIGMOD Record*, pages 375–386. ACM Press, June1–4 1990.

[18] Edie Rasmussen. Clustering algorithms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 419–442. Prentice Hall, 1992.

[19] Brian D. Ripley. *Spatial Statistics.* John Wiley & Sons, 1981.

[20] Manfred Schroeder. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise.* W.H. Freeman and Company, New York, 1991.

[21] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[22] Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 46–54, 1998.

[23] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 25, 2 of *ACM SIGMOD Record*, pages 103–114, New York, June4–6 1996. ACM Press.

[24] G.K. Zipf. *Human Behavior and Principle of Least Effort: An Introduction to Human Ecology.* Addison Wesley, Cambridge, Massachusetts, 1949.