Fast Density Estimation Using CF-kernel for Very Large Databases

Tian Zhang
IBM - Santa Teresa Lab.
tzhang@us.ibm.com

Raghu Ramakrishnan, Miron Livny
Computer Sciences Dept., Univ. of Wisconsin-Madison
raghu@cs.wisc.edu, miron@cs.wisc.edu

Abstract

This paper presents the *CF-kernel* method to estimate density functions from large datasets in a "best-efforts under the given resources" manner. By integrating the advantages of both the *kernel* method [WJ95] and the dynamical and incremental *CF tree* structure[ZRL96], the CF-kernel method improves time/space efficiency dramatically, while keeping the estimation accuracy as close to the kernel method as the available memory allows. Our theoretical proof shows that the CF-kernel method is equivalent to the kernel method under certain assumptions. Our experimental results of applying the CF-kernel method to various datasets show that it is much more time/space efficient compared with the kernel method whereas its estimation accuracy is comparable with, and converges super linearly to that of the kernel method as the memory increases.

1 Introduction

For simplicity, through the paper, we will use 1-dimensional data and equations for illustration, however our approach can be easily generalized to any dimensions. For convenience, we define some standard notations, except where otherwise stated. We assume that we are given a sample of n data points $X_1, ..., X_n$ whose underlying density function $f(x)$ is to be estimated. The symbol $\hat{f}(x)$ is used to denote the estimation of $f(x)$. Generally $\int$ means $\int_{-\infty}^{\infty}$, and $E(x)$ means the expected value, or mean, of variable x.

There are two approaches to density estimation: *parametric and nonparametric*. The parametric approach assumes that $f(x)$ belongs to some parametric family of distributions, such as the gamma family. The main drawback is that estimation accuracy depends heavily on the assumption. The nonparametric approach does not assume any pre-specified functional forms for $f(x)$, and is generally more useful in reality. The kernel method [Sil86, Dev87, WJ95] is a widely studied nonparametric density estimation method. The kernel estimation of $f(x)$ based on n data points is defined as:

$$\hat{f_K}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - X_i). \quad (1)$$

where 1) the **kernel function** $K(x)$ is usually a unimodal, symmetric and bounded density function, such as the standard normal density function; 2) $K_h(x - X_i)$ is the notation for a new density function transformed from $K(x)$ by moving the mean to X_i, and by multipling the standard deviation by h; 3) h is a constant called the **smoothing parameter**. Intuitively, a "bump" is placed on each data point, and the sum of all "bumps" reflects the overall distribution of all data points. The kernel function $K(x)$ determines the shape of each bump while the smoothing parameter h determines the width, or scope, of each bump.

Some desirable properties of the kernel method are: 1) Mathematically, no curse of dimension. 2) No need to know the data range in advance. 3) $\hat{f_K}(x)$ inherits the continuity, differentiability and integrability properties from $K(x)$. 4) $K(x)$ and h are the two factors determining the accuracy. However it has been shown[WJ95] that the choice of $K(x)$ is not critical, and the accuracy is primarily affected by the smoothing parameter h. 5) Convergence: It has been proved mathematically that with reasonable assumptions, as n approaches ∞, $\hat{f_K}(x)$ converges to $f(x)$ in terms of $MISE$ (Mean Integrated Squared Error[Sil86]).

However if the kernel method is applied to very large datasets, it is very time/space intensive because there are n distinct terms, or "bumps", in $\hat{f_K}(x)$. So it needs $O(n)$ space to store $\hat{f_K}(x)$, and then for a specific value x, it needs to scan all n terms to compute $\hat{f_K}(x)$. To overcome the problem of the kernel method, in our approach, we first summarize the dataset into an in-memory CF tree[ZRL96]. Then, instead of placing a **kernel function** on each data point, we place a **CF-kernel function** on each subcluster (or leaf entry of

KDD-99 San Diego CA USA

the CF tree), and we use the sum of all the CF-kernel functions to estimate the overall data distribution. We call it the *CF-kernel* method and demonstrate that 1) it scans the dataset once, and afterwards it improves the space/time complexity from $O(n)$ to $O(m)$, where m is the number of subclusters; 2) The estimation obtained from the CF-kernel method approximate that obtained from the kernel method, and the approximation can be made as accurate as desired by increasing available memory.

The rest of the paper is organized as below: Sec. 2 gives a review of the CF-tree. Sec. 3 describes the details of the CF-kernel method. The performance results are presented in Sec. 4, and the final comments and conclusions are given in Sec. 5 and Sec. 6.

2 CF Tree

The details of CF and CF tree, and relevant insertion and rebuilding algorithms were introduced in [ZRL96]. Here we provide a brief review. CF is a triplet summarizing the information that we maintain about a cluster. That is, CF $= (N, \vec{LS}, SS)$, where N is the number of data points in the cluster, $\vec{LS}$ is the linear sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{X}_i$, and SS is the square sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{X}_i^{\,2}$. With CF's of clusters known, we can calculate the mean and standard deviation of each cluster, or various distances between clusters very easily.

A CF tree is a height-balanced in-memory tree with two parameters: branching factor B and threshold T. Each nonleaf node contains at most B_n entries of the form $[\mathbf{CF}_i, child_i]$, where $i = 1, 2, ..., B_n$, "$child_i$" is a pointer to its i-th child node, and $\mathbf{CF}_i$ is the CF of the subcluster represented by this child. So a nonleaf node represents a cluster made up of all the subclusters represented by its entries. A leaf node contains at most B_l entries, each of the form $[\mathbf{CF}_i]$, where $i = 1, 2, ..., B_l$. In addition, each leaf node has two pointers, "*prev*" and "*next*", which are used to chain all leaf nodes together for efficient scans. A leaf node also represents a cluster made up of all the subclusters represented by its entries. But all entries in a leaf node must satisfy a *threshold requirement* with respect to a threshold value T. In this paper, we use the threshold restrict that **the standard deviation of each leaf entry has to be less than T.**

A CF tree has been shown to be a compact summary of the dataset because each leaf entry is not a single data point but a subcluster (which absorbs many data points with its standard deviation under a specific threshold T). Such a CF tree can be built dynamically as new data objects are inserted. That is, a new insertion is guided into the correct subcluster in a CF tree leaf node for clustering just as a new insertion is guided into the correct position in a B+ tree for sorting. When available memory is exhausted before data is consumed, a more compact tree can be rebuilt from the existing tree by increasing the threshold value. Additional data can then be inserted into the new tree.

3 CF-kernel Method

Our CF-kernel method improves efficiency by (1) *binning* data with an in-memory CF tree; and 2) placing a CF-kernel function on each subcluster instead of placing a kernel function on each data point. However, it differs from any existing *binned* kernel methods in that it does not require data ranges in advance for allocating bins; and given the memory, it allocates "bins" dynamically and incrementally according to the data distribution. Most importantly, our theoretical proof and empirical results will demonstrate that its accuracy is comparable with, and converges super linearly to that of the kernel method as the memory increases.

Let's say, by building the CF tree from the n data points, we partition the n data points into m disjoint subclusters (or leaf entries), where each subcluster i contains n_i data points, and is represented by its CF triplet. So $\sum_{i=1}^{m} n_i = n$. Now the "subcluster-wise" CF-kernel estimation will be defined as in equ.(2). That is, for each subcluster i, we try to summarize the average effect of the kernel functions placed on all its member points with a single CF-kernel function $CK_i(x)$.

$$\hat{f_{CK}}(x) = \frac{1}{n} \sum_{i=1}^{m} n_i CK_i(x). \tag{2}$$

What will $CK_i(x)$ be like? To answer that, let us compare the "point-wise" $\hat{f_K}(x)$ in equ.(1) and the "subcluster-wise" $\hat{f_{CK}}(x)$ in equ.(2), we can see that if we define the $CK_i(x)$ as shown in equ.(3), then we can guarantee $E[\hat{f_K}(x) - \hat{f_{CK}}(x)] = 0$.

$$CK_i(x) = E[\frac{1}{n_i} \sum_{j=1}^{n_i} K_h(x - X_j)]. \tag{3}$$

Further, if we make the **assumption that the n_i data points in subcluster i are drawn independently from an empirically-known distribution whose density function is $g_i(x)$**, then we can derive the $CK_i(x)$ as shown in equ.(3) into a even simpler form as shown in equ.(4)

$$CK_i(x) = \int K_h(x - t) g_i(t) dt \tag{4}$$

The function $CK_i(x)$ shown in equ.(4) is called the **CF-kernel function** for subcluster i.

$CK_i(x)$ represents the "bump" we place on subcluster i, which is relevant to the original choices of the kernel function $K(x)$ and the smoothing parameter h,

as well as the empirically-assumed density function of subcluster i, $g_i(x)$. Under the distribution **assumption** in subcluster and with the definition of the **CF-kernel function**, the following properties of $CK_i(x)$, as well the theorem about $\hat{f}_{CK}(x)$ compared with $\hat{f}_K(x)$ can be proved easily: 1) If $K(x)$ and $g_i(x)$ are density functions, then $CK_i(x)$ is a density function too. 2) If $K(x)$ and $g_i(x)$ are bounded, then $CK_i(x)$ is bounded too. 3) If $K(x)$ is symmetric, and $g_i(x)$ is symmetric, then $CK_i(x)$ is symmetric too.

Equivalence Theorem: The kernel estimation $\hat{f}_K(x)$, as shown in equ.(1), and the CF-kernel estimation $\hat{f}_{CK}(x)$, as shown in equ.(2), are statistically equivalent. That is, $E(\hat{f}_K(x) - \hat{f}_{CK}(x)) = 0$.

The following conclusions about memory (i.e., number of subclusters) versus accuracy also follow:
1) If $m = n$, that is the memory is large enough for each subcluster to contain only one data point, then the distribution **assumption** in subcluster is not needed at all because no information is lost, and the CF-kernel method is exactly the same as the kernel method i.e., purely nonparametric method.
2) If $m = 1$, that is the memory is so limited that we have to collapse all n data points into a single subcluster, then the distribution **assumption** in subcluster becomes extremely critical because it is exactly the density estimation we are looking for. All information except the CF triplet is lost due to the limited memory. As a result, the CF-kernel method becomes too parametric to be useful.
3) If m is some reasonable value between 1 and n so that the distribution **assumption** in subcluster becomes more acceptable, the CF-kernel method and the kernel method should have almost the same accuracy as we will show experimentally.

3.1 Computing $CK_i(x)$

How to compute $CK_i(x)$ efficiently is the key to the CF-kernel method. From the definition, it relies on the forms of $K(x)$ and $g_i(t)$. As shown in [WJ95] that the choice of $K(x)$ is not critical to the accuracy, in this section, we choose the standard normal density function for $K(x)$, i.e., $K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, and hence $K_h(x-t) = \frac{1}{\sqrt{2\pi}h} e^{-\frac{(x-t)^2}{2h^2}}$. Also as explained earlier that when the memory is reasonably large, the choice of $g_i(t)$ becomes not critical either. So here as an example and as default, we assume a normal distribution to approximate $g_i(t)$, i.e., $g_i(t) \approx \frac{1}{\sqrt{2\pi}\hat{\sigma}_i} e^{-\frac{(t-\hat{\mu}_i)^2}{2\hat{\sigma}_i^2}}$, where the mean $\hat{mu}_i$ and standard deviation $\hat{\sigma}_i$ can be computed from the CF of the subcluster, which is maintained in the CF tree. However, one can also assume other distributions to approximate $g_i(x)$, such as uniform distribution. As we will show experimentally later that when the memory size is reasonably large, the $g_i(t)$ format is no longer important in terms of accuracy.

DS	n	Description and $f(x)$
DS1	100000	Normal $f_1(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$
DS2	100000	Gamma(3) $f_2(x) = \frac{1}{2} x^2 e^{-x}$ for $x > 0$
DS3	100000	Normal $\oplus$ Gamma(3) $f_3(x) = 0.5 f_1(x) + 0.5 f_2(x-1)$
DS4	100000	Delta $\oplus$ Uniform $f_4(x) = 0.2u(0,10) + 0.08 \sum_{i=1}^{10} \delta(i)$

Table 1: Datasets Used

Now by replacing $K_h(x-t)$ and $g_i(t)$ in $CK_i(x)$ definition with our choices in this section, we can derive the $CK_i(x)$ into the form as shown in equ.(5).

$$CK_i(x) \approx \frac{1}{\sqrt{2\pi}\sqrt{\hat{\sigma}_i^2 + h^2}} e^{-\frac{(x-\hat{\mu}_i)^2}{2(\hat{\sigma}_i^2 + h^2)}} \quad (5)$$

It is interesting to note that $CK_i(x)$ is a new normal density function with the same mean $\hat{\mu}_i$, but the different standard deviation $\sqrt{\hat{\sigma}_i^2 + h^2}$. It is simple and can be calculated efficiently as "$exp(x)$" is supported in the math libraries of most programming languages such as C and FORTRAN.

3.2 Generalizing to High Dimensions

With x and t as vector variables instead of scalar variables, we can generalize our CF-kernel method to any dimensions under the condition that $K(x)$ and $g_i(x)$ are chosen of dimension orthogonal density functions.

4 Performance

The CF-kernel method still needs a scan of all data points ($O(n)$) to build the CF tree. The scalability and stability of inserting data into CF-tree has been studied in [ZRL96]. After that, compared with the kernel method, it improves the time/space complexity of storing and computing $\hat{f}_K(x)$ from n terms to m terms. In this section we will concentrate on studying the CF-kernel method in terms of its accuracy, as well as all the factors affecting its accuracy, such as memory size and $g_i(x)$ formats.

Datasets: Table 1 describes the sizes (n) and underlying density functions ($f(x)$) of the four distinct datasets we have used for performance studies. Here $u(0,10)$ represents the uniform density function defined on the interval $x \in [0,10]$, and $\delta(i)$ represents the delta density function defined at the point $x = i$.

CF Tree Setting: The parameter settings for scanning the data points and inserting them into the

DS	Kernel Time	Mem	CF-kernel Time	Mem	$\bar{D}(\hat{f})$
DS1	184041	800	80	40	4.9e-6
DS2	184927	800	91	40	1.5e-5
DS3	185411	800	76	40	1.0e-4
DS4	185197	800	72	40	2.9e-4

Table 2: $\hat{f_K}(x)$ versus $\hat{f_{CK}}(x)$

CF tree is similar to those used in Phase 1 of *BIRCH* [ZRL96] except that the outlier handling option is turned off for fair comparisons with the kernel method. Among them, the memory size is set to about 5% of the dataset size, this size is used as default unless otherwise stated in the experiments.

Metrics: To compare two density estimations on the same dataset, say $\hat{f_K}(x)$ and $\hat{f_{CK}}(x)$, we 1) take 1000 breakpoints uniformly from the whole data range; 2) define the **relative density difference** $D(\hat{f})$ as $\frac{2(\hat{f_K}(x)-\hat{f_{CK}}(x))}{(\hat{f_K}(x)+\hat{f_{CK}}(x))}$; 3) over the 1000 breakpoints, compute the average of the absolute values of $D(\hat{f})$'s, and denote it as $\bar{D}(\hat{f})$. This average difference is used as a simple indicator of overall difference between the two estimations. All the times are measured in seconds and memory are measured in kbytes.

Kernel versus CF-kernel: In this section we compare the performance of the kernel method and the CF-kernel method in terms of their running times, memory requirements, and estimation difference $\bar{D}(\hat{f})$. From Table 2, we can see that for all four datasets: 1) the kernel method runs much slower (more than 150 times) than the CF-kernel method does; 2) the kernel method uses much more memory (20 times) than the CF-kernel method does. However, the average difference $\bar{D}(\hat{f})$ between the kernel estimation and the CF-kernel estimation is very small and can be almost ignored. Fig. 2 plots the density estimation obtained with the CF-kernel method on the 1000 breakpoints over the data range for all four datasets. Visually, they look very similar to those obtained with the kernel method shown in fig. 1. So the CF-kernel method improves the time/space efficiency dramatically while keeping the estimation accuracy comparable with the kernel method.

$g_i(x)$ **Effects:** In this section, for all four datasets, we will compare two CF-kernel estimations with all other settings the same except that one assumes $g_i(x)$ as a normal distribution whereas the other assumes $g_i(x)$ as a normal distribution. Table 3 shows that: 1) The average difference $\bar{D}(\hat{f})$ between them is very close to zero. This confirms that, with respect to accuracy, assuming $g_i(x)$ as uniform or assuming $g_i(x)$

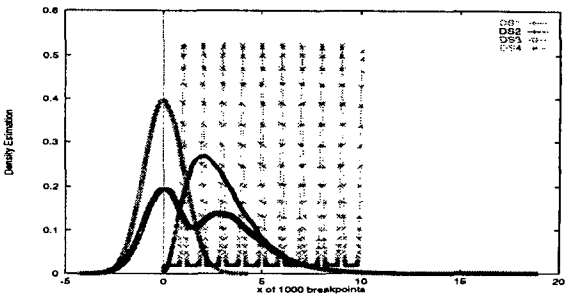

Figure 1: Kernel Estimation $\hat{f_K}(x)$

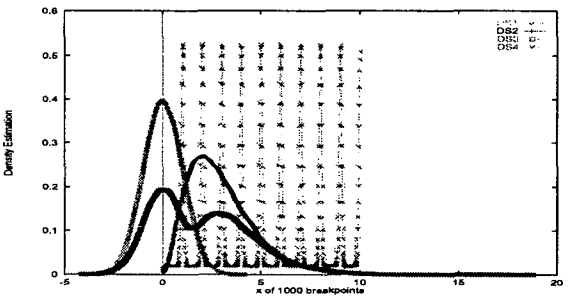

Figure 2: CF-kernel Estimation $\hat{f_{CK}}(x)$

as normal is not critical if the available memory is reasonably large. 2) The one assuming $g_i(x)$ as normal does run consistently faster than the one assuming $g_i(x)$ as uniform. This is due to the fact that assuming $g_i(x)$ as normal results in $CK(x)$ of a simplier form for computation.

Memory Effects: In this section we increase the memory size from 2 kbytes to 40 kbytes to study the effects of memory sizes in the *CF-kernel* method. As memory increases, the m (number of subclusters or leaf entries in the CF tree) increases. Fig. 3 and 4 shows how the running times scale up, and how the $\bar{D}(\hat{f})$ between the CF-kernel estimation and kernel estimation drops accordingly for all four datasets. We can see that as m increases, the running time increases almost linearly, whereas the $\bar{D}(\hat{f})$ drops to zero super linearly. So empirically we confirm that the CF-kernel method converges to the kernel method super linearly

DS	CF-kernel (uniform) Time	CF-kernel (normal) Time	$\bar{D}(\hat{f})$
DS1	104	80	6.48e-09
DS2	115	91	1.88e-08
DS3	92	76	8.56e-088
DS4	91	72	1.29e-06

Table 3: Uniform $g_i(x)$ versus Normal $g_i(x)$

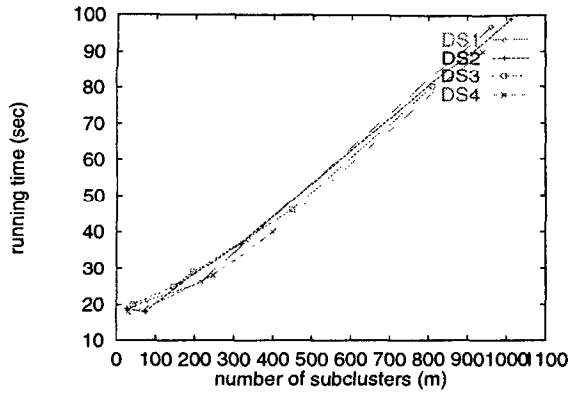

Figure 3: Running Time versus m

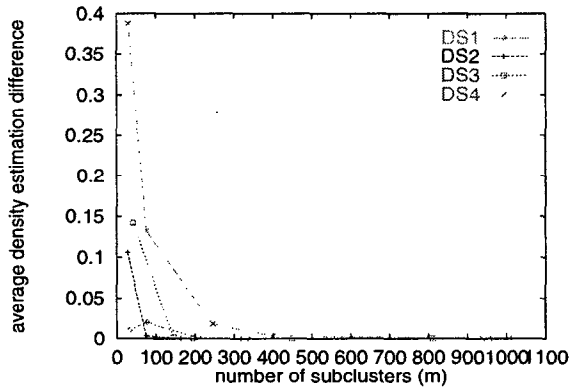

Figure 4: $\bar{D}(\hat{f})$ between $\hat{f_K}(x)$ and $\hat{f_{CK}}(x)$ versus m

as memory increases.

5 Compared with Histogram

Another widely-used nonparametric method for density estimation is the *histogram* method[Sil86, Dev87, WJ95]. Assume the data range is known, and cut into disjoint and contiguous intervals (i.e., bins of equal or different widths), the histogram estimation of $f(x)$ is defined as:

$$\hat{f_H}(x) = \frac{1}{n} \times \frac{number\ of\ X_i\ in\ the\ same\ bin\ as\ x}{width\ of\ the\ bin\ containing\ x} \quad (6)$$

The main drawbacks of the histogram method are: 1) curse of dimension[Sil86]; 2) the data range must be known in advance in order to allocate the bins effectively, so it is not an incremental method; 3) The discontinuity of $\hat{f_H}(x)$ makes the derivatives or other smoothing metrics unavailable; 4) the estimation accuracy depends not only on the bin widths but also on the bin locations.

Kernel method was proposed to try to avoid the problems encountered by histogram techniques. As mentioned in section 1, it has a lot of advantages over histogram techniques. In this paper, our major goals are 1) to establish the connection between the kernel theory and the CF tree as well as the BIRCH clustering algorithm; and 2) to improve the performance of the traditional kernel method. Our theoretical proof and performance comparisons between the traditional kernel method and the proposed CF-kernel method are provided. However the performance comparisons between the kernel method or the CF-kernel method and the histogram techniques are beyond the scope of this paper.

6 Conclusion

Density estimation for large datasets have become more and more interesting in database community, and one widely-studied method is the kernel method. However the *kernel* method does not scale up well for large datasets. In this paper we present the CF-kernel method which improves the time/space efficiency of the kernel method dramatically while keeping the estimation accuracy comparable to the kernel method. Theoretically, we 1) defined the CF-kernel estimation $\hat{f_{CK}}(x)$ and the CF-kernel function $CK_i(x)$, and 2) proved that with the distribution assumption in subcluster, the CF-kernel estimation and the kernel estimation are statistically equivalent. Empirically, we have shown that 1) the distribution in subcluster can be approximated by either uniform distribution or by normal distribution, and either way does not make much difference if the memory size is reasonably large; 2) the CF-kernel method converges to the kernel in a super linear speed as the memory size increases; 3) the CF-kernel method runs much faster than the kernel method, and uses much less memory.

References

[Dev87] Luc Devroye, *A Course in Density Estimation.* Birkhäuser Boston, 1987.

[ST87] David W. Scott, George R. Terrell, *Biased and Unbiased Cross-Validation in Density Estimation.* Journal of the Amer. Stat. Asso., Vol. 82, No. 400, p1131-1146, Dec., 1987.

[Sil86] B. W. Silverman, *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 1986.

[WJ95] M.P. Wand, M.C. Jones, *Kernel Smoothing.* Chapman and Hall, 1995.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, Miron Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases.* Proc. of ACM SIGMOD Int'l Conf. on Management of Data, p103-114, June 1996, Montreal, Canada.