Large Scale Kernel Machines

Editors:

Léon Bottou leon@bottou.org
NEC Labs America
Princeton, NJ 08540, USA

Olivier Chapelle chapelle@tuebingen.mpg.de
Max Planck Institure for Biological Cybernetics
72076 Tübingen, Germany

Denis DeCoste decosted@yahoo-inc.com
Yahoo! Research
Burbank, CA 91504, USA

Jason Weston jaseweston@gmail.com
NEC Labs America
Princeton, NJ 08540, USA

This is a draft containing only `raykar_chapter.tex` and an abbreviated front matter. Please check that the formatting and small changes have been performed correctly. Please verify the affiliation. Please use this version for sending us future modifications.

The MIT Press
Cambridge, Massachusetts
London, England

Contents

1 The Improved Fast Gauss Transform with Applications to Machine Learning **1**
1.1 Computational curse of non-parametric methods 1
1.2 Bottleneck computational primitive–Weighted superposition of kernels 2
1.3 Structured matrices and ϵ-exact approximation 4
1.4 Motivating example–polynomial kernel 5
1.5 Sum of Gaussian kernels–the discrete Gauss transform 6
1.6 Bringing computational tractability to the discrete Gauss transform 6
1.7 Multi-index notation 9
1.8 The improved fast Gauss transform 12
1.9 IFGT vs FGT 16
1.10 Numerical Experiments 19
1.11 Fast multivariate kernel density estimation 24
1.12 Conclusions 28

1 The Improved Fast Gauss Transform with Applications to Machine Learning

Vikas Chandrakant Raykar vikas@umiacs.umd.edu
Ramani Duraiswami ramani@umiacs.umd.edu
Department of Computer Science and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742, USA

In many machine learning algorithms and non-parametric statistics a key computational primitive is to compute the weighted sum of N Gaussian kernel functions at M points. The computational cost of the direct evaluation of such sums scales as $O(MN)$. In this chapter we describe some algorithms that allows one to compute the same sum in $O(M+N)$ time at the expense of reduced precision, which however can be arbitrary. Using these algorithms can significantly reduce the runtime of various machine learning procedures. In particular we discuss the improved fast Gauss transform algorithm in detail and compare it with the dual-tree algorithm of Gray and Moore (2003) and the fast Gauss transform of Greengard and Strain (1991). As an example we show how these methods can be used for fast multivariate kernel density estimation.

1.1 Computational curse of non-parametric methods

During the past few decades it has become relatively easy to gather huge amounts of data, which are often apprehensively called *massive data sets*. A few examples include datasets in genome sequencing, astronomical databases, internet databases, experimental data from particle physics, medical databases, financial records, weather reports, audio and video data. A goal in these areas is to build systems which can automatically extract

useful information from the raw data. *Learning* is a principled method for distilling *predictive* and therefore scientific theories from the data (Poggio and Smale, 2003).

The *parametric approach* to learning assumes a functional form for the model to be learnt, and then estimates the unknown parameters. Once the model has been trained *the training examples can be discarded.* The essence of the training examples have been captured in the model parameters, using which we can draw further inferences. However, unless the form of the model is known a priori, assuming it very often leads to erroneous inference. *Nonparametric methods* do not make any assumptions on the form of the underlying model. This is sometimes referred to as *'letting the data speak for themselves'* (Wand and Jones, 1995). A price to be paid is that all the available *data has to be retained* while making the inference. It should be noted that nonparametric does not mean a lack of parameters, but rather that the underlying function/model of a learning problem cannot be indexed with a finite number of parameters. The number of parameters usually grows with the size of the training data.

One of the major bottlenecks for successful inference using nonparametric methods is their computational complexity. Most state-of-the-art nonparametric machine learning algorithms have a computational complexity of either $O(N^2)$ or $O(N^3)$. This has seriously restricted the use of massive data sets. Current implementations can handle only a few thousands of training examples.

In the next section we identify the key computational primitive contributing to the $O(N^3)$ or $O(N^2)$ complexity. We discuss some algorithms that use ideas and techniques from computational physics, scientific computing, and computational geometry to speed up approximate calculation of these primitives to $O(N)$ and also provide *high accuracy guarantees.* In particular in section 1.8 we discuss the improved fast Gauss transform algorithm in detail and compare it with the dual-tree algorithm of Gray and Moore (2003) and the fast Gauss transform of Greengard and Strain (1991) (section 1.10). In section 1.11 we show how these methods can be used for fast multivariate kernel density estimation.

1.2 Bottleneck computational primitive–Weighted superposition of kernels

In most kernel based machine learning algorithms (Shawe-Taylor and Cristianini, 2004), Gaussian processes (Rasmussen and Williams, 2006), and nonparametric statistics (Izenman, 1991) the key computationally intensive task is to compute a linear combination of local kernel functions centered

on the training data, *i.e.*,

$$f(x) = \sum_{i=1}^{N} q_i k(x, x_i), \tag{1.1}$$

where,

- $\{x_i \in \mathbb{R}^d, i = 1, \ldots, N\}$ are the N training data points,
- $\{q_i \in \mathbb{R}, i = 1, \ldots, N\}$ are the appropriately chosen weights,
- $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the local kernel function,
- and $x \in \mathbb{R}^d$ is the test point at which f is to be computed.

The computational complexity to evaluate Equation 1.1 at a given test point is $O(N)$.

For kernel machines (e.g. regularized least squares (Poggio and Smale, 2003), support vector machines (Cristianini and Shawe-Taylor, 2000), kernel regression (Wand and Jones, 1995)) f is the regression/classification function. This is a consequence of the well known classical representer theorem (Wabha, 1990) which states that the solutions of certain risk minimization problems involving an empirical risk term and a quadratic regularizer can be written as expansions in terms of the kernels centered on the training examples. In case of Gaussian process regression (Williams and Rasmussen, 1996) f is the mean prediction. For non-parametric density estimation it is the kernel density estimate (Wand and Jones, 1995).

Training these models scales as $O(N^3)$ since most involve solving a linear system of equations of the form

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}, \tag{1.2}$$

where, $\mathbf{K}$ is the $N \times N$ Gram matrix where $[\mathbf{K}]_{ij} = k(x_i, x_j)$, λ is some regularization parameter or noise variance, and $\mathbf{I}$ is the identity matrix. For specific kernel methods then there are many published techniques for speeding things up. However a naive implementation would scale as $O(N^3)$.

Also many kernel methods in unsupervised learning like kernel principal component analysis (Smola et al., 1996), spectral clustering (Chung, 1997), and Laplacian eigenmaps involve computing the eigen values of the Gram matrix. Solutions to such problems can be obtained using iterative methods, where the dominant computation is evaluation of $f(x)$.

Recently, such nonparametric problems have been collectively referred to as *N-body problems in learning* by Gray and Moore (2001), in analogy with the coulombic, magnetostatic, and gravitational N-body potential problems arising in computational physics (Greengard, 1994), where all pairwise

interactions in a large ensemble of particles must be calculated.

1.3 Structured matrices and ϵ-exact approximation

In general we need to evaluate Equation 1.1 at M points $\{y_j \in \mathbb{R}^d, j = 1, \ldots, M\}$, *i.e.*,

$$f(y_j) = \sum_{i=1}^{N} q_i k(y_j, x_i) \;\; j = 1, \ldots, M, \tag{1.3}$$

leading to the quadratic $O(MN)$ cost. We will develop fast ϵ-exact algorithms that compute the sum 1.3 approximately in linear $O(M + N)$ time. The algorithm is ϵ-exact in the sense made precise below.

Definition 1.1. *For any given $\epsilon > 0$, $\widehat{f}$ is an $\epsilon - exact$ approximation to f if the maximum absolute error relative to the total weight $Q = \sum_{i=1}^{N} |q_i|$ is upper bounded by ϵ, i.e.,*

$$\max_{y_j} \left[\frac{|\hat{f}(y_j) - f(y_j)|}{Q} \right] \leq \epsilon. \tag{1.4}$$

The constant in $O(M + N)$, depends on the desired *accuracy* ϵ, which however can be *arbitrary*. In fact for machine precision accuracy there is no difference between the results of the direct and the fast methods.

The sum in equation 1.3 can be thought of as a *matrix-vector multiplication* $f = \mathbf{K}q$, where $\mathbf{K}$ is a $M \times N$ matrix the entries of which are of the form $[\mathbf{K}]_{ij} = k(y_j, x_i)$ and $q = [q_1, \ldots, q_N]^T$ is a $N \times 1$ column vector.

Definition 1.2. *A dense matrix of order $M \times N$ is called a* structured matrix *if its entries depend only on $O(M + N)$ parameters.*

Philosophically, the reason we will be able to achieve $O(M+N)$ algorithms to compute the matrix-vector multiplication is that the matrix $\mathbf{K}$ is a structured matrix, since all the entries of the matrix are determined by the set of $M + N$ points $\{x_i\}_{i=1}^{N}$ and $\{y_j\}_{i=1}^{M}$. If the entries of the of the matrix $\mathbf{K}$ were completely random than we could not do any better than $O(MN)$.

1.4 Motivating example–polynomial kernel

We will motivate the main idea using a simple polynomial kernel that is often used in kernel methods. The polynomial kernel of order p is given by

$$k(x, y) = (x \cdot y + c)^p. \tag{1.5}$$

Direct evaluation of the sum $f(y_j) = \sum_{i=1}^{N} q_i k(x_i, y_j)$ at M points requires $O(MN)$ operations. The reason for this is that for each term in the sum the x_i and y_j appear together and hence we have to do all pair-wise operations. We will compute the same sum in $O(M + N)$ time by *factorizing* the kernel and *regrouping* the terms. The polynomial kernel can be written as follows using the binomial theorem.

$$k(x, y) = (x \cdot y + c)^p = \sum_{k=0}^{p} \begin{pmatrix} p \\ k \end{pmatrix} (x \cdot y)^k c^{p-k}. \tag{1.6}$$

Also for simplicity let x and y be scalars, *i.e.*, $x, y \in \mathbb{R}$. As a result we have $(x \cdot y)^k = x^k y^k$. The multivariate case can be handled using multi-index notation and will be discussed later. So now the sum – after suitable *regrouping* – can be written as follows:

$$\begin{aligned} f(y_j) = \sum_{i=1}^{N} q_i \left[\sum_{k=0}^{p} c^{p-k} \begin{pmatrix} p \\ k \end{pmatrix} x_i^k y_j^k \right] &= \sum_{k=0}^{p} c^{p-k} \begin{pmatrix} p \\ k \end{pmatrix} y_j^k \left[\sum_{i=1}^{N} q_i x_i^k \right] \\ &= \sum_{k=0}^{p} c^{p-k} \begin{pmatrix} p \\ k \end{pmatrix} y_j^k M_k, \end{aligned} \tag{1.7}$$

where $M_k = \sum_{i=1}^{N} q_i x_i^k$, can be called the *moments*. The moments $M_0, \ldots, M_p$ can be precomputed in $O(pN)$ time. Hence f can be computed in linear $O(pN + pM)$ time. This is sometimes known as encapsulating information in terms of the moments. Also note that for this simple kernel the sum was computed exactly.

In general any kernel $k(x, y)$ can be expanded in some region as

$$k(x, y) = \sum_{k=0}^{p} \Phi_k(x) \Psi_k(y) + error, \tag{1.8}$$

where the function Φ_k depends only on x and Ψ_k on y. We call p, the *truncation number*–which has to be chosen such that the error is less than the desired accuracy ϵ. The fast summation – after suitable regrouping – is

of the form

$$f(y_j) = \sum_{k=0}^{p} A_k \Psi_k(y) + error, \tag{1.9}$$

where the moments A_k can be pre-computed as $A_k = \sum_{i=1}^{N} q_i \Phi_k(x_i)$. Using series expansions about a single point can lead to large truncation numbers. We need to organize the datapoints into different clusters using data-structures and use series expansion about the cluster centers. Also we need to give accuracy guarantees. So there are two aspects to this problem

1. Approximation theory → series expansions and error bounds.
2. Computational geometry → effective data-structures.

1.5 Sum of Gaussian kernels–the discrete Gauss transform

The most commonly used kernel function is the *Gaussian kernel*

$$k(x, y) = e^{-\|x-y\|^2/h^2}, \tag{1.10}$$

where h is called the *bandwidth* of the kernel. The bandwidth h controls the degree of smoothing, of noise tolerance, or of generalization. The sum of multivariate Gaussian kernels is known as the *discrete Gauss transform* in the scientific computing literature. More formally, for each *target point* $\{y_j \in \mathbb{R}^d\}_{j=1,\ldots,M}$ the discrete Gauss transform is defined as,

$$G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j-x_i\|^2/h^2}, \tag{1.11}$$

where $\{q_i \in \mathbb{R}\}_{i=1,\ldots,N}$ are the *source weights*, $\{x_i \in \mathbb{R}^d\}_{i=1,\ldots,N}$ are the *source points*, *i.e.*, the center of the Gaussians, and $h \in \mathbb{R}^+$ is the *source scale* or *bandwidth*. In other words $G(y_j)$ is the total weighted contribution at y_j of N Gaussians centered at x_i each with bandwidth h. The computational complexity to evaluate the discrete Gauss transform at M target points is $O(MN)$.

1.6 Bringing computational tractability to the discrete Gauss transform

Various strategies have been proposed to reduce the computational complexity of computing the sum of multivariate Gaussians. To simplify the exposition, in this section we assume $M = N$.

1.6.1 Methods based on sparse data-set representation

There are many strategies for specific problems which try to reduce this computational complexity by searching for a sparse representation of the data (Williams and Seeger, 2001; Smola and Bartlett, 2001; Fine and Scheinberg, 2001; Lee and Mangasarian, 2001; Lawrence et al., 2003; Csato and Opper, 2002; Tresp, 2000; Tipping, 2001; Snelson and Ghahramani, 2006). Most of these methods try to find a reduced subset of the original data-set using either random selection or greedy approximation. In these methods there is no guarantee on the approximation of the kernel matrix in a deterministic sense.

1.6.2 Binned Approximation based on the FFT

If the source points are on a evenly spaced grid then we can compute the Gauss transform at an even spaced grid exactly in $O(N \log N)$ using the fast Fourier transform (FFT). One of the earliest methods, especially proposed for univariate fast kernel density estimation was based on this idea (Silverman, 1982). For irregularly spaced data, the space is divided into boxes, and the data is assigned to the closest neighboring grid points to obtain grid counts. The Gauss transform is also evaluated at regular grid points. For target points not lying on the the grid the value is obtained by interpolation based on the values at the neighboring grid points. As a result there is usually no guaranteed error bound for these methods. Also another problem is that the number of grid points grows exponentially with dimension.

1.6.3 Dual-tree methods

The dual-tree methods (Gray and Moore, 2001, 2003) are based on space partitioning trees for both the source and target points. This method first builds a spatial tree – *kd*-trees or ball trees – on both the source and target points. Using the tree data structure distance bounds between nodes can be computed. The bounds can be tightened by recursing on both trees. An advantage of the dual-tree methods is that they work for all common kernel choices, not necessarily Gaussian. The dual-tree methods give good speed up only for small bandwidths. For moderate bandwidths they end up doing the same amount of work as the direct summation. These methods do give accuracy guarantees. The single tree version takes $O(N \log N)$ time while the dual-tree version is postulated to be $O(N)$.

1.6.4 Fast Gauss transform

The *Fast Gauss Transform* (FGT) is an $\epsilon - exact$ approximation algorithm that reduces the computational complexity to $O(N)$, at the expense of reduced precision. Given any $\epsilon > 0$, it computes an approximation $\hat{G}(y_j)$ to $G(y_j)$ such that the maximum absolute error relative to the total weight Q is upper bounded by ϵ. The constant depends on the desired precision, dimensionality of the problem, and the bandwidth.

The FGT is a special case of the more general *fast multipole methods* (Greengard and Rokhlin, 1987), adapted to the Gaussian potential. The fast multipole method has been called one of the ten most significant algorithms (Dongarra and Sullivan, 2000) in scientific computation discovered in the 20th century, and won its inventors, Vladimir Rokhlin and Leslie Greengard, the 2001 Steele prize. Originally this method was developed for the fast summation of the potential fields generated by a large number of sources (charges), such as those arising in gravitational or electrostatic potential problems, that are described by the Laplace equation in two or three dimensions (Greengard and Rokhlin, 1987). The expression for the potential of a source located at a point can be factored in terms of an expansion containing the product of *multipole* functions and *regular* functions. This lead to the name for the algorithm. Since then FMM has also found application in many other problems, for example, in electromagnetic scattering, radial basis function fitting, molecular and stellar dynamics, and can be viewed as a fast matrix-vector product algorithm for particular structured matrices.

The FGT was first proposed by Greengard and Strain (1991) and applied successfully to a few lower dimensional applications in mathematics and physics. It uses a local representation of the Gaussian based on conventional Taylor series, a far field representation based on Hermite expansion, and translation formula for conversion between the two representations. However the algorithm has not been widely used much in statistics, pattern recognition, and machine learning applications where higher dimensions occur commonly. An important reason for the lack of use of the algorithm in these areas is that the performance of the proposed FGT degrades exponentially with increasing dimensionality, which makes it impractical for the statistics and pattern recognition applications. The constant in the linear asymptotic cost $O(M + N)$ grows roughly as p^d, *i.e.*, exponential in the dimension d.

1.6.5 Improved fast Gauss transform

In this chapter we briefly describe an *improved fast Gauss transform* (IFGT) suitable for higher dimensions (See Raykar et al. (2005) for a longer detailed

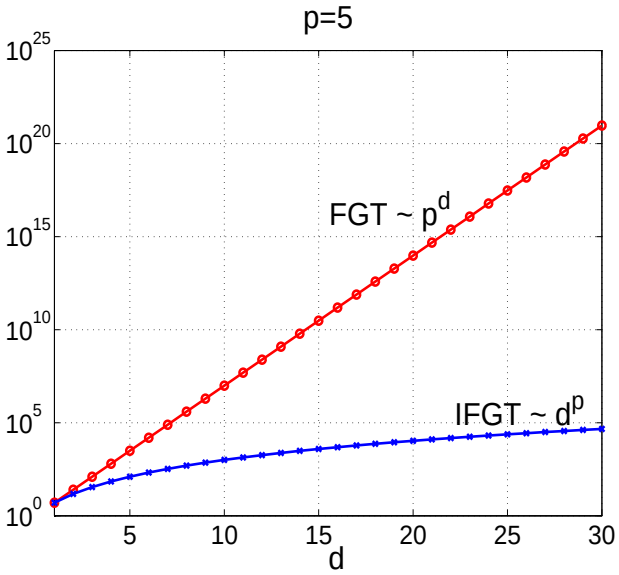

Figure 1.1: The constant term for the FGT and the IFGT complexity as a function of the dimensionality d. The FGT is exponential in d, while the IFGT is polynomial in d.

description). For the IFGT the constant term is asymptotically polynomial in d, *i.e*, is grows roughly as d^p (see Figure 1.1). The reduction is based on a new multivariate Taylor series expansion scheme combined with the efficient space subdivision using the k-center algorithm. The core IFGT algorithm was first presented in (Yang et al., 2003) and its use in kernel machines was shown in (Yang et al., 2005). More details of the algorithms including the automatic choice of the algorithm parameters and a tighter error bound can be found in (Raykar et al., 2005).

1.7 Multi-index notation

Before we present the IFGT algorithm we will discuss the notion of multi-indices, which will be useful later.

1. A multi-index $\alpha = (\alpha_1, \alpha_2 \ldots, \alpha_d) \in \mathbb{N}^d$ is a d-tuple of nonnegative integers.
2. The length of the multi-index α is defined as $|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_d$.
3. The factorial of α is defined as $\alpha! = \alpha_1!\alpha_2! \ldots \alpha_d!$.
4. For any multi-index $\alpha \in \mathbb{N}^d$ and $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ the *d-variate monomial* x^α is defined as $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d}$.

5. x^α is of *degree* n if $|\alpha| = n$.

6. The total number of d-variate monomials of degree n is $\binom{n+d-1}{d-1}$.

7. The total number of d-variate monomials of degree *less than or equal to* n is

$$r_{nd} = \sum_{k=0}^{n} \binom{k+d-1}{d-1} = \binom{n+d}{d}. \tag{1.12}$$

8. Let $x, y \in \mathbb{R}^d$ and $v = x \cdot y = x_1 y_1 + \ldots + x_d y_d$. Then using the multi-index notation v^n can be written as,

$$v^n = (x \cdot y)^n = \sum_{|\alpha|=n} \frac{n!}{\alpha!} x^\alpha y^\alpha. \tag{1.13}$$

1.7.1 Multivariate polynomial kernel

Using this notation the example which we discussed with polynomial kernel in section 1.4 can be extended to handle the multivariate case.

$$\begin{aligned}
f(y_j) &= \sum_{i=1}^{N} q_i (x_i \cdot y_j + c)^p = \sum_{i=1}^{N} q_i \left[\sum_{k=0}^{p} \binom{p}{k} c^{p-k} (x_i \cdot y_j)^k \right] \\
&= \sum_{i=1}^{N} q_i \left[\sum_{k=0}^{p} \binom{p}{k} c^{p-k} \left(\sum_{|\alpha|=k} \frac{k!}{\alpha!} x_i^\alpha y_j^\alpha \right) \right] = \sum_{i=1}^{N} q_i \left[\sum_{|\alpha|\le p} \frac{c^{p-|\alpha|} p!}{\alpha!(p-|\alpha|!)} x_i^\alpha y_j^\alpha \right] \\
&= \sum_{|\alpha|\le p} C_\alpha y_j^\alpha \left[\sum_{i=1}^{N} q_i x_i^\alpha \right] \quad \text{where, } C_\alpha = \frac{c^{p-|\alpha|} p!}{\alpha!(p-|\alpha|!)} \\
&= \sum_{|\alpha|\le p} C_\alpha y_j^\alpha M_\alpha \quad \text{where, } M_\alpha = \sum_{i=1}^{N} q_i x_i^\alpha.
\end{aligned} \tag{1.14}$$

The number of d-variate monomials of degree $|\alpha| \le p$ is $r_{pd} = \binom{p+d}{d}$. Hence the moments M_α can be computed in $O(r_{pd}N)$ time. Note that the computation of M_α depends on the x_i alone which can be done once and reused in each evaluation of $f(y_j)$. Hence the sum f can be computed at M points in $O(r_{pd}(M+N))$ time. Also note that we need r_{pd} space to store the moments.

Using the Sterling's formula $n! \approx \sqrt{2\pi n} e^{-n} n^n$, the term r_{pd} can be simplified as follows

$$r_{pd} = \binom{p+d}{d} = \frac{(p+d)!}{d!p!} \approx \left(1 + \frac{d}{p}\right)^p \left(1 + \frac{p}{d}\right)^d. \tag{1.15}$$

If $d \gg p$, r_{pd} is roughly polynomial in d, *i.e.*, d^p. While not as bad as exponential growth in d, still the constant can become large for high d. This growth of the constant with d is one of the major limitations of series based

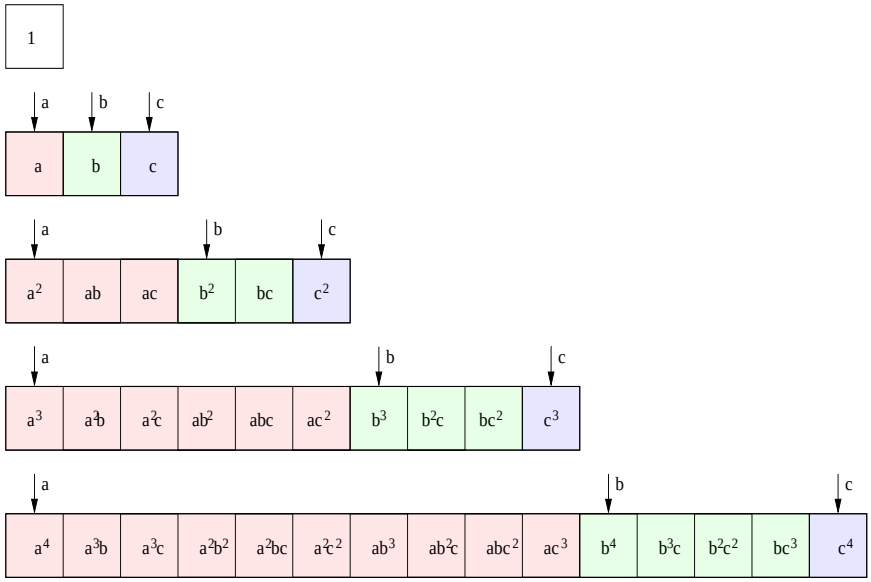

Figure 1.2: Efficient expansion of multivariate polynomials.

methods, which prevents them from being practical for high dimensional datasets. For example, when $p = 5$ and $d = 10$, $r_{pd} = 53,130$.

1.7.2 Efficient expansion of multivariate polynomials–Horner's rule

Evaluating each d-variate monomial of degree n directly requires n multiplications. Hence direct evaluation of all d-variate monomials of degree less than or equal to n requires $\sum_{k=0}^{n} k\binom{k+d-1}{d-1}$ multiplications. The storage requirement is $r_{nd} = \binom{n+d}{d}$. However, efficient evaluation using Horner's rule requires $r_{nd} - 1$ multiplications. The required storage is r_{nd}.

For a d-variate polynomial of order n, we can store all terms in a vector of length r_{nd}. Starting from the order zero term (constant 1), we take the following approach. Assume we have already evaluated terms of order $k-1$. We use an array of size d to record the positions of the d leading terms (the simple terms such as a^{k-1}, b^{k-1}, c^{k-1}, . . . in Figure 1.2) in the terms of order $k-1$. Then terms of order k can be obtained by multiplying each of the d variables with all the terms between the variables leading term and the end, as shown in the Figure 1.2 The positions of the d leading terms are updated respectively. The required storage is r_{nd} and the computations of the terms require $r_{nd} - 1$ multiplications.

1.8 The improved fast Gauss transform

For any point $x_* \in \mathbf{R}^d$ the Gauss Transform at y_j can be written as,

$$\begin{aligned} G(y_j) &= \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2}, \\ &= \sum_{i=1}^{N} q_i e^{-\|(y_j - x_*) - (x_i - x_*)\|^2/h^2}, \\ &= \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} e^{-\|y_j - x_*\|^2/h^2} e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2}. \end{aligned} \quad (1.16)$$

In Equation 1.16 the first exponential inside the summation $e^{-\|x_i - x_*\|^2/h^2}$ depends only on the source coordinates x_i. The second exponential $e^{-\|y_j - x_*\|^2/h^2}$ depends only on the target coordinates y_j. However for the third exponential $e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2}$ the source and target are entangled. The crux of the algorithm is to separate this entanglement via Taylor series.

1.8.1 Factorization

The p-term truncated Taylor series expansion for $e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2}$ can be written as (Raykar et al., 2005, Corollary 2),

$$e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2} = \sum_{n=0}^{p-1} \frac{2^n}{n!} \left[\left(\frac{y_j - x_*}{h} \right) \cdot \left(\frac{x_i - x_*}{h} \right) \right]^n + error_p. \quad (1.17)$$

The truncation number p is chosen based on the prescribed error ϵ. Using the multi-index notation (Equation 1.13), this expansion can be written as,

$$e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2} = \sum_{|\alpha| \le p-1} \frac{2^\alpha}{\alpha!} \left(\frac{y_j - x_*}{h} \right)^\alpha \left(\frac{x_i - x_*}{h} \right)^\alpha + error_p. \quad (1.18)$$

Ignoring error terms for now $G(y_j)$ can be approximated as,

$$\hat{G}(y_j) = \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} e^{-\|y_j - x_*\|^2/h^2} \left[\sum_{|\alpha| \le p-1} \frac{2^\alpha}{\alpha!} \left(\frac{y_j - x_*}{h} \right)^\alpha \left(\frac{x_i - x_*}{h} \right)^\alpha \right]. \quad (1.19)$$

1.8.2 Regrouping

Rearranging the terms Equation 1.19 can be written as

$$\begin{aligned} \hat{G}(y_j) &= \sum_{|\alpha| \le p-1} \left[\frac{2^\alpha}{\alpha!} \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} \left(\frac{x_i - x_*}{h} \right)^\alpha \right] e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^\alpha, \\ &= \sum_{|\alpha| \le p-1} C_\alpha e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^\alpha, \end{aligned} \quad (1.20)$$

where,

$$C_\alpha = \frac{2^\alpha}{\alpha!} \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} \left(\frac{x_i - x_*}{h}\right)^\alpha. \tag{1.21}$$

The coefficients C_α can be evaluated separately in $O(N)$. Evaluation of $\widehat{G}_r(y_j)$ at M points is $O(M)$. Hence the computational complexity has reduced from the quadratic $O(NM)$ to the linear $O(N + M)$. A detailed analysis of the computational complexity is provided later.

1.8.3 Space subdivision

Thus far, we have used the Taylor series expansion about a certain point x_*. However if we use the same x_* for all the points we typically would require very high truncation number since the Taylor series is valid only in a small open ball around x_*. We use an data adaptive space partitioning scheme–the farthest point clustering algorithm–to divide the N sources into K spherical clusters, S_k for $k = 1, \ldots, K$ with c_k being the center of each cluster. The Gauss transform can be written as,

$$\hat{G}(y_j) = \sum_{k=1}^{K} \sum_{|\alpha| \le p-1} C_\alpha^k e^{-\|y_j - c_k\|^2/h^2} \left(\frac{y_j - c_k}{h}\right)^\alpha, \tag{1.22}$$

where,

$$C_\alpha^k = \frac{2^\alpha}{\alpha!} \sum_{x_i \in S_k} q_i e^{-\|x_i - c_k\|^2/h^2} \left(\frac{x_i - c_k}{h}\right)^\alpha. \tag{1.23}$$

We model the space subdivision task as a k-center problem, which is defined as follows: Given a set of N points in d dimensions and a predefined number of the clusters k, find a partition of the points into clusters $S_1, \ldots, S_k$, and also the cluster centers $c_1, \ldots, c_k$, so as to minimize the cost function-the maximum radius of clusters, $\max_i \max_{x \in S_i} \|x - c_i\|$.

The k-center problem is known to be NP-hard (Bern and Eppstein, 1997). Gonzalez (1985) proposed a very simple greedy algorithm, called *farthest-point clustering*, and proved that it gives an approximation factor of 2. This algorithm works as follows– Pick an arbitrary point v_0 as the center of the first cluster and add it to the center set C. Then for $i = 1$ to k do the following: at step i, for every point v, compute its distance to the set C: $d_i(v, C) = \min_{c \in C} \|v - c\|$. Let v_i be the point that is farthest from C, i.e., the point for which $d_i(v_i, C) = \max_v d_i(v, C)$. Add v_i to set C. Report the points $v_0, v_1, \ldots, v_{k-1}$ as the cluster centers. Each point is assigned to its nearest center.

The direct implementation of farthest-point clustering has running time

$O(Nk)$. Feder and Greene (1988) gave a two-phase algorithm with optimal running time $O(N \log k)$. Figure 1.3 displays the results of farthest-point algorithm on a sample two dimensional data-set.

1.8.4 Rapid decay of the Gaussian

Since the Gaussian decays very rapidly a further speed up is achieved if we ignore all sources belonging to a cluster if the cluster is greater than a certain distance from the target point, $\|y_j - c_k\| > r_y^k$. The cluster cutoff radius depends on the desired precision ϵ. So now the Gauss transform is evaluated as

$$\hat{G}(y_j) = \sum_{\|y_j - c_k\| \leq r_y^k} \sum_{|\alpha| \leq p-1} C_\alpha^k e^{-\|y_j - c_k\|^2/h^2} \left(\frac{y_j - c_k}{h} \right)^\alpha, \tag{1.24}$$

where,

$$C_\alpha^k = \frac{2^\alpha}{\alpha!} \sum_{x_i \in S_k} q_i e^{-\|x_i - c_k\|^2/h^2} \left(\frac{x_i - c_k}{h} \right)^\alpha. \tag{1.25}$$

1.8.5 Runtime analysis

1. The farthest point clustering has a running time $O(N \log K)$ (Feder and Greene, 1988).

2. Since each source point belongs to only one cluster computing the cluster coefficients C_α^k for all the clusters is of $O(Nr_{(p-1)d})$, where $r_{(p-1)d} = \binom{p+d-1}{d}$ is the total number of d-variate monomials of degree less than or equal to $p-1$.

3. Computing $\hat{G}(y_j)$ is $O(Mnr_{(p-1)d})$ where n is the maximum number of neighbor clusters (depends on the bandwidth h and the error ϵ) which influence the target. It does not take into account the cost needed to determine n. This involves looping through all K clusters and computing the distance between each of the M test points and each of the K cluster centers, resulting in an additional $O(MK)$ term. This term can be reduced to $O(M \log K)$ if efficient nearest neighbor search techniques are used, and this is a matter of current research.

Hence the total time is

$$O(N \log K + Nr_{(p_{max}-1)d} + Mnr_{(p_{max}-1)d} + KM). \tag{1.26}$$

Assuming $M = N$, the time taken is $O(\left[\log K + (1+n)r_{(p_{max}-1)d} + K\right] N)$. The constant term depends on the dimensionality, the bandwidth, and the

accuracy required. The number of terms $r_{(p-1)d}$ is asymptotically polynomial in d. For $d \to \infty$ and moderate p, the number of terms is approximately d^p.

For each cluster we need to store $r_{(p-1)d}$ coefficients. Accounting for the initial data the storage complexity is $O(Kr_{(p-1)d} + N + M)$.

1.8.6 Choosing the parameters

Given any $\epsilon > 0$, we want to choose the following parameters,

1. K (the number of clusters),
2. p (the truncation number), and
3. $\{r_y^k\}_{k=1}^K$ (the cut off radius for each cluster),

such that for any target point y_j we can guarantee that $|\hat{G}(y_j) - G(y_j)|/Q \leq \epsilon$, where $Q = \sum_{i=1}^N |q_i|$. In order to achieve this we will use a upper bound for the actual error and choose the parameters based on this bound. Deriving tight error bounds is one of the trickiest and crucial part for any series based algorithm. Also the parameters have to be chosen automatically without any user intervention. The user just specifies the accuracy ϵ.

A criticism (Lang et al., 2005) of the original IFGT (Yang et al., 2005) was that the error bound was too pessimistic, and too many computational resources were wasted as a consequence. Further, the choice of the parameters was not automatic. An automatic way for choosing the parameters along with tighter error bounds is described in Raykar et al. (2005).

Let us define Δ_{ij} to be the error in $\widehat{G}(y_j)$ contributed by the i^{th} source x_i. We now require that

$$|\hat{G}(y_j) - G(y_j)| = \left| \sum_{i=1}^N \Delta_{ij} \right| \leq \sum_{i=1}^N |\Delta_{ij}| \leq Q\epsilon = \sum_{i=1}^N |q_i|\epsilon. \tag{1.27}$$

One way to achieve this is to let $|\Delta_{ij}| \leq |q_i|\epsilon \ \forall i = 1, \ldots, N$. Let c_k be the center of the cluster to which x_i belongs. There are two different ways in which a source can contribute to the error.

1. The first is due to ignoring the cluster S_k if it is outside a given radius r_y^k from the target point y_j. In this case,

$$\Delta_{ij} = q_i e^{-\|y_j - x_i\|^2/h^2} \text{ if } \|y_j - c_k\| > r_y^k. \tag{1.28}$$

2. The second source of error is due to truncation of the Taylor series. For all clusters which are within a distance r_y^k from the target point the error is due to the truncation of the Taylor series after order p. From Equations 1.16

and 1.17 we have,

$$\Delta_{ij} = q_i e^{-\|x_i - c_k\|^2/h^2} e^{-\|y_j - c_k\|^2/h^2} error_p \text{ if } \|y_j - c_k\| \leq r_y^k. \quad (1.29)$$

Our strategy for choosing the parameters is as follows. The cutoff radius r_y^k for each cluster is chosen based on Equation 1.28 and the radius of each cluster r_x^k. Given r_y^k and r_x^k the truncation number p is chosen based on Equation 1.29 and a bound on $error_p$. More details can be seen in Raykar et al. (2005). Algorithm 1 summarizes the procedure to choose the IFGT parameters. The IFGT is summarized in Algorithm 2.

Algorithm 1: Choosing the parameters for the IFGT

Input : d (dimension)
h (bandwidth)
ϵ (error)
N (number of sources)

Output: K (number of clusters)
r (cutoff radius)
p (truncation number)

Define
$\delta(p,a,b) = \frac{1}{p!}\left(\frac{2ab}{h^2}\right)^p e^{-(a-b)^2/h^2}$, $b_*(a,p) = \frac{a+\sqrt{a^2+2ph^2}}{2}$, and $r_{pd} = \binom{p-1+d}{d}$;

Choose the cutoff radius $r \leftarrow \min(\sqrt{d}, h\sqrt{\ln(1/\epsilon)})$;

Choose $K_{limit} \leftarrow \min\left(\lceil 20\sqrt{d}/h \rceil, N\right)$ (a rough bound on K);

for $k \leftarrow 1 : K_{limit}$ **do**
  compute an estimate of the maximum cluster radius as $r_x \leftarrow k^{-1/d}$;
  compute an estimate of the number of neighbors as $n \leftarrow \min\left((r/r_x)^d, k\right)$;
  choose $p[k]$ such that $\delta(p = p[k], a = r_x, b = \min\left[b_*(r_x, p[k]), r + r_x\right]) \leq \epsilon$;
  compute the constant term $c[k] \leftarrow k + \log k + (1+n) r_{(p[k]-1)d}$
end
choose $K \leftarrow k_*$ for which $c[k_*]$ is minimum. $p \leftarrow p[k_*]$.

1.9 IFGT vs FGT

The FGT (Greengard and Strain, 1991) is a special case of the more general single level fast multipole method (Greengard and Rokhlin, 1987), adapted to the Gaussian potential. The first step of the FGT is the spatial subdivision of the unit hypercube into N_{side}^d boxes of side $\sqrt{2}rh$ where $r < 1/2$. The sources and targets are assigned to different boxes. Given the sources in one box and the targets in a neighboring box, the computation is performed using one of the following four methods depending on the number of sources

Algorithm 2: The improved fast Gauss transform.

Input :

$x_i \in \mathbf{R}^d$ $i = 1, \ldots, N$ /* N sources in d dimensions. */
$q_i \in \mathbf{R}$ $i = 1, \ldots, N$ /* source weights. */
$h \in \mathbf{R}^+$ $i = 1, \ldots, N$ /* source bandwidth. */
$y_j \in \mathbf{R}^d$ $j = 1, \ldots, M$ /* M targets in d dimensions. */
$\epsilon > 0$ /* Desired error. */

Output: Computes an approximation $\hat{G}(y_j)$ to $G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2}$ such that the $|\hat{G}(y_j) - G(y_j)| \leq Q\epsilon$, where $Q = \sum_{i=1}^{N} |q_i|$.

Step 0 *Define* $\delta(p, a, b) = \frac{1}{p!}\left(\frac{2ab}{h^2}\right)^p e^{-(a-b)^2/h^2}$ *and* $b_*(a, p) = \frac{a+\sqrt{a^2+2ph^2}}{2}$.

Step 1 *Choose the cutoff radius r, the number of clusters K, and the truncation number p using Algorithm 1.*

Step 2 *Divide the N sources into K clusters, $\{S_k\}_{k=1}^{K}$, using the Feder and Greene's farthest-point clustering algorithm. Let c_k and r_x^k be the center and radius respectively of the k^{th} cluster. Let $r_x = \max_k\left(r_x^k\right)$ be the maximum cluster radius.*

Step 3 *Update the truncation number based on the actual r_x, i.e., choose p such that $\delta(p, a = r_x, \min\left[b_*(r_x, p), r + r_x\right]) \leq \epsilon$.*

Step 3 *For each cluster S_k with center c_k compute the coefficients C_α^k.*
$C_\alpha^k = \frac{2^\alpha}{\alpha!}\sum_{x_i \in S_k} q_i e^{-\|x_i - c_k\|^2/h^2}\left(\frac{x_i - c_k}{h}\right)^\alpha \quad \forall |\alpha| \leq p-1.$

Step 4 *For each target y_j the discrete Gauss transform is evaluated as*
$\hat{G}(y_j) = \sum_{\|y_j - c_k\| \leq r + r_x^k} \sum_{|\alpha| \leq p-1} C_\alpha^k e^{-\|y_j - c_k\|^2/h^2}\left(\frac{y_j - c_k}{h}\right)^\alpha.$

and targets in these boxes: Direct evaluation is used if the number of sources and targets are small (in practice a cutoff of the order $O(p^{d-1})$ is introduced.). If the sources are clustered in a box then they can are transformed into a *Hermite expansion* about the center of the box. This expansion is directly evaluated at each target in the target box if the number of the targets is small. If the targets are clustered then the sources or their expansion are converted to a local *Taylor series* which is then evaluated at each target in the box. Since the Gaussian decays very rapidly only a few neighboring source boxes will have influence on the target box. If boxes are too far apart than the contribution will be negligible and the computation is not performed.

There are three reasons contributing to the degradation of the FGT in higher dimensions:

1. The number of the terms in the Hermite expansion used by the FGT grows exponentially with dimensionality, which causes the constant factor associated with the asymptotic complexity $O(M + N)$ to increase exponen-

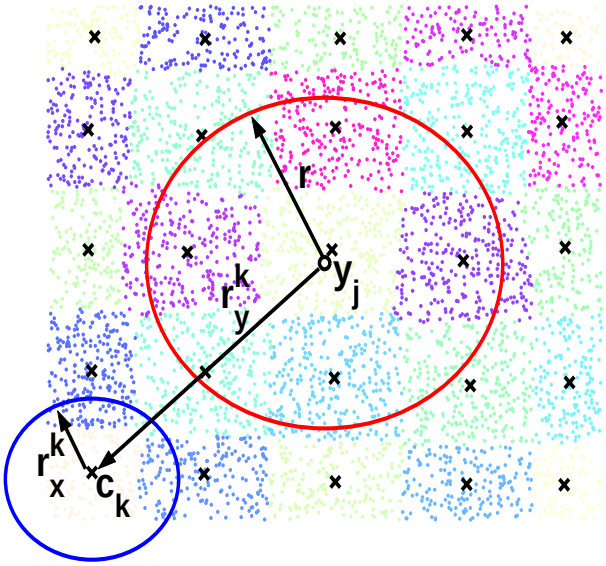

Figure 1.3: Schematic of the evaluation of the improved fast Gauss transform at the target point y_j. For the source points the results of the farthest point clustering algorithm are shown along with the center of each cluster. A set of coefficients are stored at the center of each cluster. Only the influential clusters within radius r of the target point are used in the final summation.

tially with dimensionality.

2. The constant term due to the translation of the far-field Hermite series to the local Taylor series grows exponentially fast with dimension making it impractical for dimensions greater than three.

3. The space subdivision scheme used by the fast Gauss transform is a uniform box subdivision scheme which is tolerable in lower dimensions but grows exponentially in higher dimensions. The number of nearby boxes will be large, and the boxes will mostly be empty.

The IFGT differs from the FGT in the following three ways, addressing each of the issues above.

1. A single multivariate Taylor series expansion for a factored form of the Gaussian is used to reduce the number of the expansion terms to polynomial order.

2. The expansion acts both as a far-field and local expansion. As a result we do not have separate far-field and local expansions which eliminates the cost of translation (See Figure 1.4), while achieving quickly convergent expansions in both domains.

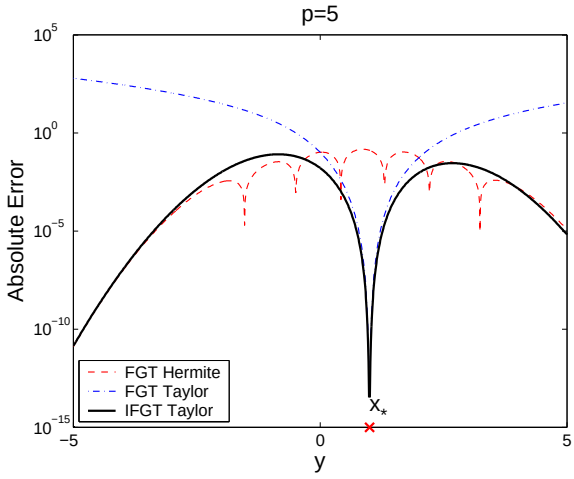

Figure 1.4: The absolute value of the actual error between the one dimensional Gaussian ($e^{-(x_i-y)/h^2}$) and different series approximations. The Gaussian was centered at $x_i = 0$ and $h = 1.0$. All the series were expanded about $x_* = 1.0$. $p = 5$ terms were retained in the series approximation.

3. The k-center algorithm is applied to subdivide the space using overlapping spheres which is more efficient in higher dimensions.

1.10 Numerical Experiments

We compare the following four methods

Direct	Naive $O(N^2)$ implementation of the Gauss transform.
FGT	The fast Gauss Transform as described in Greengard and Strain (1991).
IFGT	The improved fast Gauss transform.
Dual-tree	The kd-tree dual-tree algorithm of Gray and Moore (2003).

All the algorithms were programmed in C++ or C with MATLAB bindings and were run on a 1.83 GHz processor with 1 GB of RAM. See Appendix 1.A for the source of different software.

1.10.1 Speed up as a function of N

We first study the performance as the function of N for $d = 3$. N points were uniformly distributed in a unit cube. The Gauss transform was evaluated at $M = N$ points uniformly distributed in the unit cube. The weights

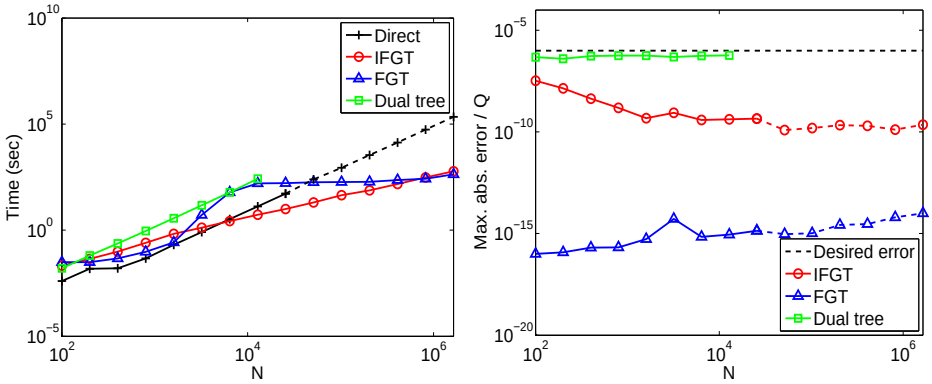

Figure 1.5: *Scaling with N* The running times and the actual error for the different methods as a function of N. [$\epsilon = 10^{-6}$, $h = 0.25$, and $d = 3$]

q_i were uniformly distributed between 0 and 1. The parameters for the algorithms were automatically chosen without any user intervention. The target error was set to 10^{-6}. Figure 1.5 shows the results for all the various methods as a function of N for bandwidth $h = 0.25$. For $N > 25,600$ the timing results for the direct evaluation were obtained by evaluating the Gauss transform at $M = 100$ points and then extrapolating the results. The following observations can be made.

1. For the IFGT the computational cost is linear in N.

2. For the FGT the cost grows linearly only after a large N when the linear term $O(p^d N)$ dominates the initial cost of Hermite-Taylor translation. This jump in the performance can also be seen in the original FGT paper (See Tables 2 and 4 in Greengard and Strain (1991)).

3. The IFGT shows a better speedup than the FGT. However the FGT finally catches up with IFGT (*i.e.* the asymptotic performance starts dominating) and shows a speedup similar to that of the IFGT. However this happens typically after a very large N which increases with the dimensionality of the problem.

4. The IFGT error is closer to the target than is the FGT.

5. The *kd*-tree algorithm appears to be doing $O(N^2)$ work. Also it takes much more time than the direct evaluation probably because of the time taken to build up the *kd*-trees. The dual-tree algorithms show good speedups only at very small bandwidths (See also Figure 1.7).

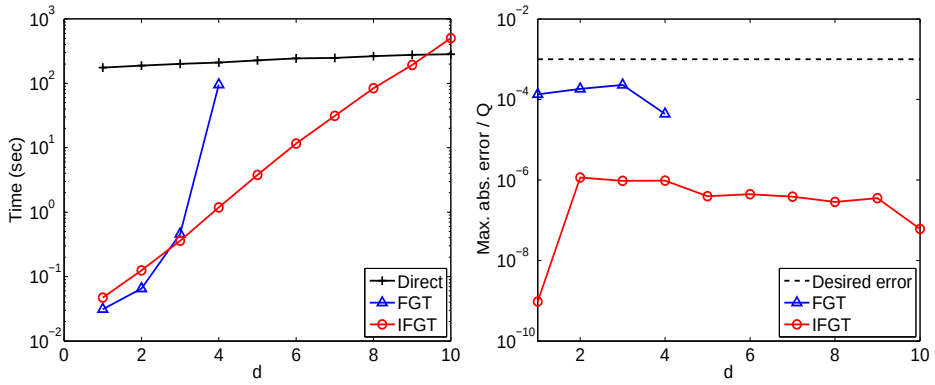

Figure 1.6: *Scaling with dimensionality* d The running times and the actual error for the different methods as a function of the dimension d. [$\epsilon = 10^{-3}$, $h = 1.0$, and $N = M = 50,000$]

1.10.2 Speed up as a function of d

The main advantage of the IFGT is in higher dimensions where we can no longer run the FGT algorithm. Figure 1.6 shows the performance for a fixed $N = M = 50,000$ as a function of d for a fixed bandwidth of $h = 1.0$.

1. The FGT becomes impractical after three dimensions with the cost of translation increasing with dimensionality. The FGT gave good speedup only for $d \leq 4$.

2. For the IFGT, as d increases the crossover point (*i.e*, the N after which IFGT shows a better performance than the direct) increases. For $N = M = 50,000$ we were able to achieve good speedups till $d = 10$.

3. The *kd*-tree method could not be run for the bandwidth chosen.

1.10.3 Speed up as a function of the bandwidth h

One of the important concerns for N-body algorithms is their scalability with bandwidth h. Figure 1.7 shows the performance of the IFGT and the dual-tree algorithm as a function of the bandwidth h. The other parameters were fixed at $N = M = 7,000$, $\epsilon = 10^{-3}$, and $d = 2, 3, 4,$ and, 5.

1. The general trend is that IFGT shows better speedups for large bandwidths while the dual-tree algorithm performs better at small bandwidths.

2. At large bandwidths the dual-tree algorithms ends up doing the same amount of work as the direct implementation. The dual-tree appears to take larger time than the direct probably because of the time taken to build up the *kd*-trees.

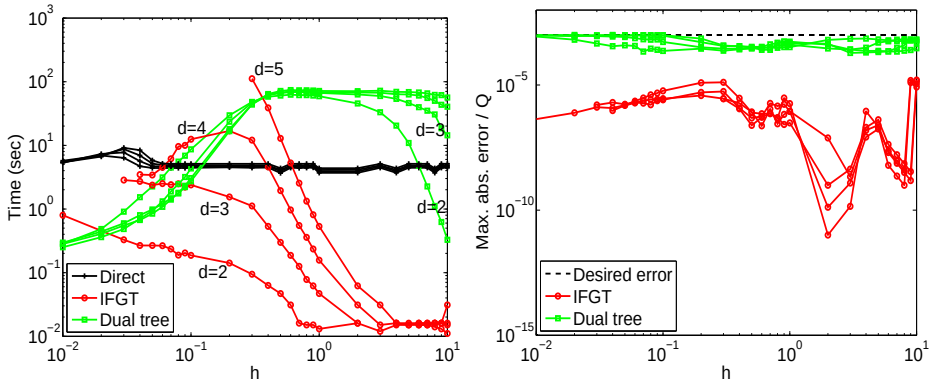

Figure 1.7: *Scaling with bandwidth h* The running times and the actual error for the different methods as a function of the bandwidth h. [$\epsilon = 10^{-3}$ and $N = M = 7,000$]

3. There is a cutoff bandwidth h_c above which the IFGT performs better than the dual-tree algorithm. This cutoff increases as the dimensionality of the datapoints increases.

For small bandwidths the number of clusters and the truncation number required by the IFGT is high. The space-subdivision employed by the IFGT algorithm is not hierarchical. As a result nearest influential clusters cannot be searched effectively. When the number of clusters K is large we end up doing a brute force search over all the clusters. In such case it may be more efficient to directly evaluate the contribution from its neighbors within a certain radius. This is exactly what the dual-tree algorithms do, albeit in a more sophisticated way. The dual-tree algorithms suffer at large bandwidths because they do not use series expansions. It would be interesting to combine the series expansions of the IFGT with the kd-tree data structures to obtain an algorithm which would perform well at all bandwidths. A recent attempt using the FGT expansions was made by Lee et al. (2006). Since they used FGT series expansion the algorithm was not practical for $d > 3$ due to the high translation costs.

1.10.4 Bandwidth as a function of d

It should be noted that IFGT and FGT show good speedups especially for large bandwidths. Figure 1.8 shows the performance for a fixed $N = M = 20,000$ as a function of d. In this case for each dimension we set the bandwidth $h = 0.5\sqrt{d}$ (Note that $\sqrt{d}$ is the length of the diagonal of a unit hypercube). The bandwidth of this order is sometimes used in high dimensional data in some machine learning tasks. With h varying with dimension

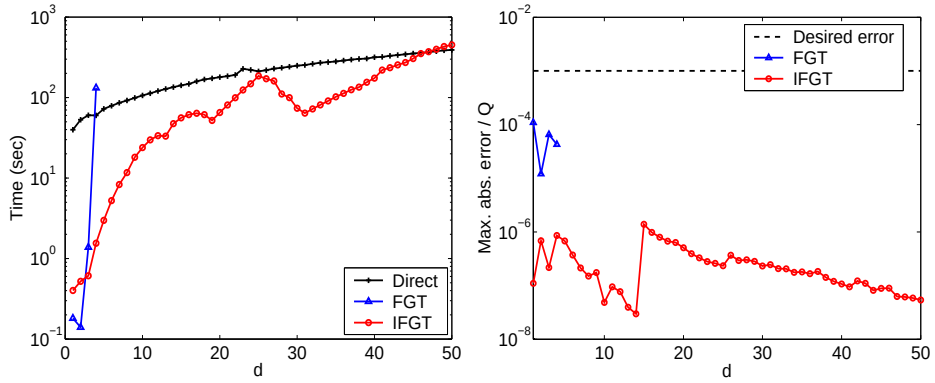

Figure 1.8: *Effect of bandwidth scaling as* $h = 0.5\sqrt{d}$ The running times and the actual error for the different methods as a function of the dimension d. The bandwidth was $h = 0.5\sqrt{d}$. [$\epsilon = 10^{-3}$ and $N = M = 20,000$]

we were able to run the algorithm for arbitrary high dimensions. The dual-tree algorithm took more than the direct method for such bandwidths and could not be run for such large dataset.

1.10.5 Structured data

Until now we showed results for the worst case scenario–data uniformly distributed in a unit hypercube. However if there is structure in the data, *i.e.*, the data is either clustered or lie on some smooth lower dimensional manifold, then the algorithms show much better speed up. Figure 1.9 compares the time taken by the IFGT and dual tree methods as a function of h for four different scenarios:

1. Both source and target points are uniformly distributed.
2. Source points are clumpy while the target points are uniform.
3. Source points uniformly distributed while the target points are clumpy.
4. Both source and target points are clumpy.

The clumpy data was generated from a mixture of 10 Gaussians. The following observations can be made:

1. For the dual tree method clumpiness either in source or target points gives better speedups.

2. For the IFGT clumpiness in source points gives a much better speed up than uniform distribution. Clumpiness in target points does not matter since IFGT clusters only the source points.

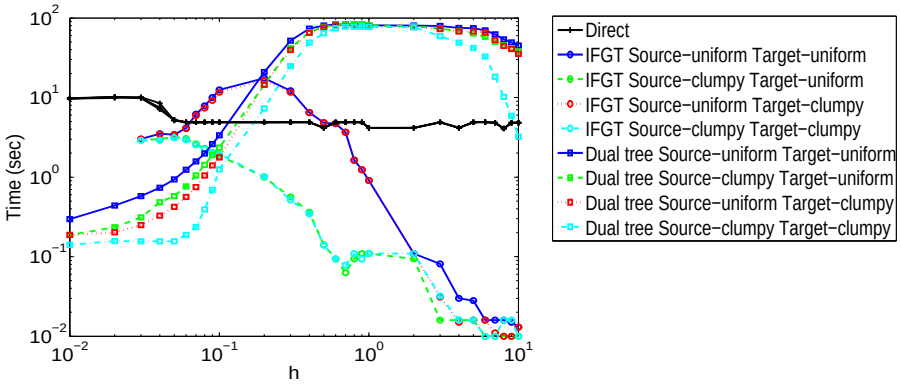

Figure 1.9: *Effect of clustered data* The running times for the different methods as a function of the bandwidth h. [$\epsilon = 10^{-3}$, $N = M = 7,000$, and $d = 4$]. The source and target points were either uniformly distributed (uniform) or drawn from mixture of 10 Gaussians (clumpy).

1.11 Fast multivariate kernel density estimation

The IFGT and other N-body algorithms can be used in any scenario where we encounter sums of Gaussians. A few applications would include kernel density estimation (Yang et al., 2003), prediction in SVMs, and mean prediction in Gaussian process regression. For training, the IFGT can also be embedded in a conjugate-gradient or any other suitable optimization procedure (Yang et al., 2005; Shen et al., 2006). In some unsupervised learning tasks the IFGT can be embedded in iterative methods used to compute the eigen vectors (De Freitas et al., 2006). While providing experiments for all applications is beyond the scope of this chapter, as an example we show how IFGT can be used to speed up multivariate kernel density estimation.

The most popular non-parametric method for density estimation is the kernel density estimator (KDE). In its most general form , the d-dimensional KDE is written as (Wand and Jones, 1995)

$$\widehat{p}_N(x) = \frac{1}{N}\sum_{i=1}^{N} K_{\mathbf{H}}\left(x - x_i\right), \text{ where } K_{\mathbf{H}}(x) = |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2}x). \quad (1.30)$$

The d-variate function K is called the *kernel function* and $\mathbf{H}$ is a symmetric positive definite $d \times d$ matrix called the *bandwidth matrix*. In order that $\widehat{p}_N(x)$ is a bona fide density, the kernel function is required to satisfy the following two conditions: $K(u) \geq 0$, and $\int K(u)du = 1$. The most commonly used kernel is the standard d-variate normal density – $K(u) = (2\pi)^{-d/2} e^{-\|u\|^2/2}$.

In general a fully parameterized $d \times d$ positive definite bandwidth matrix

$\mathbf{H}$ can be used to define the density estimate. However in high dimensions the number of independent parameters $(d(d+1)/2)$ are too large to make a good choice. Hence the most commonly used choice is $\mathbf{H} = diag(h_1^2, \ldots, h_d^2)$ or $\mathbf{H} = h^2\mathbf{I}$. For the case when $\mathbf{H} = h^2\mathbf{I}$ the density estimate can be written as,

$$\widehat{p}_N(x) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(2\pi h^2)^{d/2}} e^{-\|x-x_i\|^2/2h^2}. \tag{1.31}$$

The computational cost of evaluating Equation 1.31 at M points is $O(NM)$, making it prohibitively expensive. For example a KDE with 1 million points would take around 2 days. The proposed IFGT algorithm can be used directly to reduce the computational cost to $O(N+M)$.

1.11.1 Bandwidth selection

For a practical implementation of KDE the choice of the bandwidth h is very important. A small h leads to an estimate with small bias and large variance, while a large h leads to a small variance at the expense of an increase in the bias. Various techniques have been proposed for optimal bandwidth selection (Jones et al., 1996). The plug-in bandwidths are known to show more stable performance (Wand and Jones, 1995) than the cross-validation methods. They are based on deriving an expression for the Asymptotic Mean Integrated Squared Error (AMISE) as a function of the bandwidth and then choosing the bandwidth which minimizes it. The simplest among these known as the *rules of thumb* (ROT) assumes that the data is generated by a multivariate normal distribution. For a normal distribution with covariance matrix $\Sigma = diag(\sigma_1^2, \ldots, \sigma_d^2)$ and the bandwidth matrix of the form $\mathbf{H} = diag(h_1^2, \ldots, h_d^2)$ the optimal bandwidths are given by (Wand and Jones, 1995)

$$h_j^{ROT} = \left(\frac{4}{d+2}\right)^{1/(d+4)} N^{-1/(d+4)}\widehat{\sigma}_j, \tag{1.32}$$

where $\widehat{\sigma}_j$ is an estimate of σ_j. This method is known to provide a quick first guess and can be expected to give reasonable bandwidth when the data is close to a normal distribution. It is reported that this tends to slightly oversmooth the data. So in our experiments we only use this as a guess and show the speed up achieved over a range of bandwidths around h^{ROT}. As a practical issue we did not prefer cross-validation because we will have to do the KDE for a range of bandwidths, both small and large. We cannot use the fast methods here since the IFGT cannot be run for very small bandwidths

and the dual-tree does the same work as direct summation for moderate bandwidths.

1.11.2 Experiments

For our experimental comparison we used the SARCOS dataset [1]. The dataset contains 44,484 samples in a 21 dimensional space. The data relates to an inverse dynamics problem for a seven degrees-of-freedom SARCOS anthropomorphic robot arm. In order to ease comparisons all the dimensions were normalized to have the same variance so that we could use only one bandwidth parameter h.

Figure 1.10 compares the time taken by the direct summation, IFGT, and the *kd*-tree method for different dimensions. In each of the plots the KDE was computed for the first d dimensions. The results are shown for $N = 7,000$ points so that the methods could be compared. The KDE was evaluated at $M = N$ points. The results are shown for a range of bandwidths around the optimal bandwidth obtained using the rule of thumb plug-in method. Accuracy of $\epsilon = 10^{-2}$ was used for all the methods. The following observations can be made–

1. The IFGT is faster than the *kd*-tree algorithm at the optimal bandwidth.
2. As the bandwidth increases the IFGT shows better speedups.
3. The *kd*-tree algorithm shows good performance at very small bandwidths.
4. We were able to run the IFGT algorithm till $d = 6$ due to limited memory.
5. As d increases the speedup of the IFGT decreases. The cutoff point, *i.e.*, the value of N for which fast algorithm performs better than the direct summation, increases with dimension.

In the previous plot we used only $7,000$ points in order to compare our algorithm with the *kd*-tree method. For the IFGT the cutoff point–N after which IFGT is faster than the direct method–increases. As result for high dimensional data better speedups are observed for large N. Table 1.1 shows the time taken by the IFGT on the entire dataset.

1. This dataset can be downloaded from the website `http://www.gaussianprocess.org/gpml/data/`.

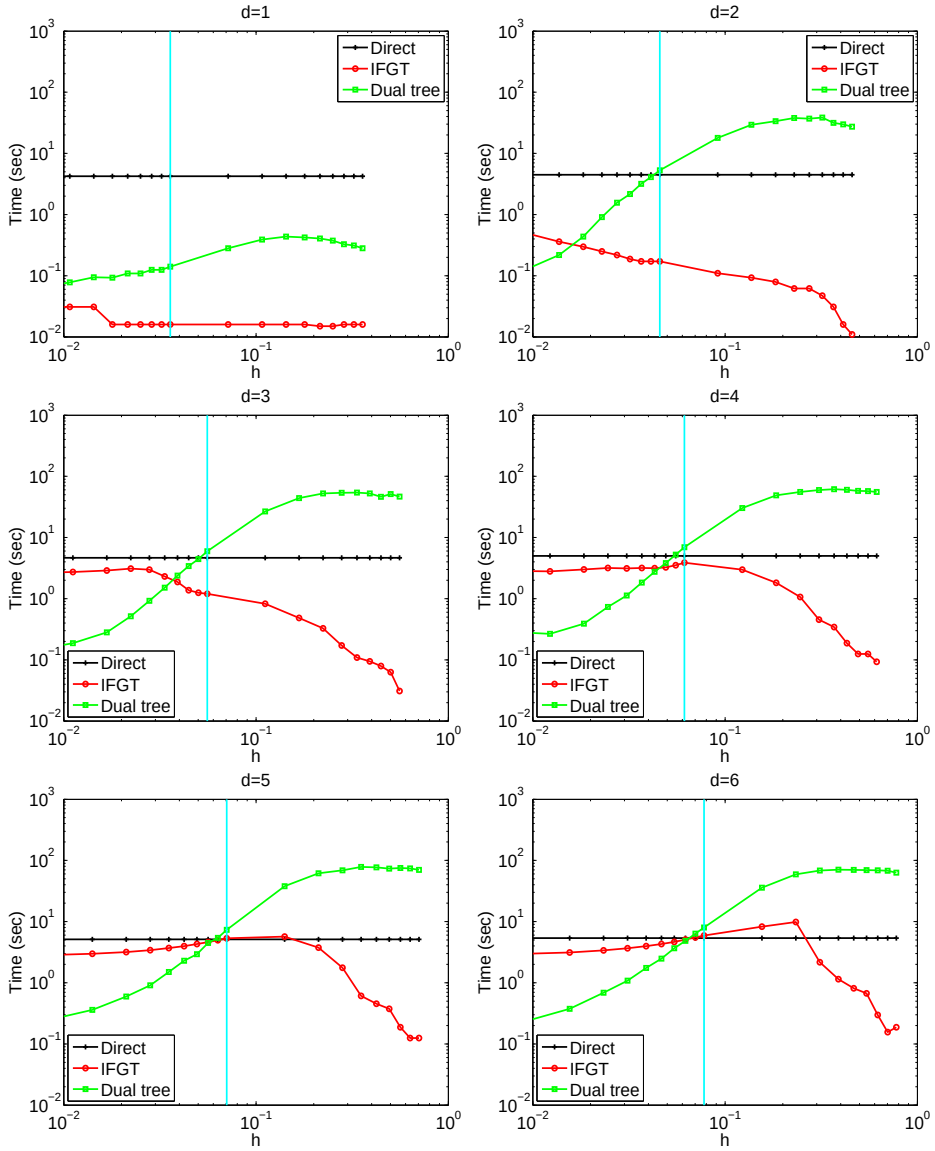

Figure 1.10: *KDE results* The run time in seconds for the direct, IFGT, and the *kd*-tree method for varying dimensionality, d. The results are shown for a range of bandwidths around the optimal bandwidth marked by the straight line in each of the plots. The error was set to $\epsilon = 10^{-2}$. The KDE was evaluated at $M = N = 7,000$ points. The IFGT could not be run across all bandwidths due to limited memory after $d > 6$.

Table 1.1: *KDE results* Time taken by the direct summation and the IFGT on the entire dataset containing $N = 44,484$ source points in d dimensions. The KDE was evaluated at $M = N$ points. The error was set to $\epsilon = 10^{-2}$. The IFGT could not be run due to limited memory after $d > 6$.

d	Optimal bandwidth	Direct time (sec.)	IFGT time (sec.)	speed up
1	0.024730	167.50	0.094	1781.91
2	0.033357	179.38	0.875	205.00
3	0.041688	187.77	6.313	29.74
4	0.049527	195.63	20.563	9.51
5	0.066808	206.49	37.844	5.46
6	0.083527	219.03	65.109	3.36

1.12 Conclusions

We described the improved fast Gauss transform which is capable of computing the sums of Gaussian kernels in linear time in dimensions as high as tens for small bandwidths and as high as hundreds for large bandwidths. While different N-body algorithms have been proposed, each of them performs well under different conditions. Table 1.2 summarizes the conditions under with various algorithms perform better.

1. For very small bandwidths the dual-tree algorithms give the best speed ups.

2. For very large bandwidths the IFGT is substantially faster than the other methods.

3. For moderate bandwidths and moderate dimensions IFGT performs better than dual-tree algorithms.

4. The FGT performs well only for $d \leq 3$.

5. For moderate bandwidths and large dimensions we may still have to resort to direct summation, and fast algorithms remain an area of active research.

Our current research is focussed on combining the IFGT factorization with the *kd*-tree based data structures so that we have an algorithm which gives good speed up both for small and large bandwidths.

One of the goals when designing these kind of algorithms is to give high accuracy guarantees. But sometimes because of the loose error bounds we end up doing much more work than necessary. In such a cause the IFGT can be used by just choosing a truncation number p and seeing how the algorithm performs.

Table 1.2: Summary of the better performing algorithms for different settings of dimensionality d and bandwidth h (assuming data is scaled to a unit hypercube). The bandwidth ranges are approximate.

	Small dimensions $d \leq 3$	Moderate dimensions $3 < d < 10$	Large dimensions $d \geq 10$
Small bandwidth $h \lessapprox 0.1$	Dual tree [kd-tree]	Dual tree [kd-tree]	Dual tree [probably anchors]
Moderate bandwidth $0.1 \lessapprox h \lessapprox 0.5\sqrt{d}$	FGT, IFGT	IFGT	
Large bandwidth $h \gtrapprox 0.5\sqrt{d}$	FGT, IFGT Dual tree	IFGT	IFGT

Appendix

1.A N-body learning software

The implementation of various N-body algorithms can be a bit tricky. However some software is available in the public domain.

1. The C++ code with MATLAB bindings for both the FGT and the IFGT implementation are available at `http://www.umiacs.umd.edu/~vikas/Software/IFGT/IFGT_code.htm`.

2. Fortran code for the FGT is available at `http://math.berkeley.edu/~strain/Codes/index.html`.

3. The C++ code with MATLAB bindings for the dual-tree algorithms can be downloaded from the website `http://www.cs.ubc.ca/~awll/nbody_methods.html`.

4. MATLAB code for fast kernel density estimation based on N-body algorithms is also available at `http://www.ics.uci.edu/~ihler/`.

References

Marshall Bern and David Eppstein. *Approximation algorithms for NP-hard problems*, chapter Approximation algorithms for geometric problems, pages 296–345. PWS Publishing Company, Boston, 1997.

Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines (and other kernel-based learning methods)*. Cambridge University Press, 2000.

Lehel Csato and Manfred Opper. Sparse on-line Gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

Nando De Freitas, Yang Wang, Maryam Mahdaviani, and Dustin Lang. Fast Krylov methods for N-body learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

Jack Dongarra and Francis Sullivan. The top ten algorithms of the century. *Computing in Science and Engineering*, 2(1):22–23, 2000.

Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proc. 20th ACM Symp. Theory of Computing*, pages 434–444, 1988.

Shai Fine and Katya Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

Teofilo Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

Alexander G. Gray and Andrew W. Moore. N-body problems in statistical learning. In *Advances in Neural Information Processing Systems*, pages 521–527, 2001.

Alexander G. Gray and Andrew W. Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM International conference on Data Mining*, 2003.

Leslie Greengard. Fast algorithms for classical physics. *Science*, 265(5174): 909–914, 1994.

Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987.

Leslie Greengard and John Strain. The fast Gauss transform. *SIAM Journal of Scientific and Statistical Computing*, 12(1):79–94, 1991.

Leslie F. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. The MIT Press, 1988.

Alan J. Izenman. Recent developments in nonparametric density estimation. *Journal of American Staistical Association*, 86(413):205–224, 1991.

M. Chris Jones, James S. Marron, and Simon J. Sheather. A brief survey

of bandwidth selection for density estimation. *Journal of American Statistical Association*, 91(433):401–407, March 1996.

Dustin Lang, Mike Klaas, and Nando de Freitas. Empirical testing of fast kernel density estimation algorithms. Technical Report UBC TR-2005-03, Dept. of Computer Science, University of British Columbia, 2005.

Neil D. Lawrence, Matthias Seeger, and Ralf Herbrich. *Advances in Neural Information Processing Systems 15*, chapter Fast Sparse Gaussian Process methods: The Informative Vector Machine, pages 625–632. MIT Press, 2003.

Dongryeol Lee, Alexander G. Gray and Andrew W. Moore. Dual-tree fast Gauss transforms. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

Yuh-Jye Lee and Olvi L. Mangasarian. RSVM: Reduced support vector machines. In *First SIAM International Conference on Data Mining, Chicago*, 2001.

Tomaso Poggio and Steve Smale. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society*, 50(5):537–544, 2003.

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Vikas C. Raykar, Changjaing Yang, Ramani Duraiswami, and Nail A. Gumerov. Fast computation of sums of Gaussians in high dimensions. Technical Report CS-TR-4767, Department of Computer Science, University of Maryland, CollegePark, 2005.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Yirong Shen, Andrew Ng and Matthias Seeger. Fast Gaussian process regression using KD-trees. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

Bernard W. Silverman. Algorithm AS 176: Kernel density estimation using the fast Fourier transform. *Journal of Royal Statistical society Series C: Applied statistics*, 31(1):93–99, 1982.

Alexander J. Smola and Peter L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems*, page 619625. MIT Press, 2001.

Alexander J. Smola, Bernhard Schölkopf, and Klaus-Robert Muller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Re-

port 44, Max-Planck-Institut fr biologische Kybernetik, Tubingen, 1996.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

Michael E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1:211–244, 2001.

Volker Tresp. A Bayesian committee machine. *Neural Computation*, 12(11): 2719–2741, 2000.

Grace Wabha. *Spline Models for Observational data*. SIAM, 1990.

M. P. Wand and M. Chris Jones. *Kernel Smoothing*. Chapman and Hall, London, 1995.

Christopher K. I. Williams and Carl E. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems*, volume 8, 1996.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, page 682688. MIT Press, 2001.

Changjaing Yang, Ramani Duraiswami, and Larry Davis. Efficient kernel machines using the improved fast Gauss transform. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1561–1568. MIT Press, Cambridge, MA, 2005.

Changjaing Yang, Ramani Duraiswami, Nail A. Gumerov, and Larry Davis. Improved fast Gauss transform and efficient kernel density estimation. In *IEEE International Conference on Computer Vision*, pages 464–471, 2003.

Index

N-body learning, 4

dual-tree methods, 7

farthest point clustering, 13
fast Gauss transform, 8, 16
fast kernel density estimation, 24
fast multipole methods, 8

Gauss transform, 6

Horner's rule, 11

improved fast Gauss transform, 1, 12

matrix-vector multiplication, 4
multi-index notation, 9

polynomial kernel, 5