Computational tractability of machine learning algorithms for tall fat data

Getting good enough solutions as fast as possible

Vikas Chandrakant Raykar
vikas@cs.umd.edu

University of Maryland, CollegePark

May 4, 2006

Outline of the proposal

Tall Data

Large number of training examples with small number of attributes

	Attribute 1	Attribute 2	Attribute 3
1	☐	☐	☐
3	☐	☐	☐
4	☐	☐	☐
.	☐	☐	☐
.	☐	☐	☐
.	☐	☐	☐
.	☐	☐	☐
.	☐	☐	☐
.	☐	☐	☐
.	☐	☐	☐
.	☐	☐	☐
10^6	☐	☐	☐

Fat Data

Small number of training examples with large number of attributes

	Attribute 1	.	.	.	.	.	.	.	.	Attribute 1000
1	□	□	□	□	□	□	□	□	□	□
2	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
100	□	□	□	□	□	□	□	□	□	□

Tall Fat Data

Large number of training examples with large number of attributes

	Attribute 1	.	.	.	.	.	.	.	.	Attribute 1000
1	□	□	□	□	□	□	□	□	□	□
2	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
10^6	□	□	□	□	□	□	□	□	□	□

Tall Fat Data

Large number of training examples with large number of attributes

	Attribute 1	.	.	.	.	.	.	.	.	Attribute 1000
1	□	□	□	□	□	□	□	□	□	□
2	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
.	□	□	□	□	□	□	□	□	□	□
10^6	□	□	□	□	□	□	□	□	□	□

Currently we can handle only tall and slightly fat data.

Learning with massive data sets

Huge data sets containing

- millions of training examples (tall data)
- with large number of attributes (fat data)

are relatively easy to gather.

Learning with massive data sets

Huge data sets containing

- millions of training examples (tall data)
- with large number of attributes (fat data)

are relatively easy to gather.

Example

Genome sequencing, internet databases, experimental data from particle physics, medical databases, financial records, weather reports, audio and video data.

Learning with massive data sets

Huge data sets containing

- millions of training examples (tall data)
- with large number of attributes (fat data)

are relatively easy to gather.

Example

Genome sequencing, internet databases, experimental data from particle physics, medical databases, financial records, weather reports, audio and video data.

Learning is a principled method for inferring predictive models from the data

Poggio, T. and Smale, S. 2003. The mathematics of learning: Dealing with data. Notices of the American Mathematical Society 50, 5, 537–544..

Two approaches to learning

Parametric approach

- Assumes a known parametric form for the model to be learnt.
- Training $\Leftrightarrow$ Estimate the unknown parameters.

Two approaches to learning

Parametric approach

- Assumes a known parametric form for the model to be learnt.
- Training $\Leftrightarrow$ Estimate the unknown parameters.

Once the model has been trained, for future prediction the training examples can be discarded.

- The essence of the training examples have been captured in the model parameters.

Two approaches to learning

Parametric approach

- Assumes a known parametric form for the model to be learnt.
- Training $\Leftrightarrow$ Estimate the unknown parameters.

Once the model has been trained, for future prediction the training examples can be discarded.

- The essence of the training examples have been captured in the model parameters.

Leads to erroneous inference unless the model is known *a priori*.

Two approaches to learning

Non-parametric approach

Two approaches to learning

Non-parametric approach

- Do not make any assumptions on the form of the underlying function.
- Letting the data speak for themselves.
- Perform better than parametric methods.

Two approaches to learning

Non-parametric approach

- Do not make any assumptions on the form of the underlying function.
- Letting the data speak for themselves.
- Perform better than parametric methods.

However all the available *data has to be retained* while making the inference.

The computational consequence of this can be quite significant.

Computational curse of non-parametric methods

Let N be the number of training examples

Computational curse of non-parametric methods

Let N be the number of training examples

Most state-of-the-art non-parametric methods in machine leaning and computational statistics scale as either $\mathcal{O}(N^3)$ or $\mathcal{O}(N^2)$.

Computational curse of non-parametric methods

Let N be the number of training examples

Most state-of-the-art non-parametric methods in machine leaning and computational statistics scale as either $\mathcal{O}(N^3)$ or $\mathcal{O}(N^2)$.

- This has seriously restricted the use of massive data sets.
- Current implementations can handle only a few thousands of training examples.
- Both the data set size and processor speed are growing according to Moore's law.

Computational curse of non-parametric methods

Let N be the number of training examples

Most state-of-the-art non-parametric methods in machine leaning and computational statistics scale as either $\mathcal{O}(N^3)$ or $\mathcal{O}(N^2)$.

- This has seriously restricted the use of massive data sets.
- Current implementations can handle only a few thousands of training examples.
- Both the data set size and processor speed are growing according to Moore's law.

Example

A simple kernel density estimation with 1 million points would take around 2 days.

Goals of the proposed thesis

1. Identify the key computational primitives contributing to the $\mathcal{O}(N^3)$ or $\mathcal{O}(N^2)$ complexity.
2. Speedup up these primitives by approximate algorithms that scale as $\mathcal{O}(N)$ and provide high accuracy guarantees.

Goals of the proposed thesis

1. Identify the key computational primitives contributing to the $\mathcal{O}(N^3)$ or $\mathcal{O}(N^2)$ complexity.
2. Speedup up these primitives by approximate algorithms that scale as $\mathcal{O}(N)$ and provide high accuracy guarantees.
3. Enable the use of massive datasets for different learning algorithms.

Tools

We use ideas and techniques from

- Computational physics $\mapsto$ `fast multipole methods.`

Tools

We use ideas and techniques from

- Computational physics $\mapsto$ `fast multipole methods.`
- Scientific computing $\mapsto$ `iterative methods, pre-conditioners.`

Tools

We use ideas and techniques from

- Computational physics $\mapsto$ `fast multipole methods.`
- Scientific computing $\mapsto$ `iterative methods, pre-conditioners.`
- Computational geometry $\mapsto$ `clustering,` *kd*`-trees.`

Tools

We use ideas and techniques from

- Computational physics $\mapsto$ `fast multipole methods.`
- Scientific computing $\mapsto$ `iterative methods, pre-conditioners.`
- Computational geometry $\mapsto$ `clustering,` *kd*`-trees.`

and apply it to

- Machine learning $\mapsto$ `kernel machines, Gaussian processes.`

Tools

We use ideas and techniques from

- Computational physics $\mapsto$ `fast multipole methods.`
- Scientific computing $\mapsto$ `iterative methods, pre-conditioners.`
- Computational geometry $\mapsto$ `clustering,` *kd*`-trees.`

and apply it to

- Machine learning $\mapsto$ `kernel machines, Gaussian processes.`
- Computational statistics $\mapsto$ `kernel density estimation.`

Top ten algorithms of the century [1]

1. Monte Carlo method.
2. Simplex method of linear programming.
3. Krylov Subspace Iteration method.
4. Householder matrix decomposition.
5. Fortran compiler.
6. QR algorithm for eigenvalue calculation.
7. Quicksort algorithm.
8. Fast Fourier Transform.
9. Integer Relation Detection Algorithm.
10. Fast Multipole methods.

[1] Dongarra, J. and Sullivan, F. 2000. The top ten algorithms of the century. Computing in Science and Engineering.

Outline of the proposal

Key Computational tasks

	Training (N examples)	Prediction (at N points)	Choosing parameters
KDE		$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Kernel regression	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Gaussian processes	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^3)$
SVM	$\mathcal{O}(N_{sv}^3)$	$\mathcal{O}(N_{sv}N)$	
Laplacian eigenmaps	$\mathcal{O}(N^3)$		
Kernel PCA	$\mathcal{O}(N^3)$		

Key Computational tasks

	Training (N examples)	Prediction (at N points)	Choosing parameters
KDE		$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Kernel regression	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Gaussian processes	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^3)$
SVM	$\mathcal{O}(N_{sv}^3)$	$\mathcal{O}(N_{sv}N)$	
Laplacian eigenmaps	$\mathcal{O}(N^3)$		
Kernel PCA	$\mathcal{O}(N^3)$		

Identify the key computational primitives contributing to the $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$ complexity.

Canonical learning tasks

Training data

$\{x_i \in \mathbf{R}^d, y_i \in \mathbf{M}\}_{i=1}^N$

Canonical learning tasks

Training data

$\{x_i \in \mathbf{R}^d, y_i \in \mathbf{M}\}_{i=1}^N$

Learning can be viewed as function estimation $f : \mathbf{R}^d \rightarrow \mathbf{M}$

- Regression $\mathbf{M} = \mathbf{R}$
- Binary Classification $\mathbf{M} = \{-1, +1\}$.
- Density estimation

Canonical learning tasks

Training data

$\{x_i \in \mathbf{R}^d, y_i \in \mathbf{M}\}_{i=1}^N$

Learning can be viewed as function estimation $f : \mathbf{R}^d \rightarrow \mathbf{M}$

- Regression $\mathbf{M} = \mathbf{R}$
- Binary Classification $\mathbf{M} = \{-1, +1\}$.
- Density estimation

Three tasks

- Training $\rightarrow$ Learning the function f from examples.
- Prediction $\rightarrow$ Given a new x predict y..
- Model Selection $\rightarrow$ Choosing the hyperparameters.

Kernel machines

Minimize the regularized empirical risk functional $R_{reg}[f]$.

$$\min_{f \in \mathcal{H}} R_{reg}[f] = \frac{1}{N} \sum_{i=1}^{N} L[f(x_i), y_i] + \lambda \|f\|_{\mathcal{H}}^2, \tag{1}$$

where $\mathcal{H}$ denotes a reproducing kernel Hilbert space (RKHS) [2].

[2] Wabha, G. 1990. Spline Models for Observational data. SIAM.

Kernel machines

Minimize the regularized empirical risk functional $R_{reg}[f]$.

$$\min_{f \in \mathcal{H}} R_{reg}[f] = \frac{1}{N} \sum_{i=1}^{N} L[f(x_i), y_i] + \lambda \|f\|_{\mathcal{H}}^2, \tag{1}$$

where $\mathcal{H}$ denotes a reproducing kernel Hilbert space (RKHS) [2].

Theorem (Representer Theorem)

If $k : X \times X \mapsto Y$ is the kernel of the RKHS $\mathcal{H}$ then the minimizer of Equation 1 is of the form

$$f(x) = \sum_{i=1}^{N} q_i k(x, x_i). \tag{2}$$

[2] Wabha, G. 1990. Spline Models for Observational data. SIAM.

Examples

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

[3] Poggio, T. and Smale, S. 2003. The mathematics of learning: Dealing with data. Notices of the American Mathematical Society 50, 5, 537-544.

[4] Cristianini, N. and Shawe-Taylor, J. 2000. An Introduction to Support Vector Machines (and other kernel-based learning methods). Cambridge University Press.

[5] Wand, M. P. and Jones, M. C. 1995. Kernel Smoothing. Chapman and Hall, London.

[6] Rasmussen, C. E. and Williams, C. K. I. 2006. Gaussian Processes for Machine Learning. The MIT Press.

Examples

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

- **Kernel machines** (e.g. RLS [3], SVM [4]) f is the regression/classification function. [Representer theorem]

[3] Poggio, T. and Smale, S. 2003. The mathematics of learning: Dealing with data. Notices of the American Mathematical Society 50, 5, 537-544.

[4] Cristianini, N. and Shawe-Taylor, J. 2000. An Introduction to Support Vector Machines (and other kernel-based learning methods). Cambridge University Press.

[5] Wand, M. P. and Jones, M. C. 1995. Kernel Smoothing. Chapman and Hall, London.

[6] Rasmussen, C. E. and Williams, C. K. I. 2006. Gaussian Processes for Machine Learning. The MIT Press.

Examples

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

- **Kernel machines** (e.g. RLS [3], SVM [4]) f is the regression/classification function. [Representer theorem]
- **Density estimation** f is the kernel density estimate [5].

[3] Poggio, T. and Smale, S. 2003. The mathematics of learning: Dealing with data. Notices of the American Mathematical Society 50, 5, 537-544.

[4] Cristianini, N. and Shawe-Taylor, J. 2000. An Introduction to Support Vector Machines (and other kernel-based learning methods). Cambridge University Press.

[5] Wand, M. P. and Jones, M. C. 1995. Kernel Smoothing. Chapman and Hall, London.

[6] Rasmussen, C. E. and Williams, C. K. I. 2006. Gaussian Processes for Machine Learning. The MIT Press.

Examples

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

- **Kernel machines** (e.g. RLS [3], SVM [4]) f is the regression/classification function. [Representer theorem]
- **Density estimation** f is the kernel density estimate [5].
- **Gaussian processes** [6] f is the mean prediction.

[3] Poggio, T. and Smale, S. 2003. The mathematics of learning: Dealing with data. Notices of the American Mathematical Society 50, 5, 537-544.

[4] Cristianini, N. and Shawe-Taylor, J. 2000. An Introduction to Support Vector Machines (and other kernel-based learning methods). Cambridge University Press.

[5] Wand, M. P. and Jones, M. C. 1995. Kernel Smoothing. Chapman and Hall, London.

[6] Rasmussen, C. E. and Williams, C. K. I. 2006. Gaussian Processes for Machine Learning. The MIT Press.

Prediction

Prediction

Given N training examples $\{x_i\}_{i=1}^N$, the key computational task is to compute a weighted linear combination of local kernel functions centered on the training data, i.e.,

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

Prediction

Given N training examples $\{x_i\}_{i=1}^N$, the key computational task is to compute a weighted linear combination of local kernel functions centered on the training data, i.e.,

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

The computation complexity to predict at M points given N training examples scales as $\mathcal{O}(MN)$.

Training

- Training these models scales as $\mathcal{O}(N^3)$ since most involve solving the linear system of equation

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}.$$

Training

- Training these models scales as $\mathcal{O}(N^3)$ since most involve solving the linear system of equation

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}.$$

  - $\mathbf{K}$ is the dense $N \times N$ Gram matrix where $[\mathbf{K}]_{ij} = k(x_i, x_j)$.
  - $\mathbf{I}$ is the identity matrix.
  - λ is some regularization parameter or noise variance.

Training

- Training these models scales as $\mathcal{O}(N^3)$ since most involve solving the linear system of equation

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}.$$

  - $\mathbf{K}$ is the dense $N \times N$ Gram matrix where $[\mathbf{K}]_{ij} = k(x_i, x_j)$.
  - $\mathbf{I}$ is the identity matrix.
  - λ is some regularization parameter or noise variance.

Direct inversion requires $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ storage.

Unsupervised learning

- Methods like
  - kernel principal component analysis [7]
  - spectral clustering [8]
  - nonlinear dimensionality reduction (Laplacian eigenmaps [9])

  involve computing the eigen vectors of the Gram/Laplacian matrix.
- Computing eigenvectors of a dense matrix is $\mathcal{O}(N^3)$

[7] Smola, A., Scholkopf, B., and Muller, K.-R. 1996. Nonlinear component analysis as a kernel eigenvalue problem. Tech. Rep. 44, Max-Planck-Institut fr biologische Kybernetik, Tubingen.

[8] Chung, F. 1997. Spectral Graph Theory. Amer. Math. Society Press.

[9] M. Belkin and P. Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. Proceedings of Advances in Neural Information Processing Systems. Vol. 14, 2002.

Model selection

- Most non-parametric methods require choosing some parameter (e.g. bandwidth h of the kernel).

Model selection

- Most non-parametric methods require choosing some parameter (e.g. bandwidth h of the kernel).
- Two approaches.
  - Cross-validation.
  - Maximizing thee marginal likelihood.

Model selection

- Most non-parametric methods require choosing some parameter (e.g. bandwidth h of the kernel).
- Two approaches.
  - Cross-validation.
  - Maximizing thee marginal likelihood.
- Automatic procedures to choose these parameters are iterative with each iteration costing $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$.

N-body problems in statistical learning

$\mathcal{O}(N^2)$ because computations involve considering pair-wise elements.

[10] A. Gray and A. Moore. N-body problems in statistical learning. In Advances in Neural Information Processing Systems, pages 521-527, 2001.

[11] Greengard, L. 1994. Fast algorithms for classical physics. Science 265, 5174, 909-914

N-body problems in statistical learning

$\mathcal{O}(N^2)$ because computations involve considering pair-wise elements.
N-body problems in statistical learning[10]
in analogy with the
Coulombic N-body problems [11] occurring in computational physics.

- These are potential based problems involving forces or charges.
- In our case the potential corresponds to the kernel function.

[10] A. Gray and A. Moore. N-body problems in statistical learning. In Advances in Neural Information Processing Systems, pages 521-527, 2001.
[11] Greengard, L. 1994. Fast algorithms for classical physics. Science 265, 5174, 909-914

Primitive 1–Iterative methods

Reduce training time from $\mathcal{O}(N^3)$ to $\mathcal{O}(kN^2)$

We need to solve the linear system of equation

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}.$$

$\mathbf{K}$ is the $N \times N$ Gram matrix where $[\mathbf{K}]_{ij} = k(x_i, x_j)$.

Primitive 1–Iterative methods

Reduce training time from $\mathcal{O}(N^3)$ to $\mathcal{O}(kN^2)$

We need to solve the linear system of equation

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}.$$

$\mathbf{K}$ is the $N \times N$ Gram matrix where $[\mathbf{K}]_{ij} = k(x_i, x_j)$.

- Direct inversion is $\mathcal{O}(N^3)$.

Primitive 1–Iterative methods

Reduce training time from $\mathcal{O}(N^3)$ to $\mathcal{O}(kN^2)$

We need to solve the linear system of equation

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}.$$

$\mathbf{K}$ is the $N \times N$ Gram matrix where $[\mathbf{K}]_{ij} = k(x_i, x_j)$.

- Direct inversion is $\mathcal{O}(N^3)$.
- We will use iterative methods like conjugate-gradient to bring it down to $\mathcal{O}(kN^2)$–k is the number of iterations.

Primitive 1–Iterative methods

Reduce training time from $\mathcal{O}(N^3)$ to $\mathcal{O}(kN^2)$

We need to solve the linear system of equation

$$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}.$$

$\mathbf{K}$ is the $N \times N$ Gram matrix where $[\mathbf{K}]_{ij} = k(x_i, x_j)$.

- Direct inversion is $\mathcal{O}(N^3)$.
- We will use iterative methods like conjugate-gradient to bring it down to $\mathcal{O}(kN^2)$–k is the number of iterations.
- The quadratic complexity is due to the matrix-vector product $\mathbf{K}q$ for some q.

Primitive 2-Fast Matrix Vector Multiplication

We need a fast algorithm to compute

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

Primitive 2-Fast Matrix Vector Multiplication

We need a fast algorithm to compute

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

Matrix Vector Multiplication $f = \mathbf{K}q$

$$\begin{pmatrix} f(y_1) \\ f(y_2) \\ \vdots \\ f(y_M) \end{pmatrix} = \begin{pmatrix} k(y_1, x_1) & k(y_1, x_2) & \ldots & k(y_1, x_N) \\ k(y_2, x_1) & k(y_2, x_2) & \ldots & k(y_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(y_M, x_1) & k(y_M, x_2) & \ldots & k(y_M, x_N) \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{pmatrix}$$

Primitive 2-Fast Matrix Vector Multiplication

We need a fast algorithm to compute

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

Matrix Vector Multiplication $f = \mathbf{K}q$

$$\begin{pmatrix} f(y_1) \\ f(y_2) \\ \vdots \\ f(y_M) \end{pmatrix} = \begin{pmatrix} k(y_1, x_1) & k(y_1, x_2) & \ldots & k(y_1, x_N) \\ k(y_2, x_1) & k(y_2, x_2) & \ldots & k(y_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(y_M, x_1) & k(y_M, x_2) & \ldots & k(y_M, x_N) \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{pmatrix}$$

- Direct computation is $\mathcal{O}(MN)$.

Primitive 2-Fast Matrix Vector Multiplication

We need a fast algorithm to compute

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

Matrix Vector Multiplication $f = \mathbf{K}q$

$$\begin{pmatrix} f(y_1) \\ f(y_2) \\ \vdots \\ f(y_M) \end{pmatrix} = \begin{pmatrix} k(y_1, x_1) & k(y_1, x_2) & \ldots & k(y_1, x_N) \\ k(y_2, x_1) & k(y_2, x_2) & \ldots & k(y_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(y_M, x_1) & k(y_M, x_2) & \ldots & k(y_M, x_N) \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{pmatrix}$$

- Direct computation is $\mathcal{O}(MN)$.
- Reduce from $\mathcal{O}(MN)$ to $\mathcal{O}(M + N)$

Why should $\mathcal{O}(M + N)$ be possible?

Why should $\mathcal{O}(M + N)$ be possible?

Exploit the structure in the matrix.

Why should $\mathcal{O}(M+N)$ be possible?

Exploit the structure in the matrix.

Structured matrix

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

Why should $\mathcal{O}(M+N)$ be possible?

Exploit the structure in the matrix.

Structured matrix

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

$\mathbf{K}$ is a structured matrix.

$[\mathbf{K}]_{ij} = k(x_i, y_j)$

Why should $\mathcal{O}(M+N)$ be possible?

Exploit the structure in the matrix.

Structured matrix

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

$\mathbf{K}$ is a structured matrix.

$[\mathbf{K}]_{ij} = k(x_i, y_j) = e^{-\|x_i - y_j\|^2/h^2}$ (Gaussian kernel)

Motivating toy example

Consider

$$G(y_j) = \sum_{i=1}^{N} q_i (x_i - y_j)^2 \text{ for } j = 1, \ldots, M.$$

Direct summation is $\mathcal{O}(MN)$.

Factorize and regroup

$$G(y_j) = \sum_{i=1}^{N} q_i (x_i - y_j)^2$$

Factorize and regroup

$$
\begin{aligned}
G(y_j) &= \sum_{i=1}^{N} q_i (x_i - y_j)^2 \\
&= \sum_{i=1}^{N} q_i (x_i^2 - 2x_i y_j + y_j^2)
\end{aligned}
$$

Factorize and regroup

$$
\begin{aligned}
G(y_j) &= \sum_{i=1}^{N} q_i (x_i - y_j)^2 \\
&= \sum_{i=1}^{N} q_i (x_i^2 - 2x_i y_j + y_j^2) \\
&= \left[\sum_{i=1}^{N} q_i x_i^2\right] - 2y_j \left[\sum_{i=1}^{N} q_i x_i\right] + y_j^2 \left[\sum_{i=1}^{N} q_i\right]
\end{aligned}
$$

Factorize and regroup

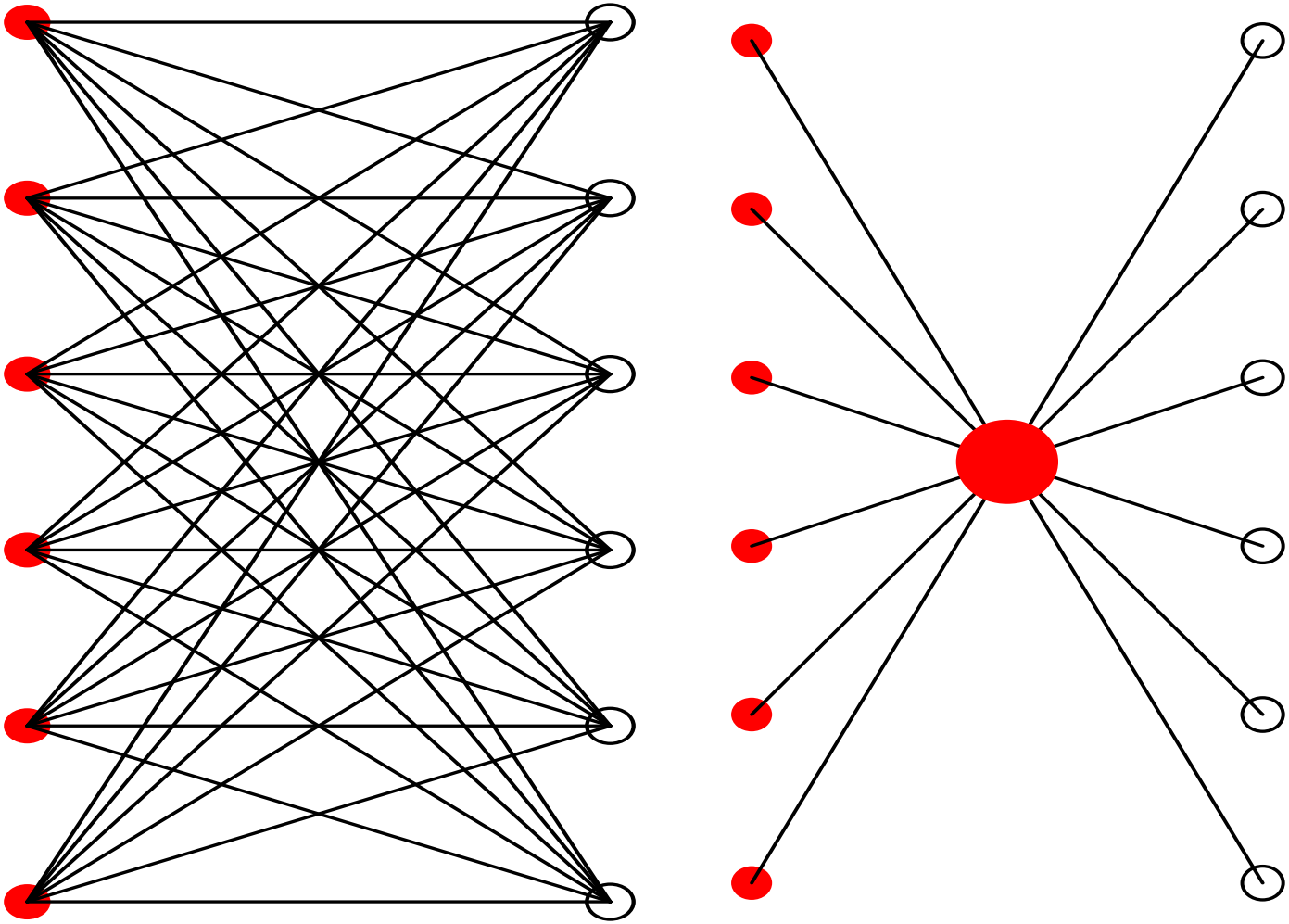
$$
\begin{aligned}
G(y_j) &= \sum_{i=1}^{N} q_i (x_i - y_j)^2 \\
&= \sum_{i=1}^{N} q_i (x_i^2 - 2x_i y_j + y_j^2) \\
&= \left[\sum_{i=1}^{N} q_i x_i^2\right] - 2y_j \left[\sum_{i=1}^{N} q_i x_i\right] + y_j^2 \left[\sum_{i=1}^{N} q_i\right] \\
&= M_2 - 2y_j M_1 + y_j^2 M_0
\end{aligned}
$$

Factorize and regroup

$$
\begin{aligned}
G(y_j) &= \sum_{i=1}^{N} q_i (x_i - y_j)^2 \\
&= \sum_{i=1}^{N} q_i (x_i^2 - 2x_i y_j + y_j^2) \\
&= \left[\sum_{i=1}^{N} q_i x_i^2\right] - 2y_j \left[\sum_{i=1}^{N} q_i x_i\right] + y_j^2 \left[\sum_{i=1}^{N} q_i\right] \\
&= M_2 - 2y_j M_1 + y_j^2 M_0
\end{aligned}
$$

The moments M_2, M_1, and M_0 can be pre-computed in $\mathcal{O}(N)$.
Hence the computational complexity is $\mathcal{O}(M + N)$.
Encapsulating information in terms of the moments.

Direct vs Fast



In general

For any kernel $K(x, y)$ we can expand as

$$K(x, y) = \sum_{k=1}^{p} \Phi_k(x)\Psi_k(y) + error.$$

In general

For any kernel $K(x, y)$ we can expand as

$$K(x, y) = \sum_{k=1}^{p} \Phi_k(x)\Psi_k(y) + error.$$

The fast summation is of the form

$$G(y_j) = \sum_{k=1}^{p} A_k \Psi_k(y) + error,$$

where the moments A_k can be pre-computed as

$$A_k = \sum_{i=1}^{N} q_i \Phi_k(x_i).$$

In general

For any kernel $K(x, y)$ we can expand as

$$K(x, y) = \sum_{k=1}^{p} \Phi_k(x)\Psi_k(y) + error.$$

The fast summation is of the form

$$G(y_j) = \sum_{k=1}^{p} A_k \Psi_k(y) + error,$$

where the moments A_k can be pre-computed as

$$A_k = \sum_{i=1}^{N} q_i \Phi_k(x_i).$$

- Organize using data-structures to use this effectively.
- Give accuracy guarantees.

Two aspects of the problem

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

Gaussian kernel

The most commonly used kernel function in machine learning is the *Gaussian kernel*

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

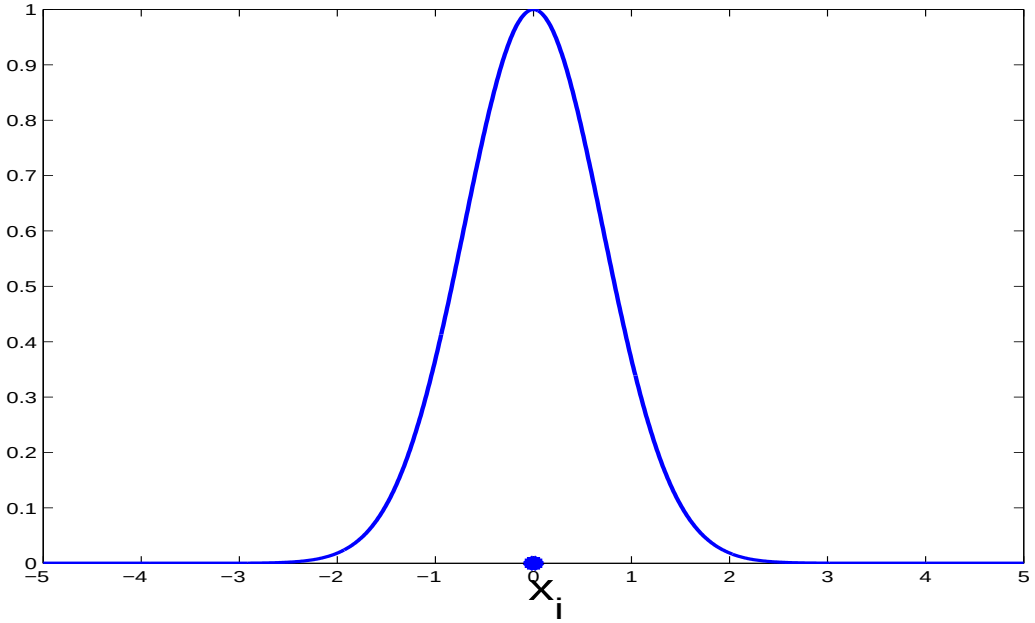
where h is called the *bandwidth* of the kernel.

Gaussian kernel

The most commonly used kernel function in machine learning is the *Gaussian kernel*

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

where h is called the *bandwidth* of the kernel.



Notion of ϵ-exact approximation

Notion of ϵ-exact approximation

- Direct computation is $\mathcal{O}(MN)$.
- We will compute $f(y_j)$ approximately so as to reduce the computational complexity to $\mathcal{O}(N + M)$.
- Speedup at the expense of reduced precision.

Notion of ϵ-exact approximation

- Direct computation is $\mathcal{O}(MN)$.
- We will compute $f(y_j)$ approximately so as to reduce the computational complexity to $\mathcal{O}(N + M)$.
- Speedup at the expense of reduced precision.

- User provides a accuracy parameter ϵ.

Notion of ϵ-exact approximation

- Direct computation is $\mathcal{O}(MN)$.
- We will compute $f(y_j)$ approximately so as to reduce the computational complexity to $\mathcal{O}(N+M)$.
- Speedup at the expense of reduced precision.

- User provides a accuracy parameter ϵ.
- The algorithm computes $\hat{f}(y_j)$ such that $|\hat{f}(y_j) - f(y_j)| < \epsilon$.

Notion of ϵ-exact approximation

- Direct computation is $\mathcal{O}(MN)$.
- We will compute $f(y_j)$ approximately so as to reduce the computational complexity to $\mathcal{O}(N+M)$.
- Speedup at the expense of reduced precision.

- User provides a accuracy parameter ϵ.
- The algorithm computes $\hat{f}(y_j)$ such that $|\hat{f}(y_j) - f(y_j)| < \epsilon$.

- The constant in $\mathcal{O}(N+M)$ depends on the accuracy ϵ.

Notion of ϵ-exact approximation

- Direct computation is $\mathcal{O}(MN)$.
- We will compute $f(y_j)$ approximately so as to reduce the computational complexity to $\mathcal{O}(N+M)$.
- Speedup at the expense of reduced precision.

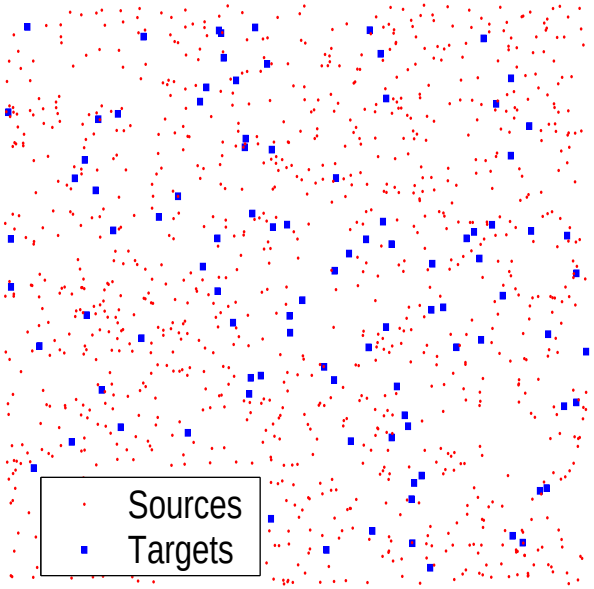
- User provides a accuracy parameter ϵ.
- The algorithm computes $\hat{f}(y_j)$ such that $|\hat{f}(y_j) - f(y_j)| < \epsilon$.

- The constant in $\mathcal{O}(N+M)$ depends on the accuracy ϵ.
- Smaller the accuracy $\rightarrow$ Larger the speedup.
- ϵ can be arbitrarily small.
- For machine level precision no difference between the direct and the fast methods.

Discrete Gauss Transform

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

- $\{q_i \in \mathbf{R}\}_{i=1,\ldots,N}$ are the N source weights.
- $\{x_i \in \mathbf{R}^d\}_{i=1,\ldots,N}$ are the N source points.
- $\{y_j \in \mathbf{R}^d\}_{j=1,\ldots,M}$ are the M target points.
- $h \in \mathbf{R}^+$ is the source scale or bandwidth.



Outline of the proposal

Fast Fourier Transform (FFT)

- If the sources and targets are on a uniform grid we can use FFT.

Fast Fourier Transform (FFT)

- If the sources and targets are on a uniform grid we can use FFT.
- Computational complexity is reduced to $\mathcal{O}(N \log N)$.

Fast Fourier Transform (FFT)

- If the sources and targets are on a uniform grid we can use FFT.
- Computational complexity is reduced to $\mathcal{O}(N \log N)$.
- For irregularly spaced points can use gridded approximations.

Fast Fourier Transform (FFT)

- If the sources and targets are on a uniform grid we can use FFT.
- Computational complexity is reduced to $\mathcal{O}(N \log N)$.
- For irregularly spaced points can use gridded approximations.
- However no accuracy guarantees.

B. 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.

Sparse data-set methods and low rank representation

- Select a subset of the data.

Sparse data-set methods and low rank representation

- Select a subset of the data.
- Use low rank approximations to the Gram matrix.

Sparse data-set methods and low rank representation

- Select a subset of the data.
- Use low rank approximations to the Gram matrix.
- Different strategies for selection.

Sparse data-set methods and low rank representation

- Select a subset of the data.
- Use low rank approximations to the Gram matrix.
- Different strategies for selection.
- However no accuracy guarantees.

Williams, C. K. I. and Seeger, M. 2001. Using the Nyström method to speed up kernel machines. In Advances in Neural Information Processing Systems. MIT Press, 682-688.
Smola, A. and Bartlett, B. 2001. Sparse greedy gaussian process regression. In Advances in Neural Information Processing Systems. MIT Press, 619-625.
Fine, S. and Scheinberg, K. 2001. Efficient SVM training using low-rank kernel representations. Journal of Machine Learning Research 2, 243-264.
Lee, Y.-J. and Mangasarian, O. 2001. Rsvm: Reduced support vector machines. In First SIAM International Conference on Data Mining, Chicago.

Dual-tree methods

- Organize both the source and target data using a kd-tree.
- Expand the cross product of the trees.
- Spend time only where it is essential.
- Gives accuracy guarantees.

Dual-tree methods

- Organize both the source and target data using a kd-tree.
- Expand the cross product of the trees.
- Spend time only where it is essential.
- Gives accuracy guarantees.
  - However as ϵ goes to zero the algorithm does not scale well.
  - No series expansions.

Dual-tree methods

- Organize both the source and target data using a kd-tree.
- Expand the cross product of the trees.
- Spend time only where it is essential.
- Gives accuracy guarantees.
  - However as ϵ goes to zero the algorithm does not scale well.
  - No series expansions.
- Postulated to be $\mathcal{O}(N)$ (no proof).
- Single tree version is $\mathcal{O}(N \log N)$

A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In SIAM International conference on Data Mining, 2003.
Y. Shen, A. Ng, and M. Seeger. Fast Gaussian process regression using KD-trees. In Y. Weiss, B. Scholkopf, and J. Platt, editors, Advances in Neural Information Processing Systems 18. MIT Press, Cambridge, MA, 2006

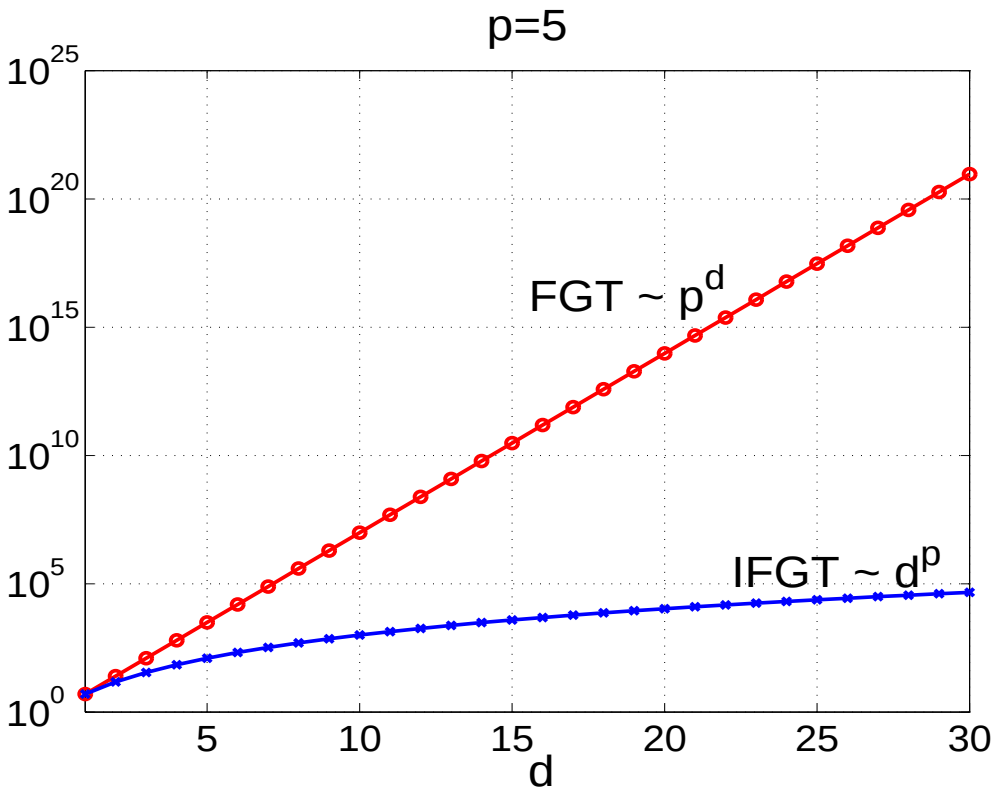
Fast Gauss Transform (FGT)

- $\epsilon -$ *exact* approximation algorithm.
- Computational complexity is $\mathcal{O}(M + N)$.
- Proposed by Greengard and Strain and applied successfully to a few lower dimensional applications in mathematics and physics.
- However the algorithm has not been widely used much in statistics, pattern recognition, and machine learning applications where higher dimensions occur commonly.

L. Greengard and J. Strain. The fast Gauss transform. SIAM Journal of Scientific and Statistical Computing, 12(1):79-94, 1991

Constants are important

- FGT $\sim \mathcal{O}(p^d(M+N))$.
- We propose a method Improved FGT (IFGT) which scales as $\sim \mathcal{O}(d^p(M+N))$.



Outline of the proposal

Improved Fast Gauss Transform (IFGT)

1. The number of the terms grows exponentially with dimensionality d.

Improved Fast Gauss Transform (IFGT)

1. The number of the terms grows exponentially with dimensionality d.
   - Different series expansion–reduces the number of the expansion terms to the polynomial order.

Improved Fast Gauss Transform (IFGT)

1. The number of the terms grows exponentially with dimensionality d.
   - Different series expansion–reduces the number of the expansion terms to the polynomial order.
2. The space subdivision scheme is a uniform box subdivision scheme which is inefficient in higher dimensions.

Improved Fast Gauss Transform (IFGT)

1. The number of the terms grows exponentially with dimensionality d.
   - Different series expansion–reduces the number of the expansion terms to the polynomial order.
2. The space subdivision scheme is a uniform box subdivision scheme which is inefficient in higher dimensions.
   - k-center algorithm is applied to subdivide the space which is more efficient.

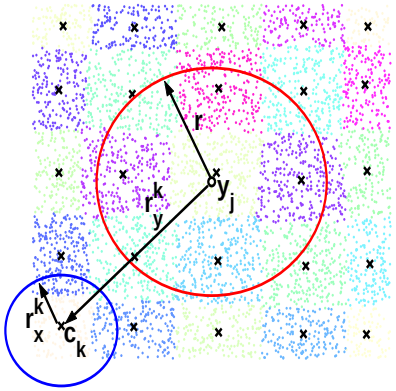
Improved Fast Gauss Transform (IFGT)

1. The number of the terms grows exponentially with dimensionality d.
   - Different series expansion–reduces the number of the expansion terms to the polynomial order.
2. The space subdivision scheme is a uniform box subdivision scheme which is inefficient in higher dimensions.
   - k-center algorithm is applied to subdivide the space which is more efficient.
3. The constant term due to the translation of the far-field Hermite series to the local Taylor series grows exponentially fast with dimension.

Improved Fast Gauss Transform (IFGT)

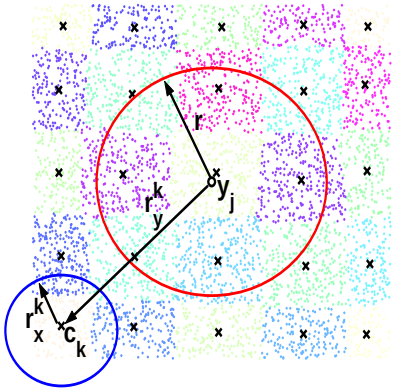
1. The number of the terms grows exponentially with dimensionality d.
   - Different series expansion–reduces the number of the expansion terms to the polynomial order.
2. The space subdivision scheme is a uniform box subdivision scheme which is inefficient in higher dimensions.
   - k-center algorithm is applied to subdivide the space which is more efficient.
3. The constant term due to the translation of the far-field Hermite series to the local Taylor series grows exponentially fast with dimension.
   - No translation – Our expansion can act both as a far-field and local expansion.

Brief idea of IFGT [12]



[12] C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In Advances in Neural Information Processing Systems, pages 15611568, 2005.

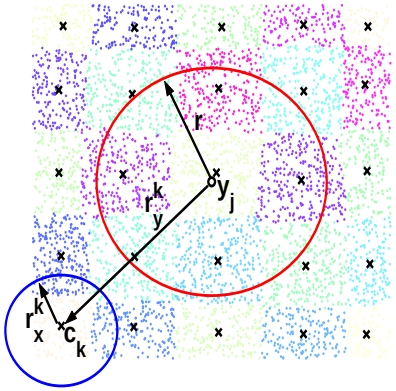
Brief idea of IFGT [12]



- **Step 0** Determine parameters of algorithm based on specified error bound, kernel bandwidth, and data distribution.

[12] C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In Advances in Neural Information Processing Systems, pages 15611568, 2005.

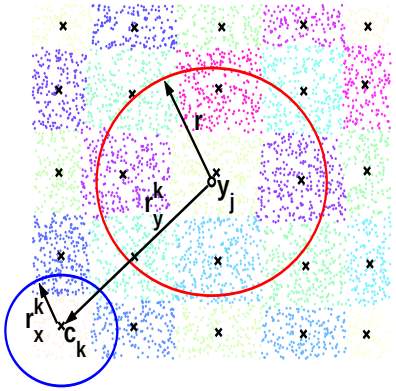
Brief idea of IFGT [12]



- **Step 0** Determine parameters of algorithm based on specified error bound, kernel bandwidth, and data distribution.
- **Step 1** Subdivide the d-dimensional space using a k-center clustering based geometric data structure ($\mathcal{O}(N \log K)$).

[12] C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In Advances in Neural Information Processing Systems, pages 15611568, 2005.

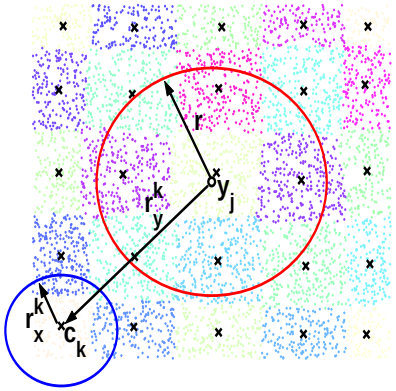
Brief idea of IFGT [12]



- **Step 0** Determine parameters of algorithm based on specified error bound, kernel bandwidth, and data distribution.
- **Step 1** Subdivide the d-dimensional space using a k-center clustering based geometric data structure ($\mathcal{O}(N \log K)$).
- **Step 2** Build a p truncated representation of kernels inside each cluster using a set of decaying basis functions ($\mathcal{O}(Nd^p)$).

[12] C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In Advances in Neural Information Processing Systems, pages 15611568, 2005.

Brief idea of IFGT [12]



- **Step 0** Determine parameters of algorithm based on specified error bound, kernel bandwidth, and data distribution.
- **Step 1** Subdivide the d-dimensional space using a k-center clustering based geometric data structure ($\mathcal{O}(N \log K)$).
- **Step 2** Build a p truncated representation of kernels inside each cluster using a set of decaying basis functions ($\mathcal{O}(Nd^p)$).
- **Step 3** Collect the influence of all the the data in a neighborhood using coefficients at cluster center and evaluate ($\mathcal{O}(Md^p)$).

[12] C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In Advances in Neural Information Processing Systems, pages 15611568, 2005.

Sample result

For example in three dimensions and 1 million training and test points [h=0.4]

- IFGT – 6 minutes.
- Direct – 34 hours.

with an error of 10^{-8}.

Separate out i and j

For any point $x_* \in \mathbf{R}^d$

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

Separate out i and j

For any point $x_* \in \mathbf{R}^d$

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

Separate out i and j

For any point $x_* \in \mathbf{R}^d$

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

Separate out i and j

For any point $x_* \in \mathbf{R}^d$

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

Crux of the algorithm

Separate this entanglement via the Taylor's series expansion of the exponentials.

Factorization via multivariate Taylor's series

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

Factorization via multivariate Taylor's series

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

The truncation number p is chosen based on the prescribed error ϵ.

Factorization via multivariate Taylor's series

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

The truncation number p is chosen based on the prescribed error ϵ. Using multi-index notation this can be written as

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

Let us ignore the error and regroup

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

$$= \sum_{|\alpha| \le p-1} C_\alpha e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^\alpha .$$

Moments are precomputed in $\mathcal{O}(N)$

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

Let us ignore the error and regroup

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

$$= \sum_{|\alpha| \le p-1} C_\alpha e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^\alpha .$$

Moments are precomputed in $\mathcal{O}(N)$

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

Evaluation of $\hat{G}(y_j)$ at M points is $\mathcal{O}(M)$.

Let us ignore the error and regroup

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

$$= \sum_{|\alpha| \le p-1} C_\alpha e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^\alpha .$$

Moments are precomputed in $\mathcal{O}(N)$

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

Evaluation of $\hat{G}(y_j)$ at M points is $\mathcal{O}(M)$.
Hence the computational complexity has reduced from the quadratic $\mathcal{O}(NM)$ to the linear $\mathcal{O}(N + M)$.

Space subdivision

- We expanded around a point x_*.

Space subdivision

- We expanded around a point x_*.
- Same x_* for all the points may require very high truncation numbers.

Space subdivision

- We expanded around a point x_*.
- Same x_* for all the points may require very high truncation numbers.
- Divide the N sources into K clusters.

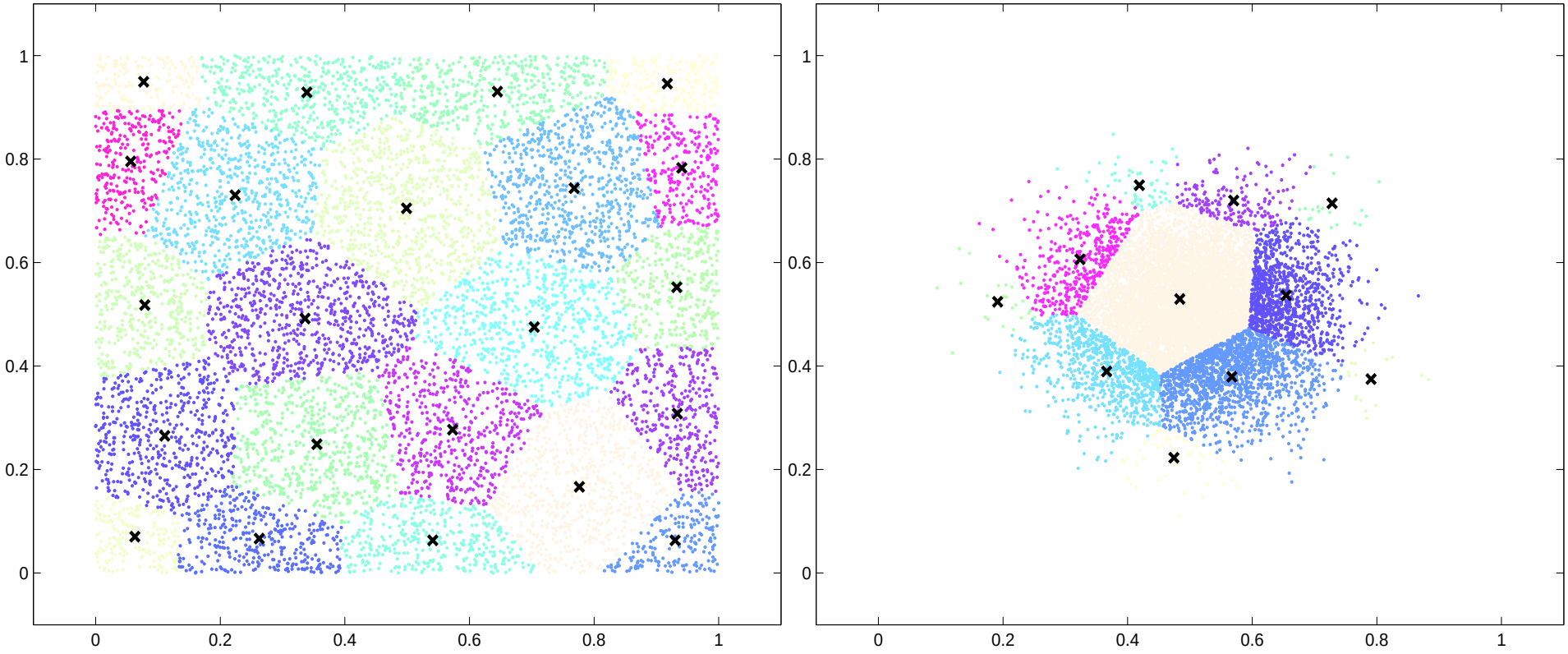
Space subdivision

- We expanded around a point x_*.
- Same x_* for all the points may require very high truncation numbers.
- Divide the N sources into K clusters.
- We use k-center clustering algorithm.

Space subdivision

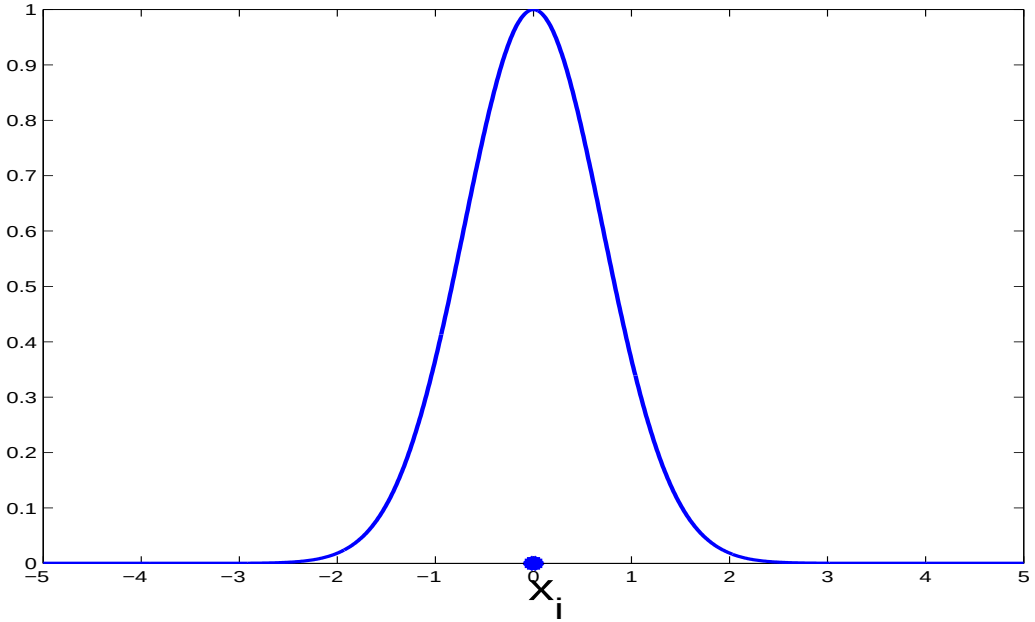
- We expanded around a point x_*.
- Same x_* for all the points may require very high truncation numbers.
- Divide the N sources into K clusters.
- We use k-center clustering algorithm.

k-center clustering example



Rapid decay of the Gaussian

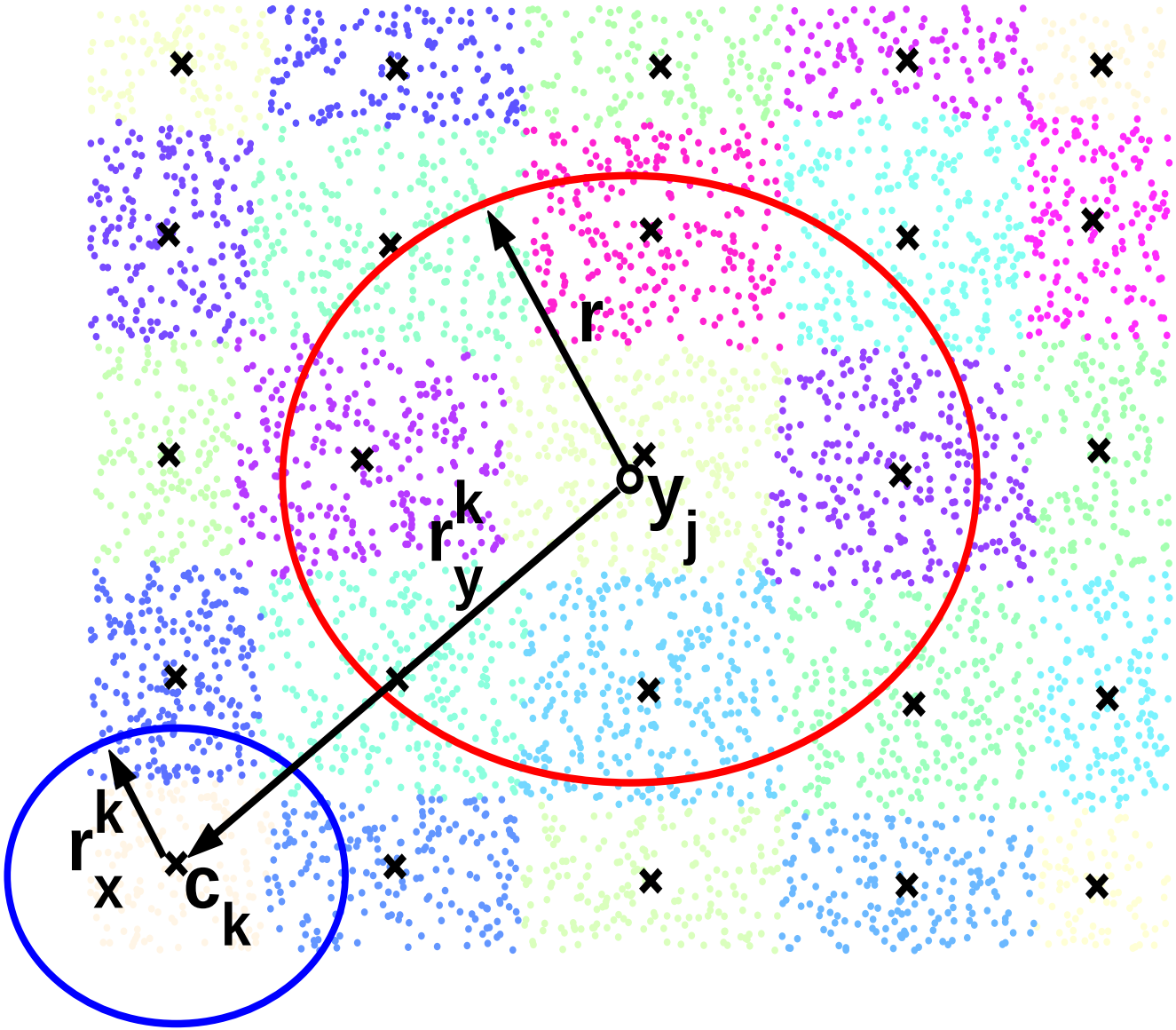
Since the Gaussian decays rapidly consider only influential clusters.



IFGT

- Step 0 Choose the parameters.
- Step 1 Subdivide the source points into K clusters.
- Step 2 Compute the cluster coefficients at the center of each cluster.
- Step 3 For each target point sum the contribution from influential clusters.

IFGT Illustration



Complexity

Computational complexity

$\mathcal{O}\left(N \log K + N r_{(p-1)d} + M n r_{(p-1)d}\right).$

Complexity

Computational complexity

$\mathcal{O}\left(N \log K + N r_{(p-1)d} + M n r_{(p-1)d}\right)$.

- $r_{(p-1)d} = \binom{p+d-1}{d}$ is the total number of d-variate monomials of degree less than or equal to $p-1$.
- The d-variate monomials can be efficiently evaluated using the Horner's rule.
- n is the maximum number of influential clusters.

Complexity

Computational complexity

$\mathcal{O}\left(N \log K + N r_{(p-1)d} + M n r_{(p-1)d}\right)$.

- $r_{(p-1)d} = \binom{p+d-1}{d}$ is the total number of d-variate monomials of degree less than or equal to $p-1$.
- The d-variate monomials can be efficiently evaluated using the Horner's rule.
- n is the maximum number of influential clusters.

Storage complexity

$\mathcal{O}(K r_{(p-1)d} + N + M)$

Choosing the parameters

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

- K (the number of clusters),
- p (the truncation number),
- and the cut off radius

such that for any target point y_j we can guarantee that

$$\frac{|\hat{G}(y_j) - G(y_j)|}{Q} \leq \epsilon,$$

where $Q = \sum_{i=1}^{N} |q_i|$.

Automatic parameter selection

- The error bound proposed in the original paper was incorrect and not tight to be useful in practice.
- No strategy for choosing the parameters to achieve the desired bound.
- We propose automatic choice of the algorithm parameters [13] .

[13] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov, Fast computation of sums of Gaussians in high dimensions. CS-TR-4767, Department of Computer Science, University of Maryland, CollegePark, 2005.

Automatic parameter selection

- The error bound proposed in the original paper was incorrect and not tight to be useful in practice.
- No strategy for choosing the parameters to achieve the desired bound.
- We propose automatic choice of the algorithm parameters [13] .

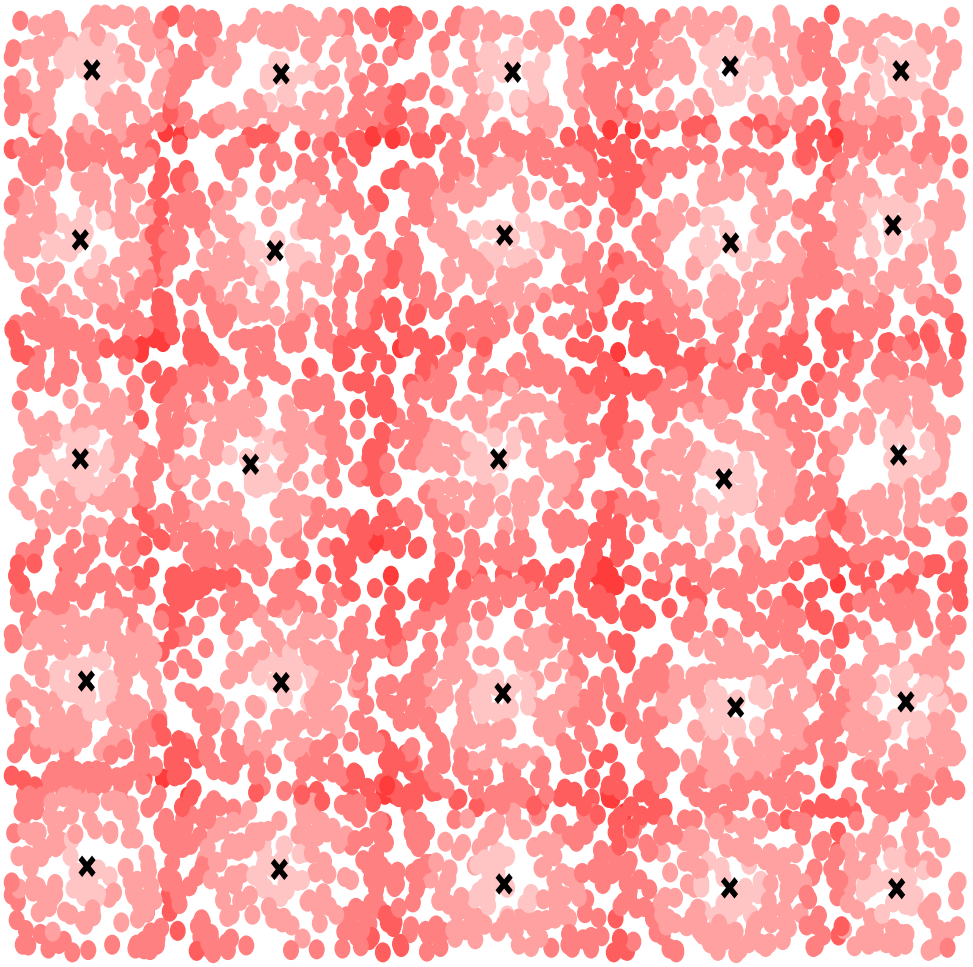
Strategy

- Derive tight bounds for the error.
- Choose the parameters such that the bound is less than ϵ.

[13] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov, Fast computation of sums of Gaussians in high dimensions. CS-TR-4767, Department of Computer Science, University of Maryland, CollegePark, 2005.

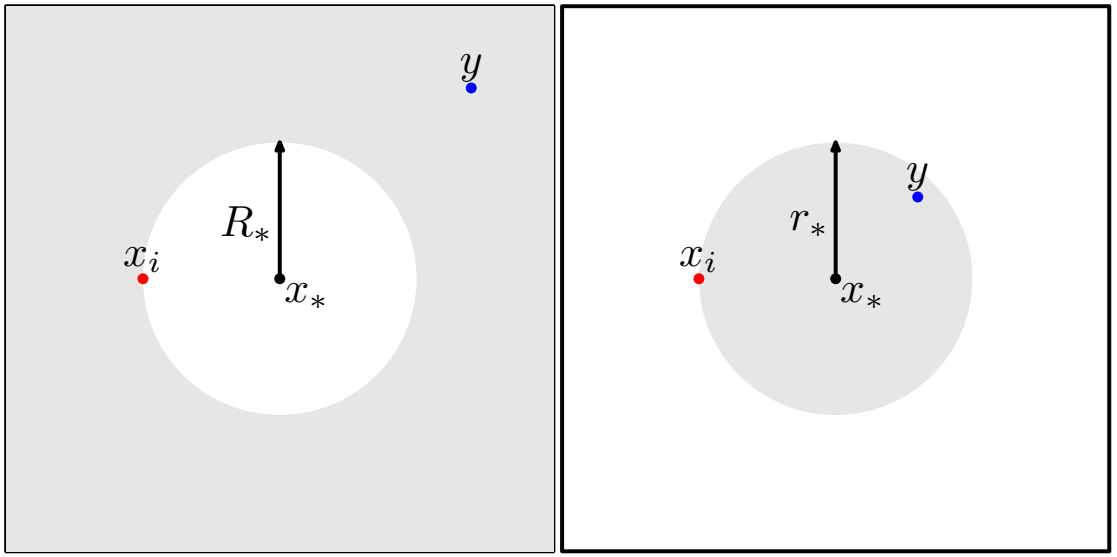
Point-wise truncation numbers

- A tighter point-wise error bound.
- Truncation number for each source is different.



Fast multipole methods

- The FGT belongs to a more general class of methods called fast multipole methods [14].
- The general fast multipole methods use two kinds of factorization
- Far-field expansion and Local expansion.



[14] Greengard, L. and Rokhlin, V. 1987. A fast algorithm for particle simulations. J. of Comp. Physics 73, 2, 325-348.

Comparison with FGT expansions

Far-field Hermite expansion

$$e^{-\|y-x_i\|^2/h^2} = \sum_{\alpha \geq 0} \left[\frac{1}{\alpha!} \left(\frac{x_i - x_*}{h} \right)^{\alpha} \right] h_\alpha \left(\frac{y - x_*}{h} \right)$$

Local Taylor expansion

$$e^{-\|y-x_i\|^2/h^2} = \sum_{\beta \geq 0} \left[\frac{1}{\beta!} h_\beta \left(\frac{x_i - x_*}{h} \right) \right] \left(\frac{y - x_*}{h} \right)^{\beta}$$

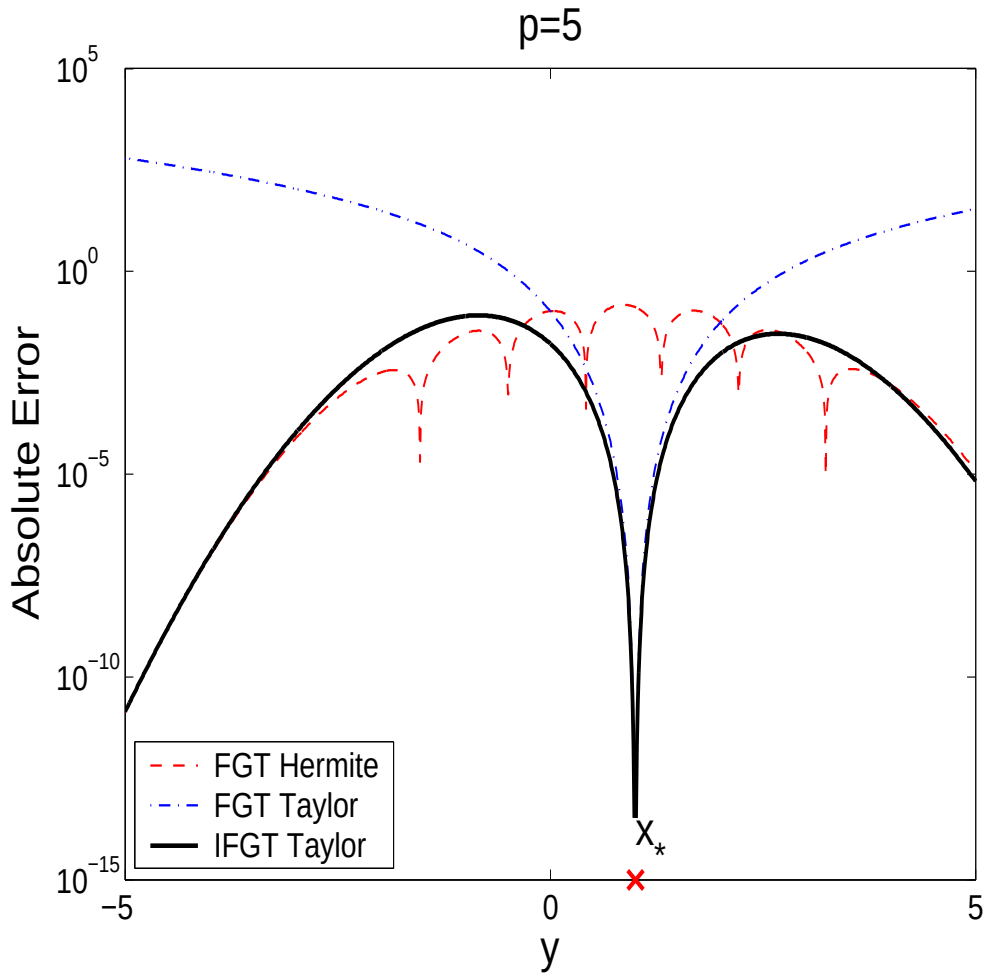
Compare this with the

Single IFGT expansion

$$e^{-\|y-x_i\|^2/h^2} = \sum_{|\alpha| \geq 0} \left[\frac{2^\alpha}{\alpha!} e^{-\|x_i - x_*\|^2/h^2} \left(\frac{x_i - x_*}{h} \right)^{\alpha} \right] e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^{\alpha}$$

IFGT expansion is both local as well as far-field

Hence we avoid the expensive translation operation.



FGT vs IFGT complexity

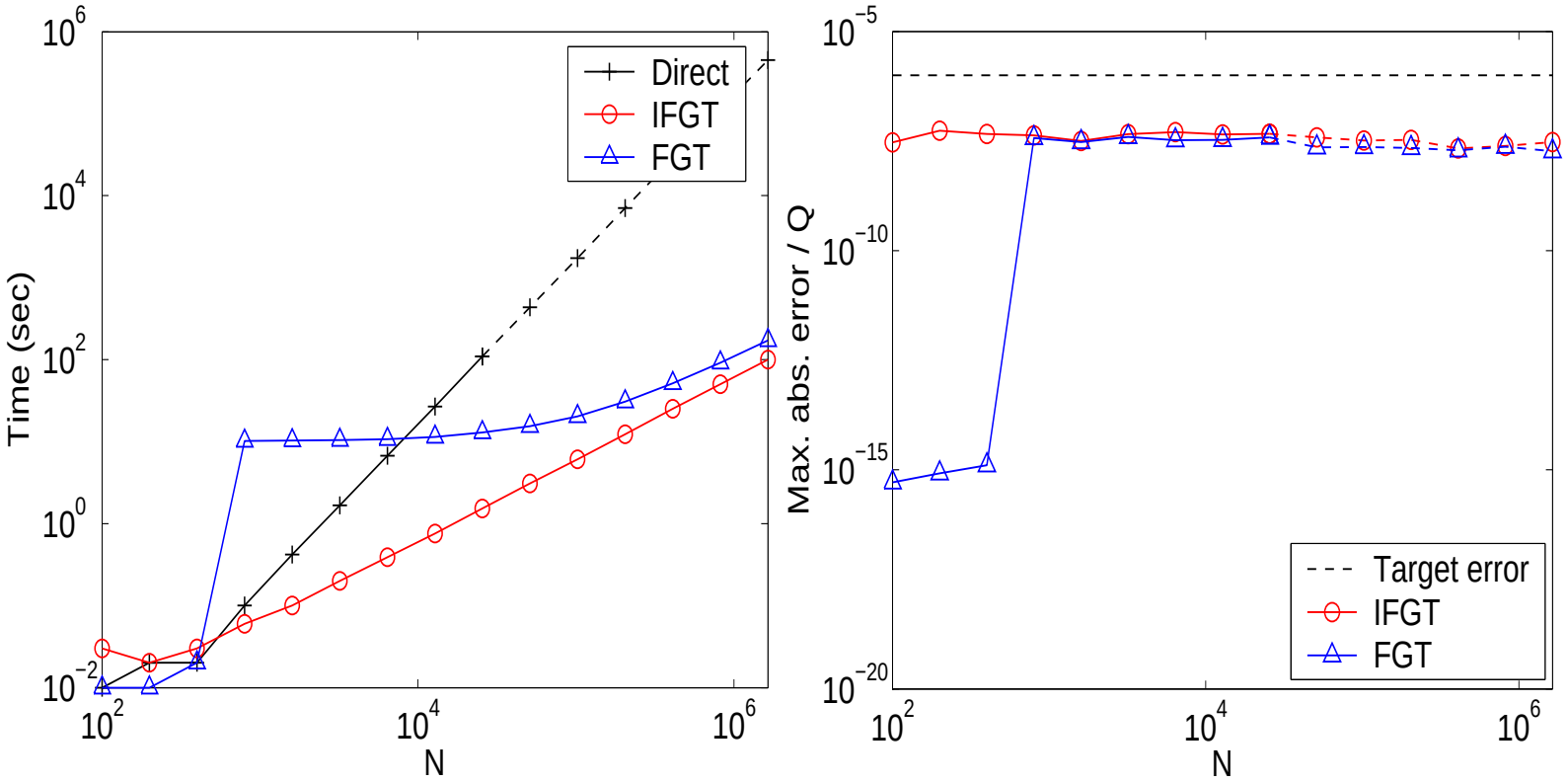
d	FGT					IFGT		
	# of boxes (N_{side}^d)	p	# of terms (p^d)	n	Constant term	K	p	# of terms ($r_{(p-1)d}$)
1	3	9	9	2	7.0+002	5	9	9
2	9	10	100	2	1.5e+005	7	15	120
3	27	10	1000	2	1.9e+007	15	16	816
4	81	11	14641	2	3.6e+009	29	17	4845
5	243	11	161051	2	4.3e+011	31	20	42504
6	729	12	2985984	2	9.0e+013	62	20	177100
7	2187	14	105413504	2	3.7e+016	67	22	1184040

FGT vs IFGT complexity

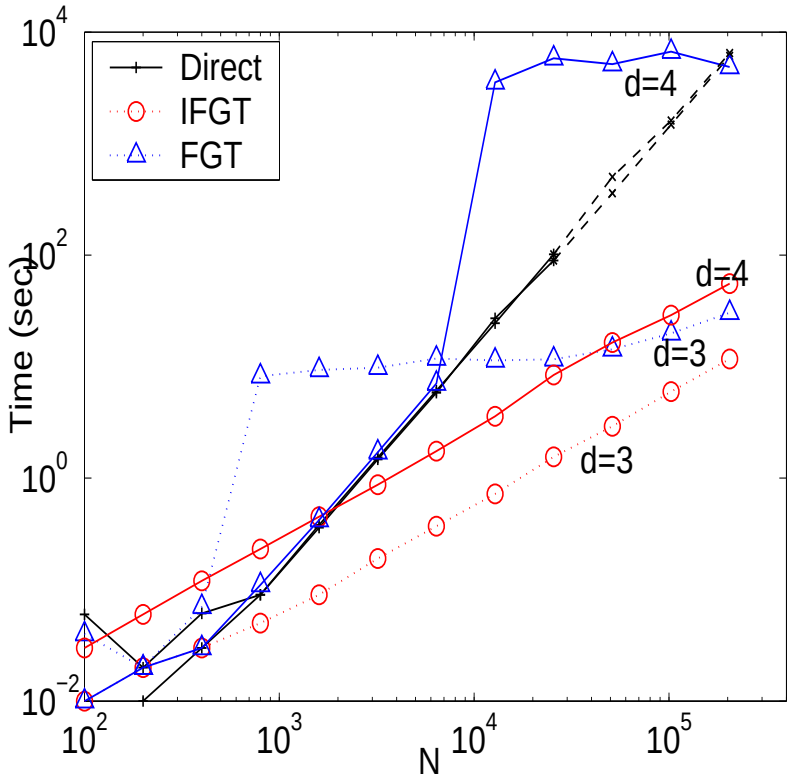
d	FGT					IFGT		
	# of boxes (N_{side}^d)	p	# of terms (p^d)	n	Constant term	K	p	# of terms ($r_{(p-1)d}$)
1	3	9	9	2	7.0+002	5	9	9
2	9	10	100	2	1.5e+005	7	15	120
3	27	10	1000	2	1.9e+007	15	16	816
4	81	11	14641	2	3.6e+009	29	17	4845
5	243	11	161051	2	4.3e+011	31	20	42504
6	729	12	2985984	2	9.0e+013	62	20	177100
7	2187	14	105413504	2	3.7e+016	67	22	1184040

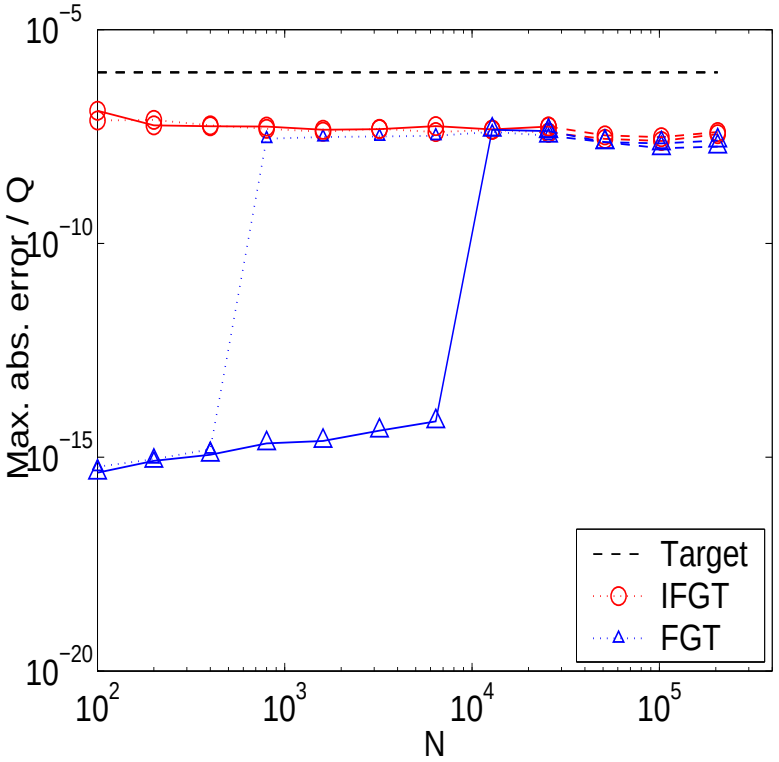
Also IFGT simple to code.

Speedup as a function of N [$d = 3$ and $h = 1.0$]



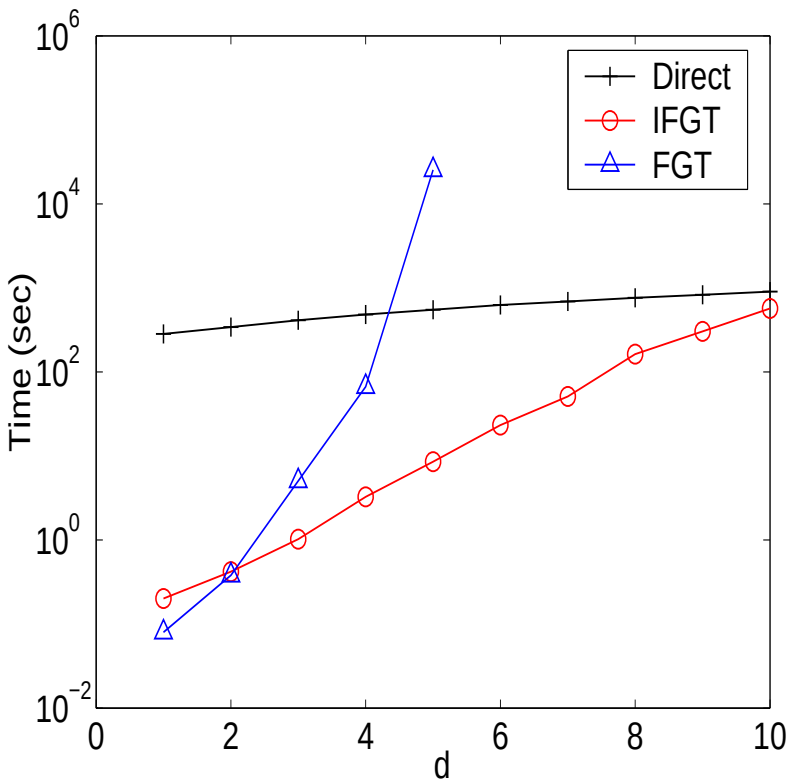
Speedup as a function of N and d

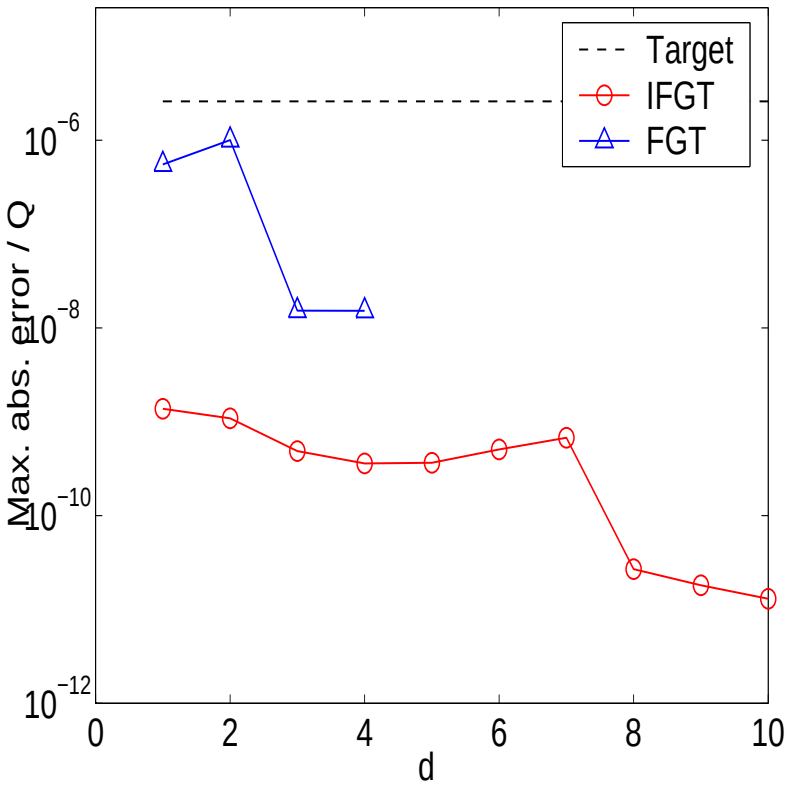




Speedup as a function of d $[h = 2.0]$

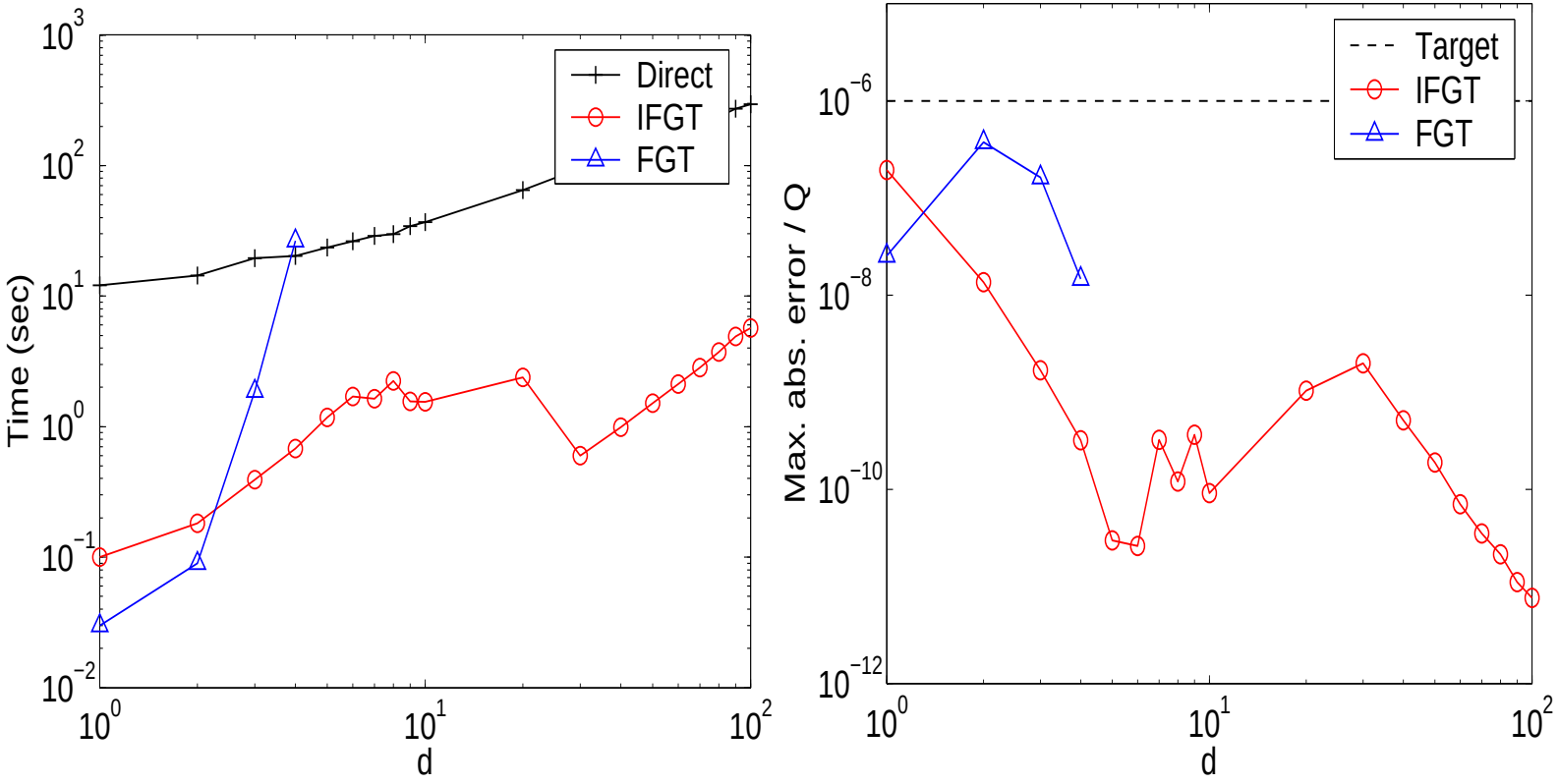
FGT cannot be run for $d > 3$





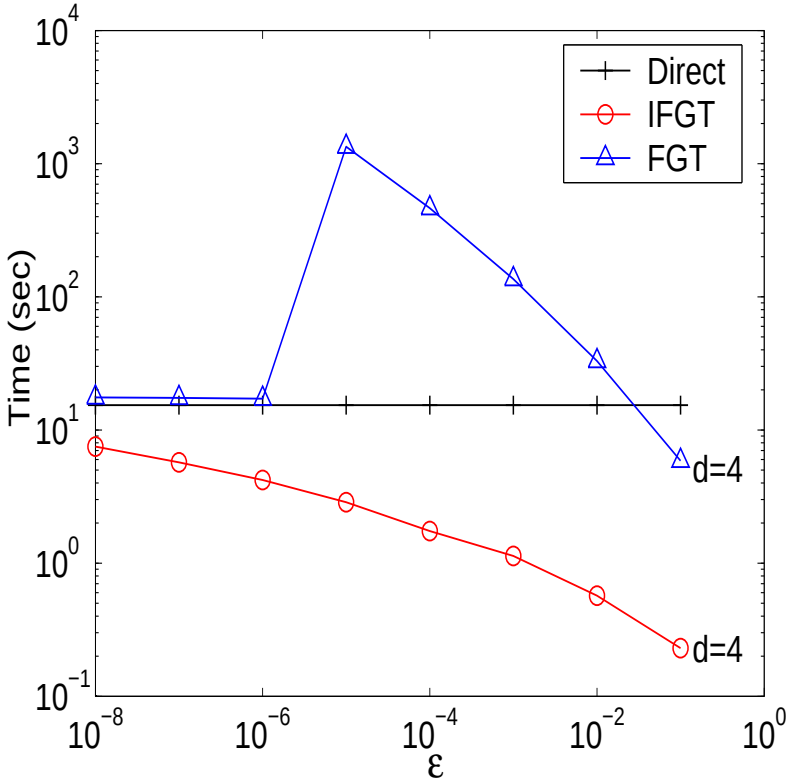
Speedup as a function of d $[h = \sqrt{d}]$

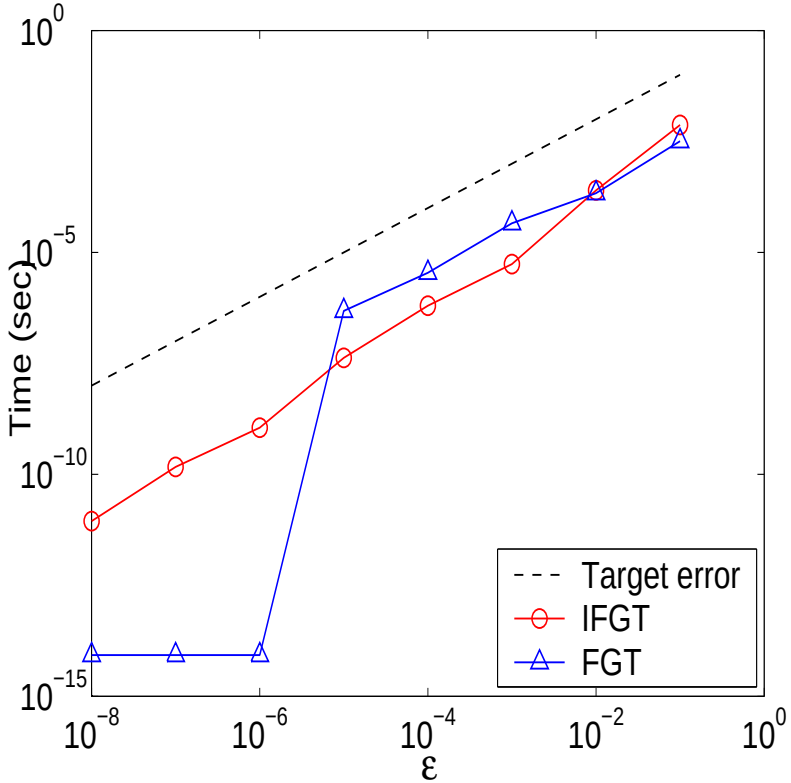
IFGT scales well with d.



Speedup as a function of ϵ

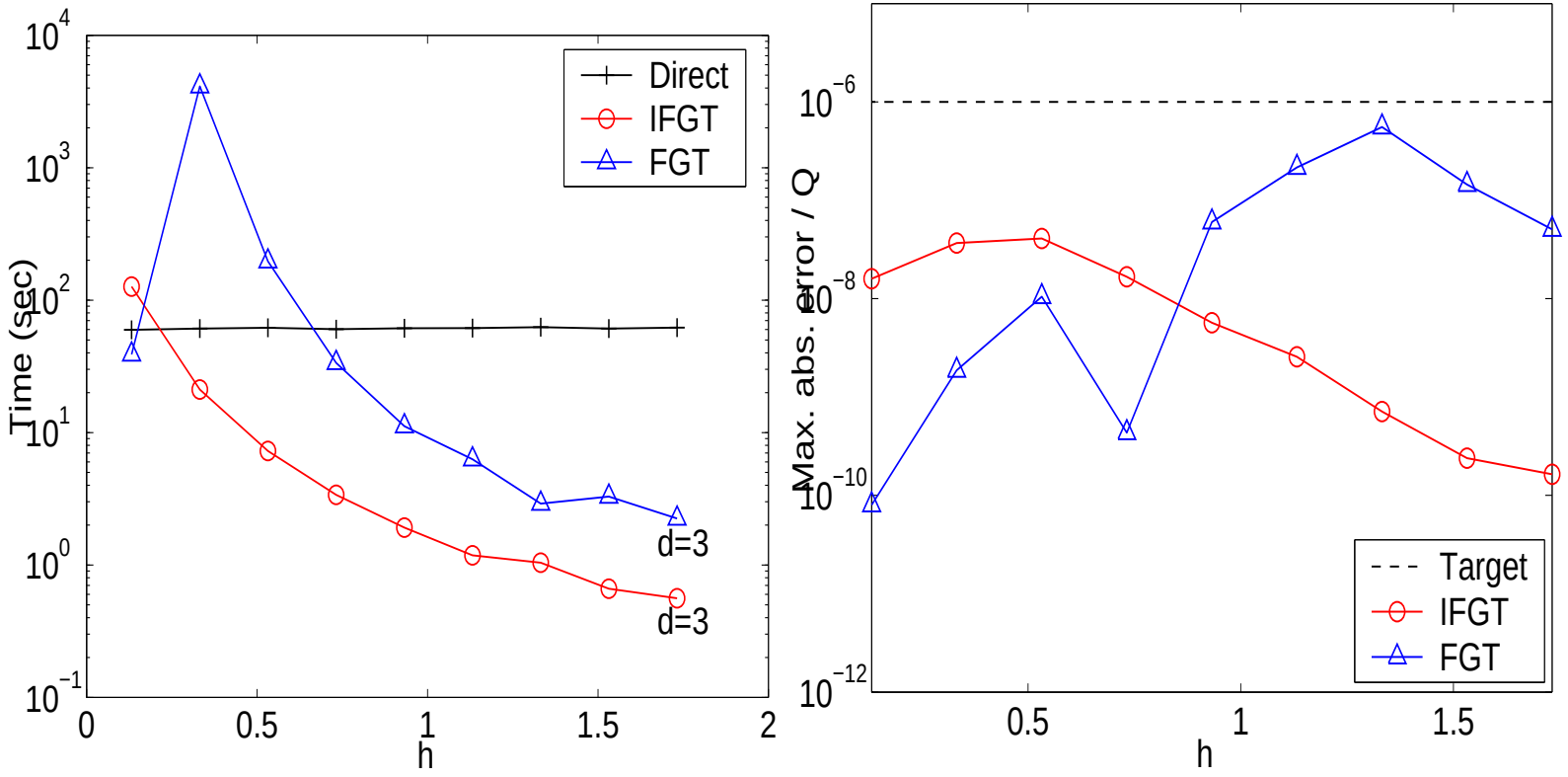
Better speedup for lower precision.





Speedup as a function of h

Better speedup at larger bandwidths.



Hyperparameter selection for kernel methods

- The IFGT can be used in any kernel machine where we encounter sums of Gaussians.

Hyperparameter selection for kernel methods

- The IFGT can be used in any kernel machine where we encounter sums of Gaussians.
- Most kernel methods require choosing some hyperparameters (e.g. bandwidth h of the kernel).

Hyperparameter selection for kernel methods

- The IFGT can be used in any kernel machine where we encounter sums of Gaussians.
- Most kernel methods require choosing some hyperparameters (e.g. bandwidth h of the kernel).
- Optimal procedures to choose these parameters are $\mathcal{O}(N^2)$.

Hyperparameter selection for kernel methods

- The IFGT can be used in any kernel machine where we encounter sums of Gaussians.
- Most kernel methods require choosing some hyperparameters (e.g. bandwidth h of the kernel).
- Optimal procedures to choose these parameters are $\mathcal{O}(N^2)$.
- Most of these procedures involve solving some optimization which involves taking the derivatives of kernel sums.

Hyperparameter selection for kernel methods

- The IFGT can be used in any kernel machine where we encounter sums of Gaussians.
- Most kernel methods require choosing some hyperparameters (e.g. bandwidth h of the kernel).
- Optimal procedures to choose these parameters are $\mathcal{O}(N^2)$.
- Most of these procedures involve solving some optimization which involves taking the derivatives of kernel sums.
- The derivatives of Gaussian sums involve sums of products of Hermite polynomials and Gaussians.
- $G_r(y_j) = \sum_{i=1}^{N} q_i H_r\left(\frac{y_j - x_i}{h}\right) e^{-(y_j - x_i)^2/2h^2} \quad j = 1, \ldots, M.$

Hyperparameter selection for kernel methods

- The IFGT can be used in any kernel machine where we encounter sums of Gaussians.
- Most kernel methods require choosing some hyperparameters (e.g. bandwidth h of the kernel).
- Optimal procedures to choose these parameters are $\mathcal{O}(N^2)$.
- Most of these procedures involve solving some optimization which involves taking the derivatives of kernel sums.
- The derivatives of Gaussian sums involve sums of products of Hermite polynomials and Gaussians.
- $G_r(y_j) = \sum_{i=1}^{N} q_i H_r\left(\frac{y_j - x_i}{h}\right) e^{-(y_j - x_i)^2/2h^2} \quad j = 1, \ldots, M.$
- Fast algorithms have been developed for such sums.

Kernel density estimation

- The most popular method for density estimation is the kernel density estimator (KDE).

$$\widehat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K \left(\frac{x - x_i}{h} \right)$$

Kernel density estimation

- The most popular method for density estimation is the kernel density estimator (KDE).

$$\widehat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

- IFGT can be directly used to accelerate KDE.

Kernel density estimation

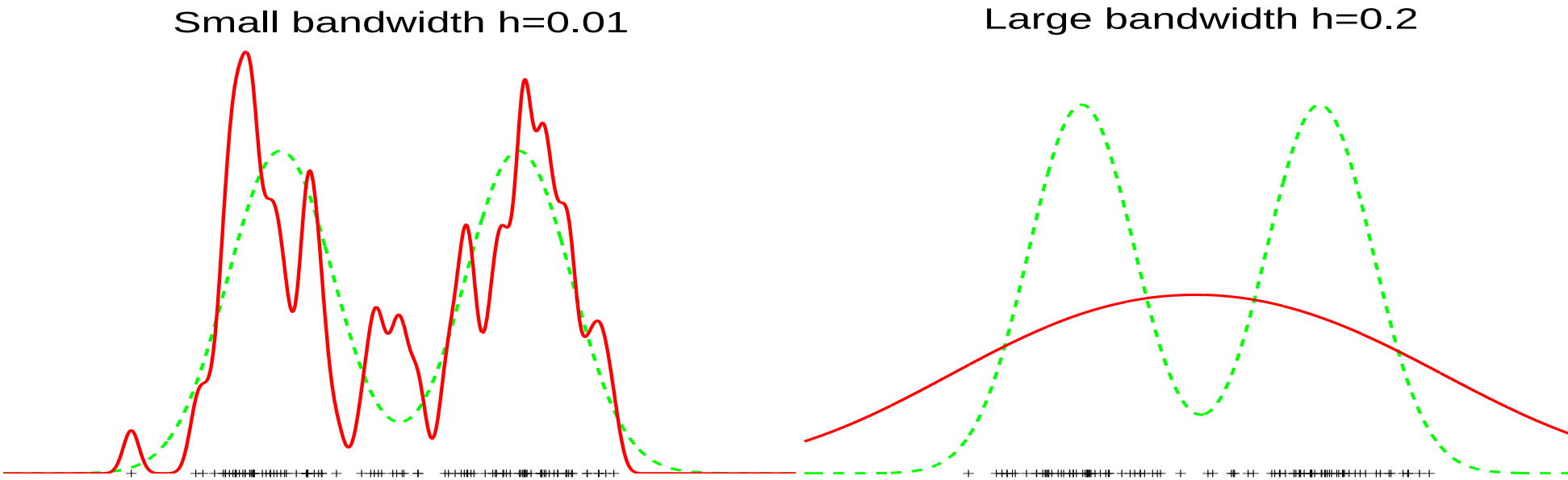
- The most popular method for density estimation is the kernel density estimator (KDE).

$$\widehat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

- IFGT can be directly used to accelerate KDE.
- Efficient use of KDE requires choosing h optimally.

The bandwidth h is a very crucial parameter

- As h decreases towards 0, the number of modes increases to the number of data points and the KDE is very noisy.
- As h increases towards ∞, the number of modes drops to 1, so that any interesting structure has been smeared away and the KDE just displays a unimodal pattern.



Fast optimal bandwidth selection

- The state-of-the-art method for optimal bandwidth selection for kernel density estimation scales as $\mathcal{O}(N^2)$.

[15] Fast optimal bandwidth selection for kernel density estimation. Vikas C. Raykar and Ramani Duraiswami, In Proceedings of the sixth SIAM International Conference on Data Mining, Bethesda, April 2006, pp. 524-528.

Fast optimal bandwidth selection

- The state-of-the-art method for optimal bandwidth selection for kernel density estimation scales as $\mathcal{O}(N^2)$.
- We present a fast computational technique that scales as $\mathcal{O}(N)$ [15].

[15] Fast optimal bandwidth selection for kernel density estimation. Vikas C. Raykar and Ramani Duraiswami, In Proceedings of the sixth SIAM International Conference on Data Mining, Bethesda, April 2006, pp. 524-528.

Fast optimal bandwidth selection

- The state-of-the-art method for optimal bandwidth selection for kernel density estimation scales as $\mathcal{O}(N^2)$.
- We present a fast computational technique that scales as $\mathcal{O}(N)$ [15].
- The core part is a fast $\epsilon - exact$ algorithm for kernel density derivative estimation which reduces the computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

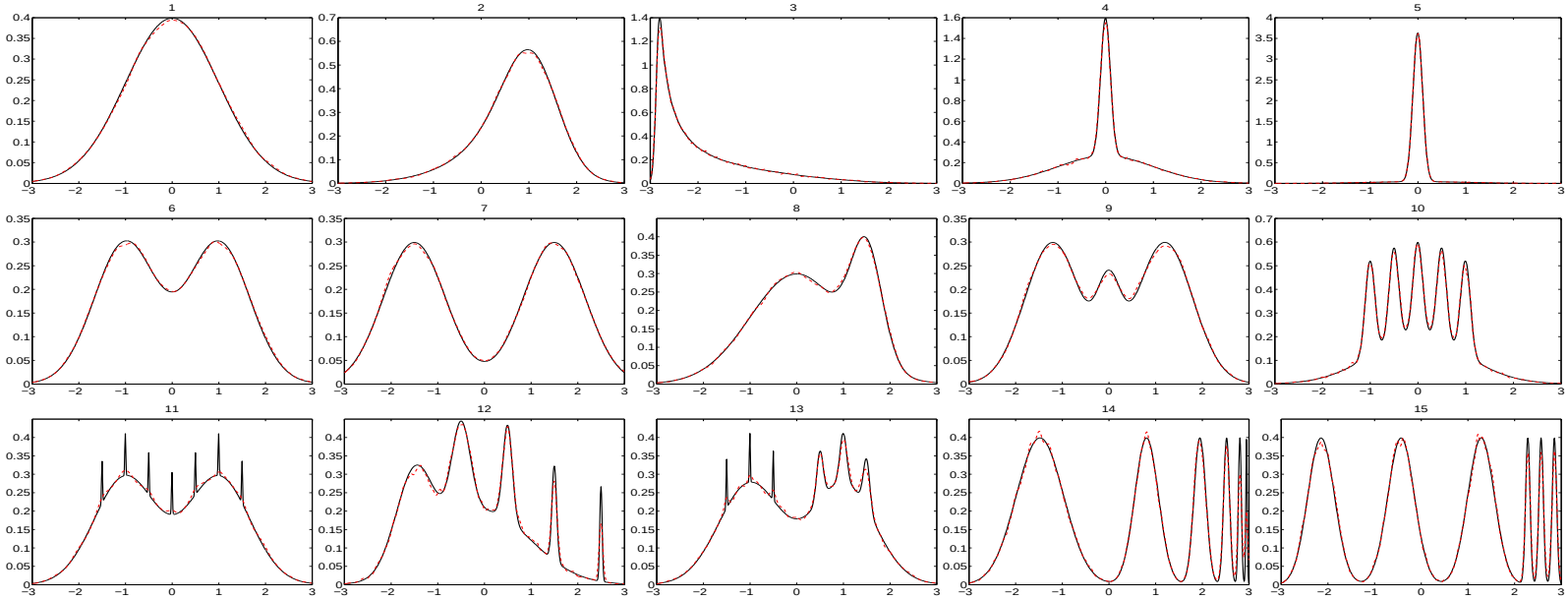
[15] Fast optimal bandwidth selection for kernel density estimation. Vikas C. Raykar and Ramani Duraiswami, In Proceedings of the sixth SIAM International Conference on Data Mining, Bethesda, April 2006, pp. 524-528.

Fast optimal bandwidth selection

- The state-of-the-art method for optimal bandwidth selection for kernel density estimation scales as $\mathcal{O}(N^2)$.
- We present a fast computational technique that scales as $\mathcal{O}(N)$ [15].
- The core part is a fast $\epsilon - exact$ algorithm for kernel density derivative estimation which reduces the computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.
- For example for $N = 409,600$ points.
  - Direct evaluation $\rightarrow$ 12.76 **hours**.
  - Fast evaluation $\rightarrow$ 65 **seconds** with an error of around 10^{-12}.

[15] Fast optimal bandwidth selection for kernel density estimation. Vikas C. Raykar and Ramani Duraiswami, In Proceedings of the sixth SIAM International Conference on Data Mining, Bethesda, April 2006, pp. 524-528.

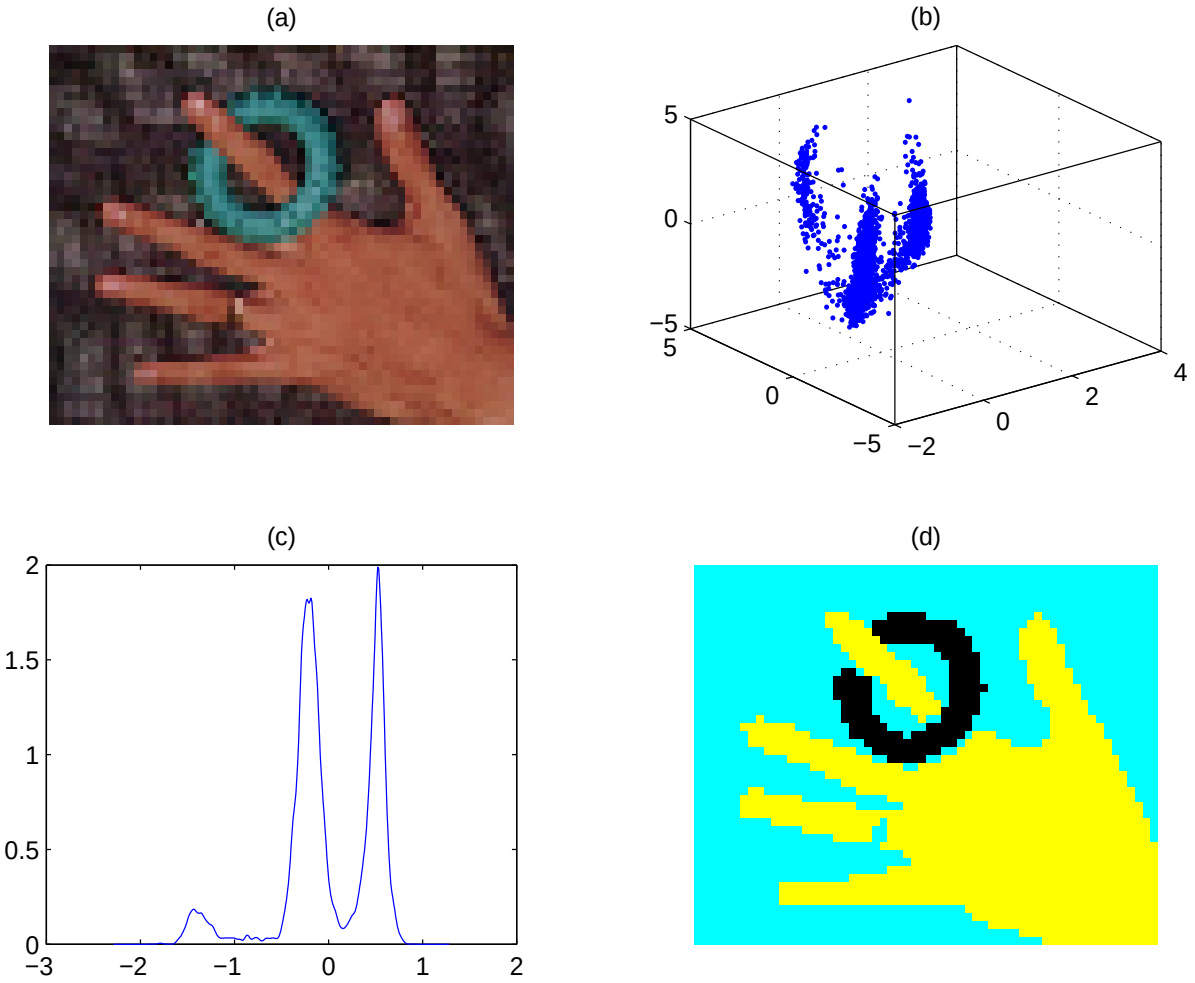
Marron Wand normal mixtures



Speedup for Marron Wand normal mixtures

	h_{direct}	h_{fast}	T_{direct} (sec)	T_{fast} (sec)	Speedup	Rel. Err.
1	0.122213	0.122215	4182.29	64.28	65.06	1.37e-005
2	0.082591	0.082592	5061.42	77.30	65.48	1.38e-005
3	0.020543	0.020543	8523.26	101.62	83.87	1.53e-006
4	0.020621	0.020621	7825.72	105.88	73.91	1.81e-006
5	0.012881	0.012881	6543.52	91.11	71.82	5.34e-006
6	0.098301	0.098303	5023.06	76.18	65.93	1.62e-005
7	0.092240	0.092240	5918.19	88.61	66.79	6.34e-006
8	0.074698	0.074699	5912.97	90.74	65.16	1.40e-005
9	0.081301	0.081302	6440.66	89.91	71.63	1.17e-005
10	0.024326	0.024326	7186.07	106.17	67.69	1.84e-006
11	0.086831	0.086832	5912.23	90.45	65.36	1.71e-005
12	0.032492	0.032493	8310.90	119.02	69.83	3.83e-006
13	0.045797	0.045797	6824.59	104.79	65.13	4.41e-006
14	0.027573	0.027573	10485.48	111.54	94.01	1.18e-006
15	0.023096	0.023096	11797.34	112.57	104.80	7.05e-007

Speedup for projection pursuit



Speedup for projection pursuit

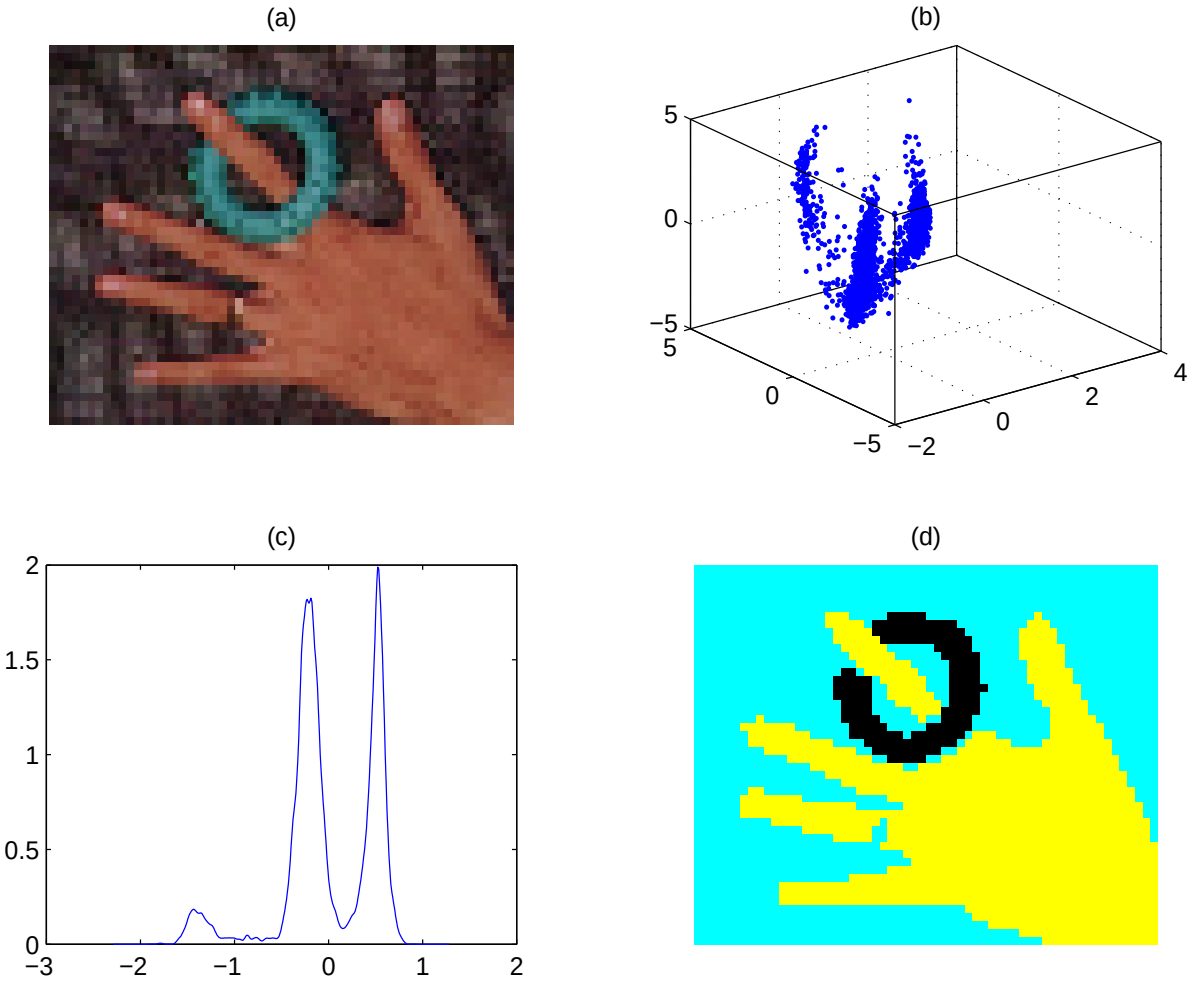


Image segmentation via PP with optimal KDE took 15 minutes while that using the direct method takes around 7.5 hours.

Outline of the proposal

Gaussian processes for regression

Gaussian processes for regression

- Gaussian processes handle nonparametric regression in a Bayesian framework.

Gaussian processes for regression

- Gaussian processes handle nonparametric regression in a Bayesian framework.
- The regression function is represented by an ensemble of functions, on which we place a Gaussian prior.

Gaussian processes for regression

- Gaussian processes handle nonparametric regression in a Bayesian framework.
- The regression function is represented by an ensemble of functions, on which we place a Gaussian prior.
- This prior is updated in the light of the training data.

Gaussian processes for regression

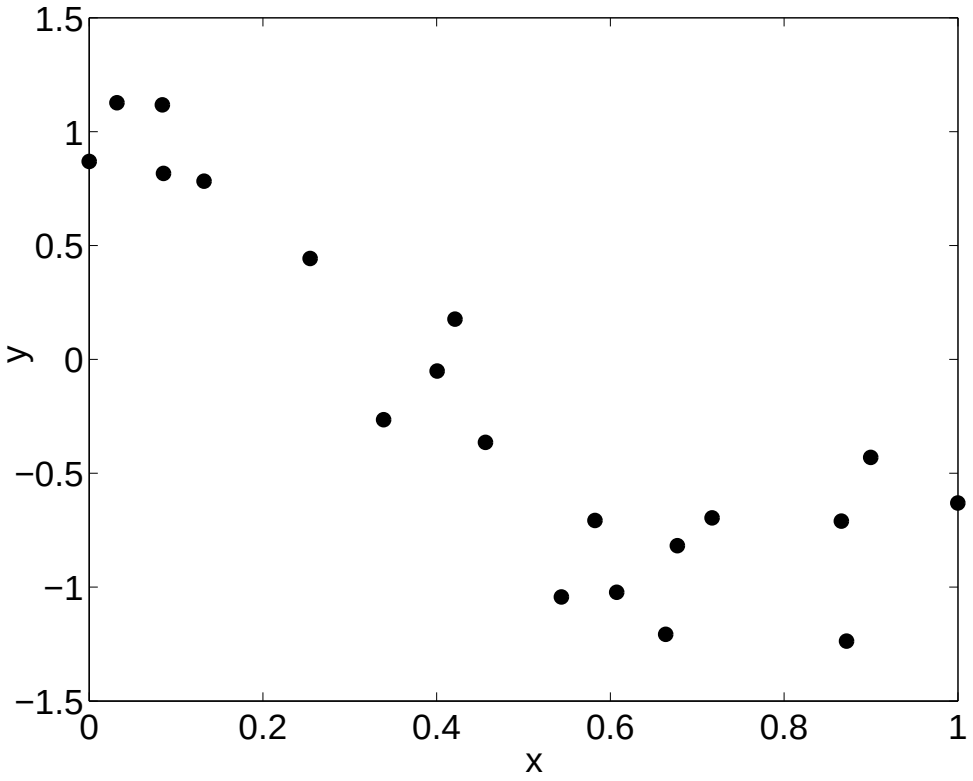
- Gaussian processes handle nonparametric regression in a Bayesian framework.
- The regression function is represented by an ensemble of functions, on which we place a Gaussian prior.
- This prior is updated in the light of the training data.
- As a result we obtain predictions together with valid estimates of uncertainty.

Gaussian process regression

Gaussian process regression

Regression problem

- Training data $\mathfrak{T} = \{x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^N$
- Predict y for a new x.



Gaussian process model

Model

$y = f(x) + \varepsilon$

Gaussian process model

Model

$y = f(x) + \varepsilon$

- ε is $\mathcal{N}(0, \sigma^2)$.

Gaussian process model

Model

$y = f(x) + \varepsilon$

- ε is $\mathcal{N}(0, \sigma^2)$.
- $f(x)$ is a zero-mean Gaussian process with covariance function $K(x, x^{'})$.
- Most common covariance function is the Gaussian.

Gaussian process model

Model

$y = f(x) + \varepsilon$

- ε is $\mathcal{N}(0, \sigma^2)$.
- $f(x)$ is a zero-mean Gaussian process with covariance function $K(x, x^{'})$.
- Most common covariance function is the Gaussian.

Gaussian process model

Model

$y = f(x) + \varepsilon$

- ε is $\mathcal{N}(0, \sigma^2)$.
- $f(x)$ is a zero-mean Gaussian process with covariance function $K(x, x')$.
- Most common covariance function is the Gaussian.

Infer the posterior

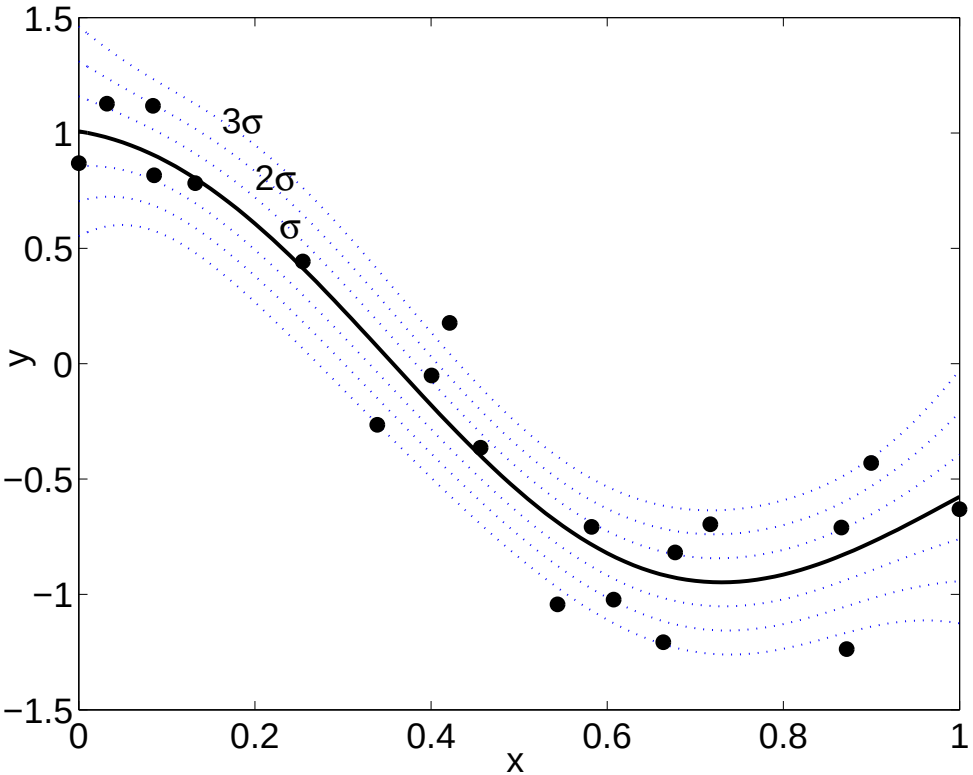
Given the training data $\mathfrak{T}$ and a new input x_* our task is to compute the posterior $p(f_*|x_*, \mathfrak{T})$.

Solution

- The posterior is a Gaussian.
- The mean is used as the prediction.
- The variance is the uncertainty associated with the prediction.

Solution

- The posterior is a Gaussian.
- The mean is used as the prediction.
- The variance is the uncertainty associated with the prediction.



Direct Training

$$\xi = (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}$$

- Direct computation of the inverse of a matrix requires $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ storage.
- Impractical even for problems of moderate size (typically a few thousands).
- For example N=25,600 takes around 10 hours, assuming you have enough RAM.

Iterative methods

An effective way is to solve the following large scale linear system using iterative methods.
$(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}$.

- The iterative method generates a sequence of approximate solutions ξ_k at each step which converge to the true solution ξ.
- Since $\mathbf{K} + \sigma^2 \mathbf{I}$ is symmetric and positive definite we can use the *conjugate-gradient* method.
- Given a tolerance parameter $0 < \eta < 1$ a practical conjugate gradient scheme iterates till it computes a vector ξ_k such that $\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_k\|_2 \leq \eta \|\mathbf{y} - \widetilde{\mathbf{K}}\xi_0\|_2$.

Computational cost of conjugate-gradient

- Requires *one matrix-vector multiplication* and $5N$ flops per iteration.
- Four vectors of length N are required for storage.
- Hence computational cost now reduces to $\mathcal{O}(kN^2)$.
- For example N=25,600 takes around 17 minutes (compare to 10 hours).

CG+IFGT

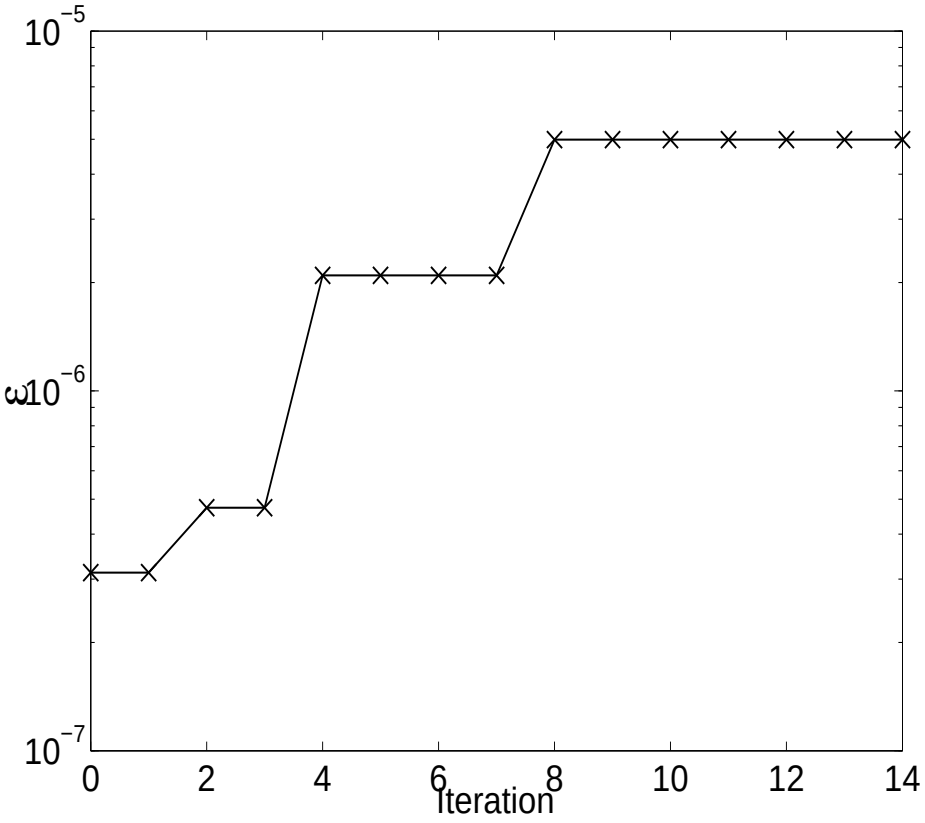
- The core computational step in each conjugate-gradient iteration is the multiplication of the matrix $\mathbf{K}$ with a vector, say $\mathbf{q}$.
- Coupled with the CG the IFGT reduces the computational cost of GP regression to $\mathcal{O}(N)$.
- For example N=25,600 takes around 3 secs. (compare to 10 hours[direct] or 17 minutes[CG]).

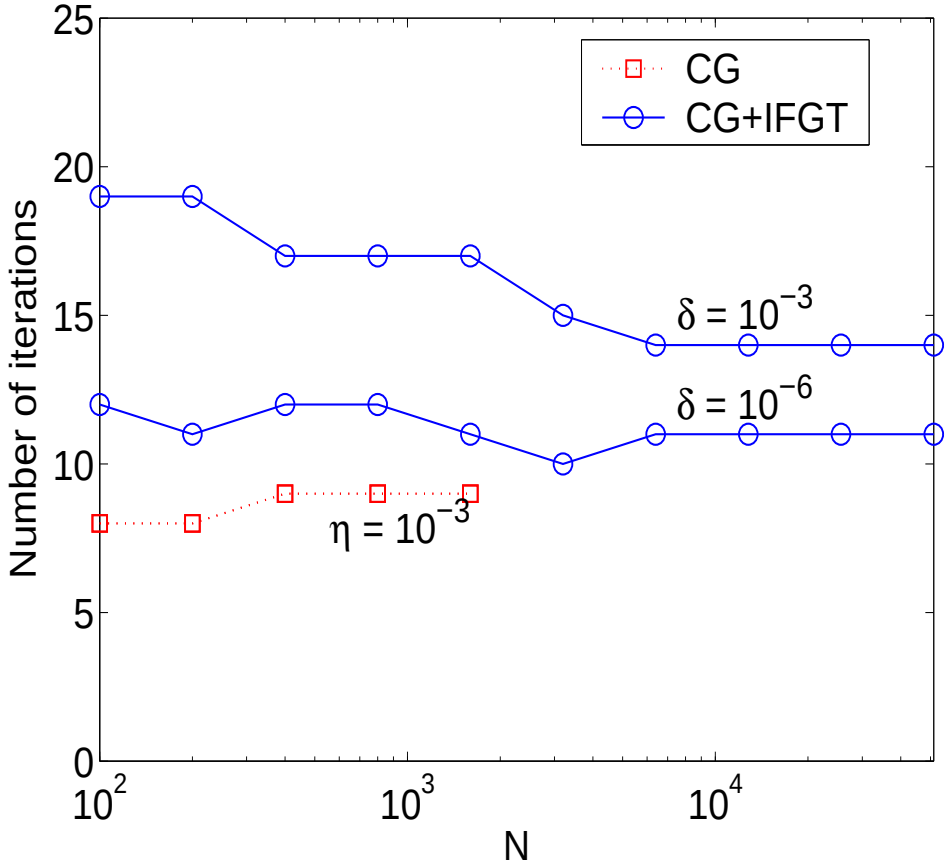
Computational cost

$\widetilde{\mathbf{K}} = \mathbf{K} + \sigma^2 \mathbf{I}$	*Direct Inversion*		*Conjugate gradient*		*Conjugate gradient +IFGT*	
	Time	Space	Time	Space	Time	Space
Training phase $\xi = \widetilde{\mathbf{K}}^{-1}\mathbf{y}$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Mean prediction $y = \mathbf{k}(x)^T \xi$	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Uncertainty $\mathbf{k}(x,x) - \mathbf{k}(x)^T \widetilde{\mathbf{K}}^{-1} \mathbf{k}(x)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$

How to choose ϵ for inexact CG?

Use the theory of inexact Krylov subspace methods[16]
Matrix-vector product may be performed in an increasingly inexact manner as the iteration progresses and still allow convergence to the solution.





[16] V. Simoncini and D. B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. SIAM J. Sci. Comput., 25(2):454–477, 2004.

IFGT with variable scales

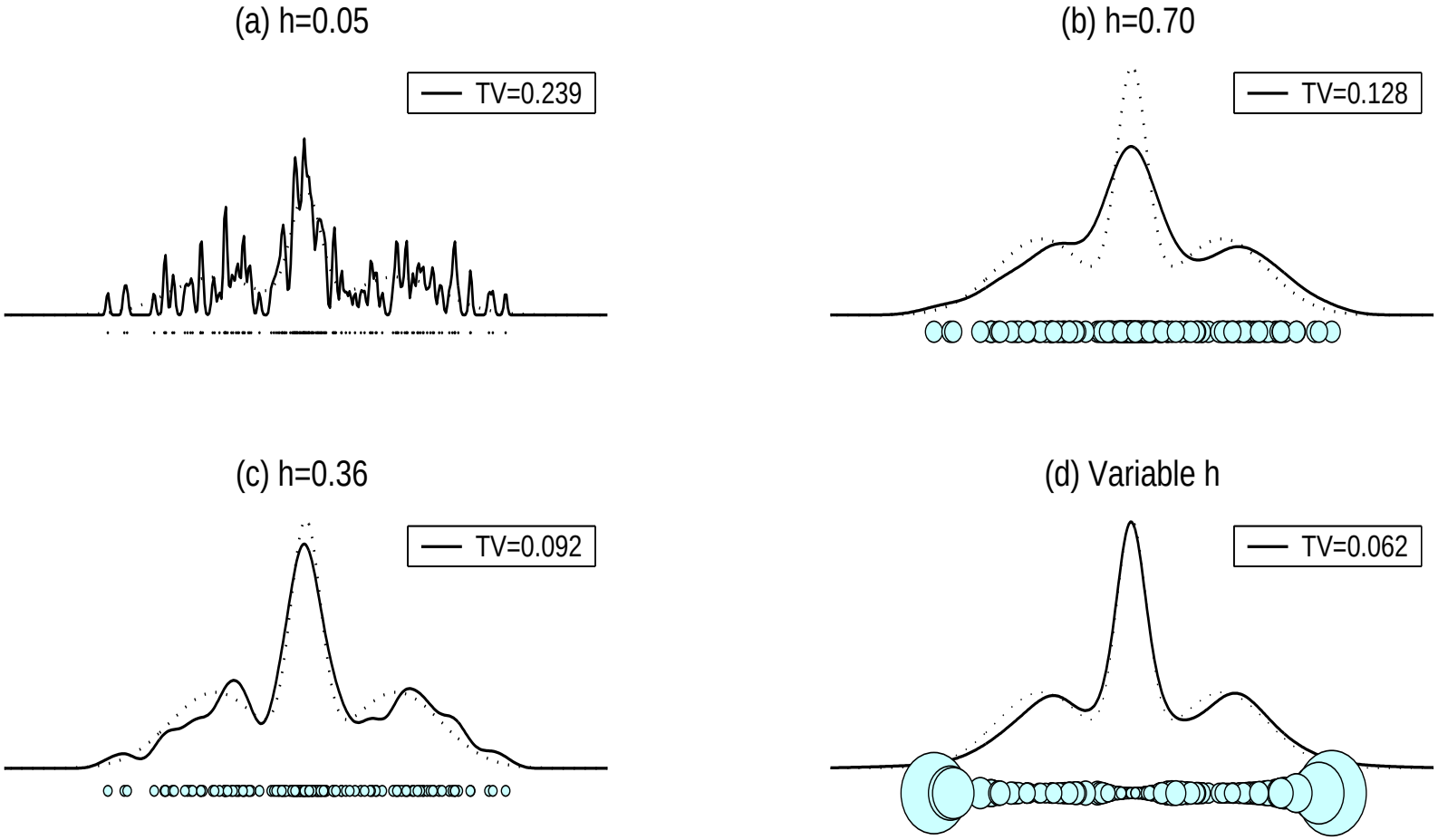
$$G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2 / h_i^2}.$$

Approach: Build a composite factorization that builds a Taylor series for h_i as well.

Resulting IFGT runs at about the same speed.

For example for N = M = 1,024,000 while the direct evaluation takes around 2.6 days the fast evaluation requires only 4.65 minutes with an error of around 10^{-5}.

Variable bandwidth density estimation



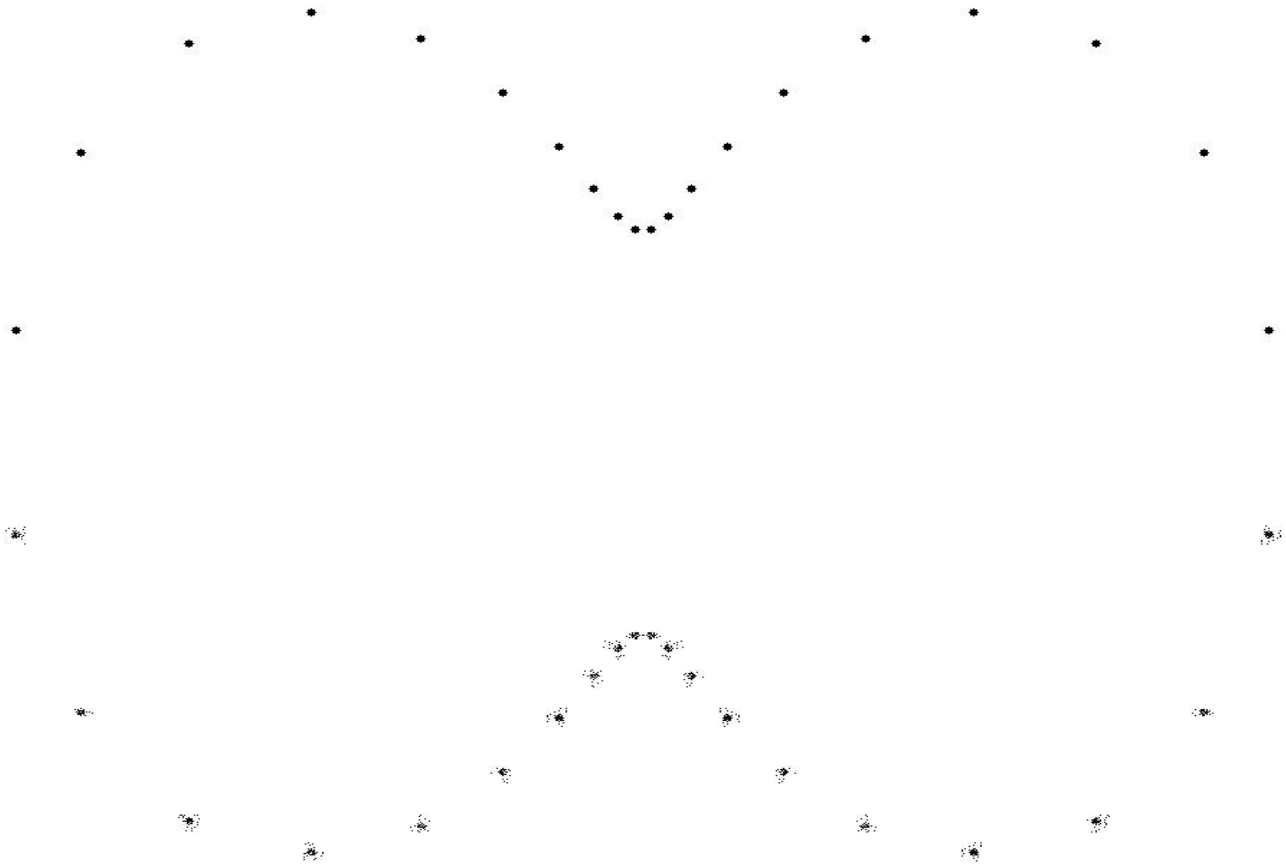
Segmentation using adaptive mean shift

1.34 hours vs 2.1 minutes

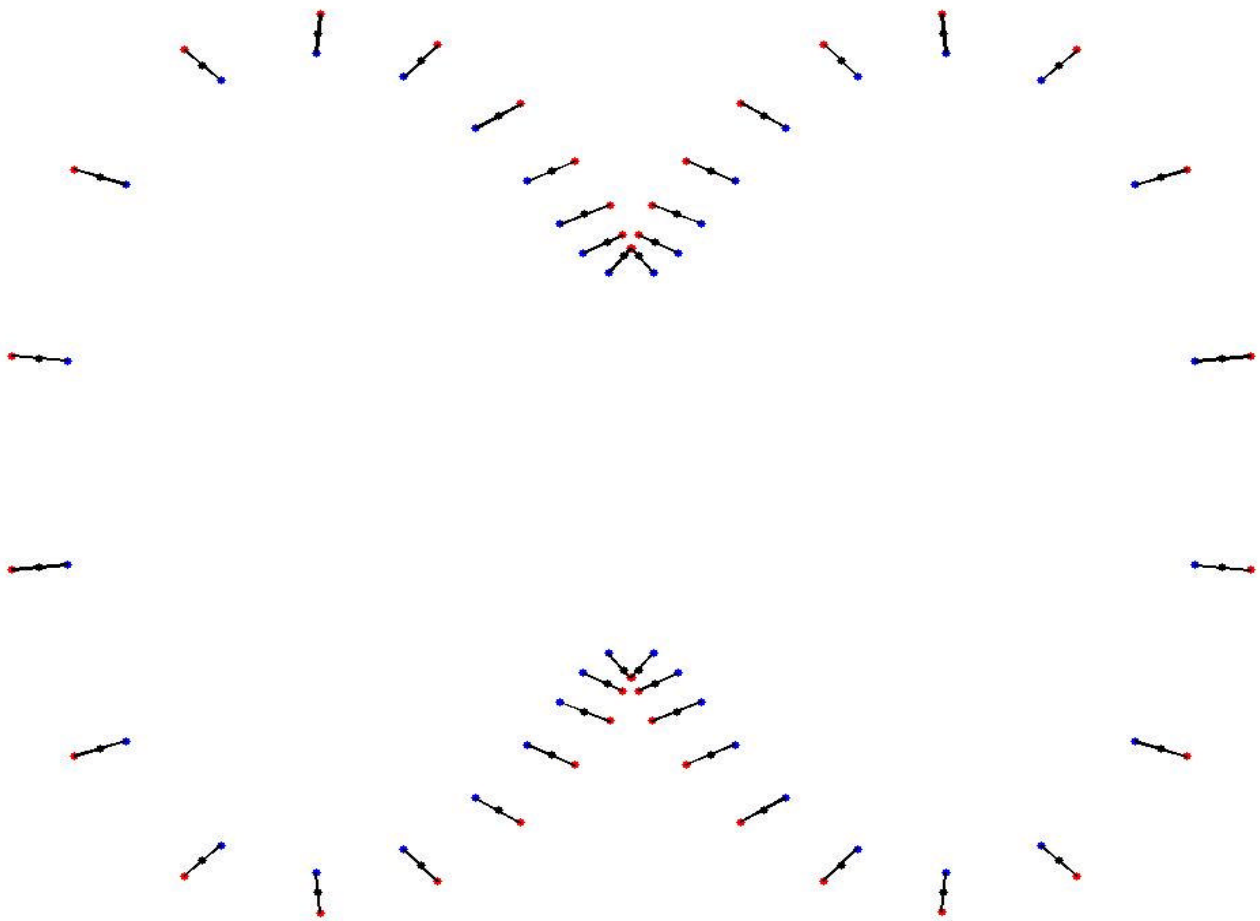
Implicit surface fitting as a regression problem

GPR_slides.pdf

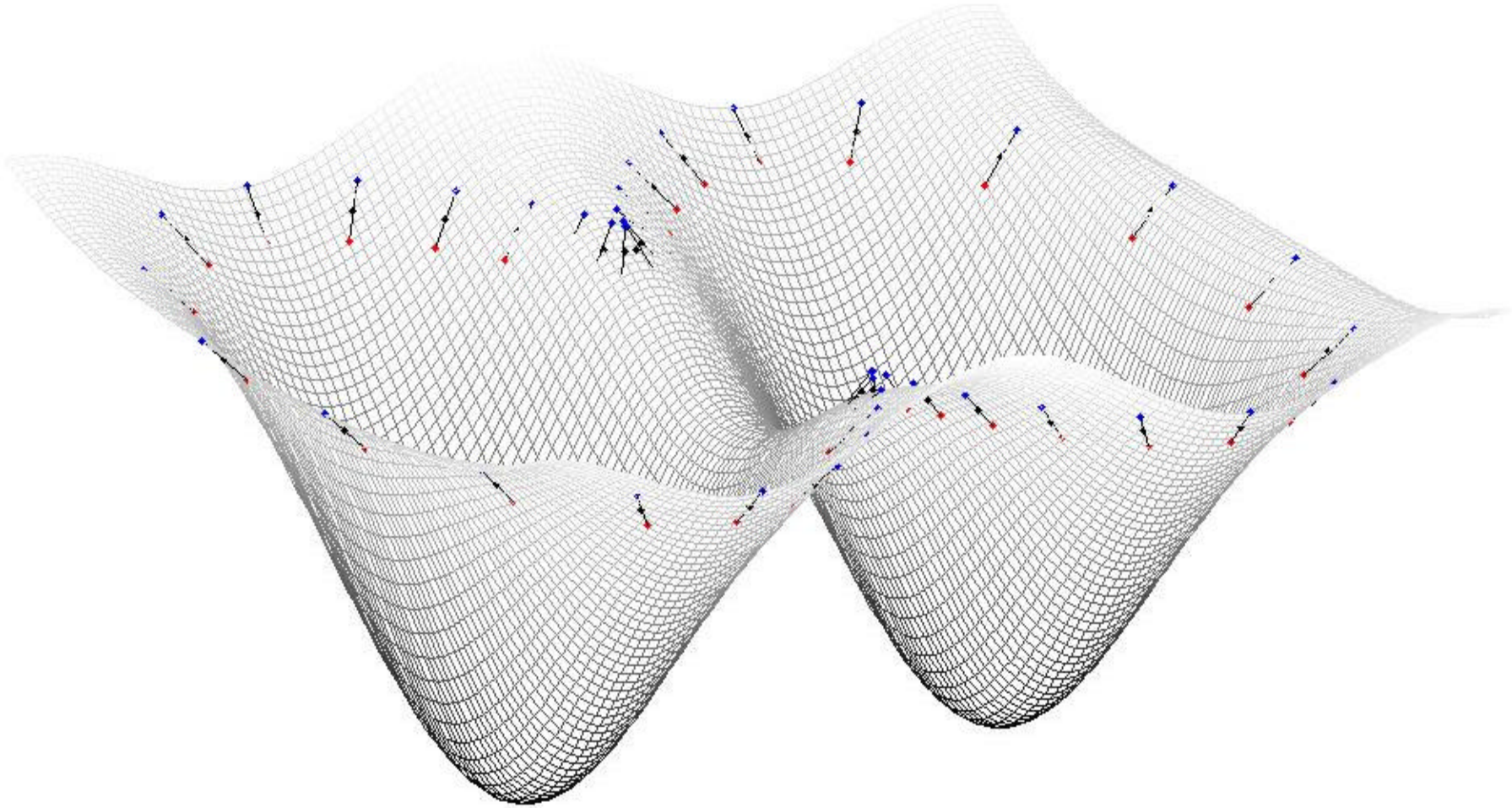
Point cloud data



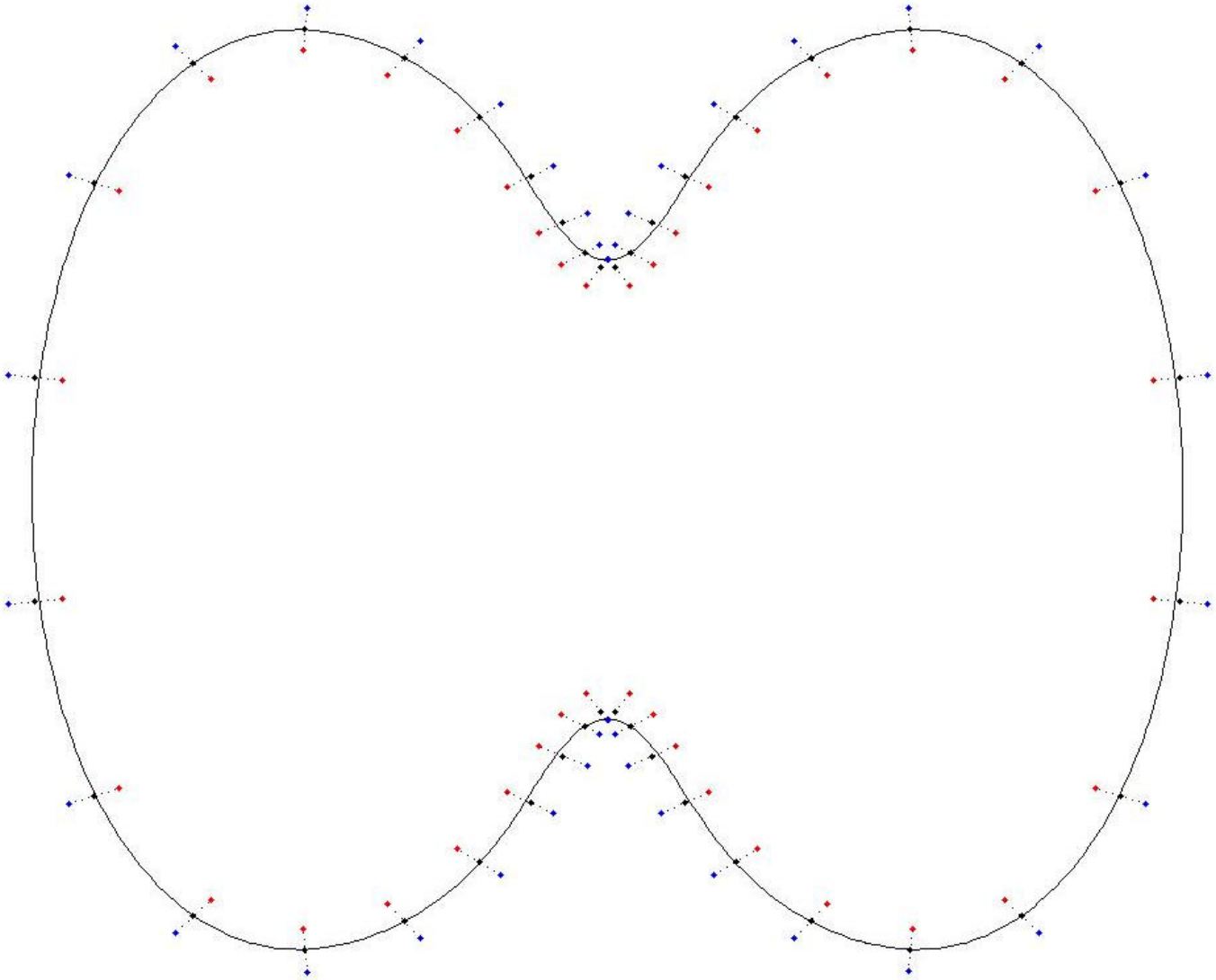
Surface normals off surface points



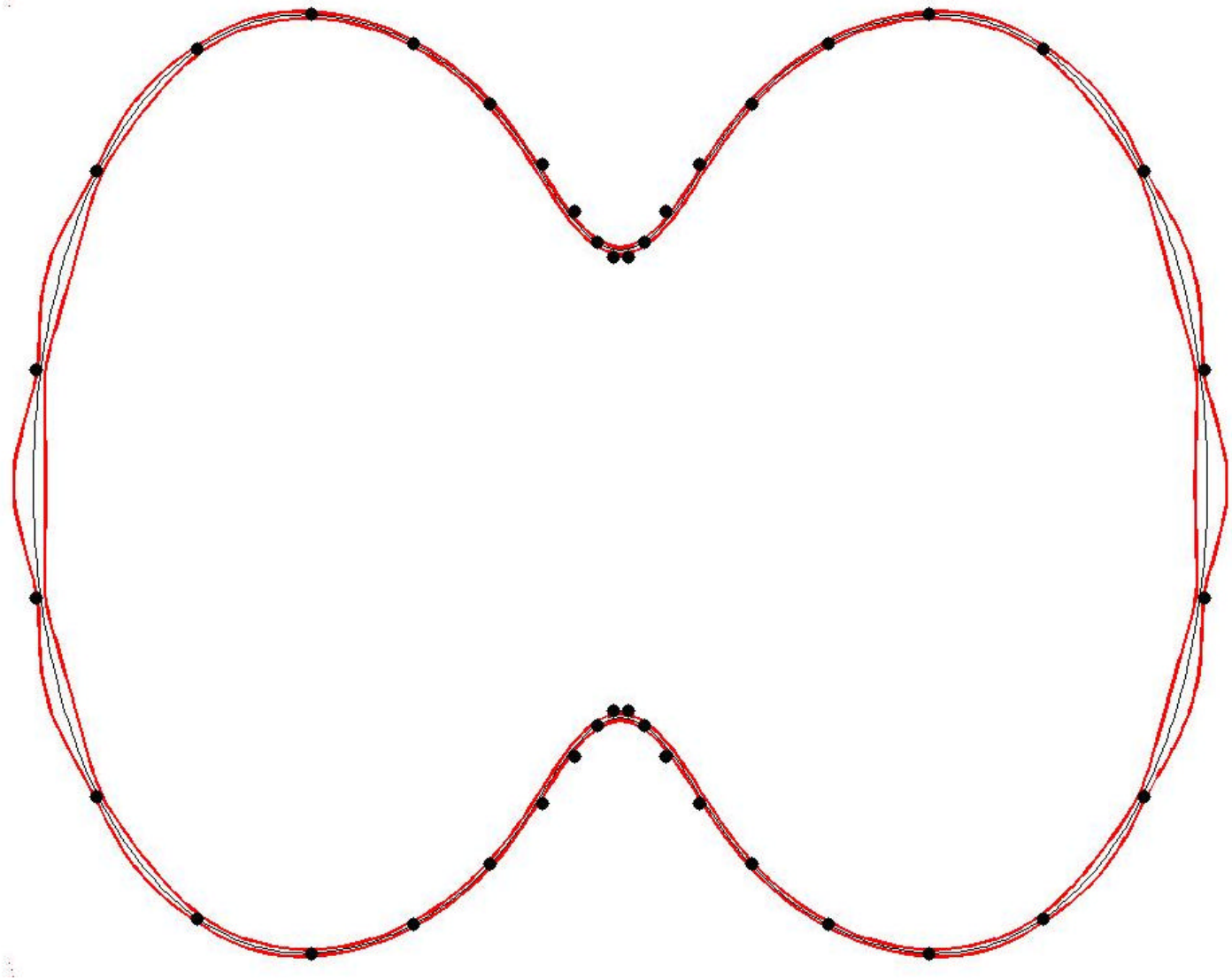
Fitted implicit surface via GPR



Zero level set of the implicit surface

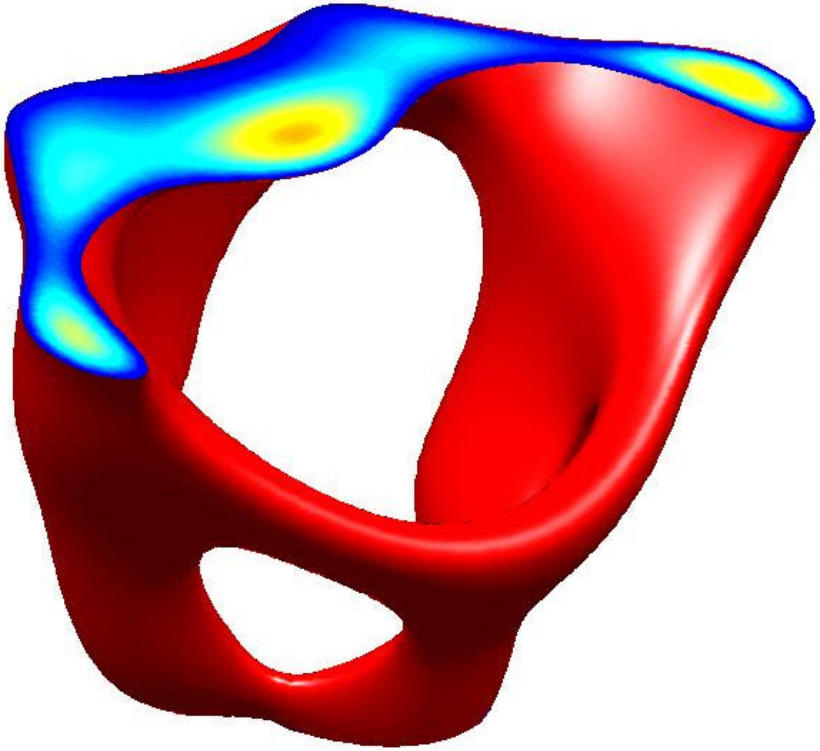


2 sigma uncertanity surfaces

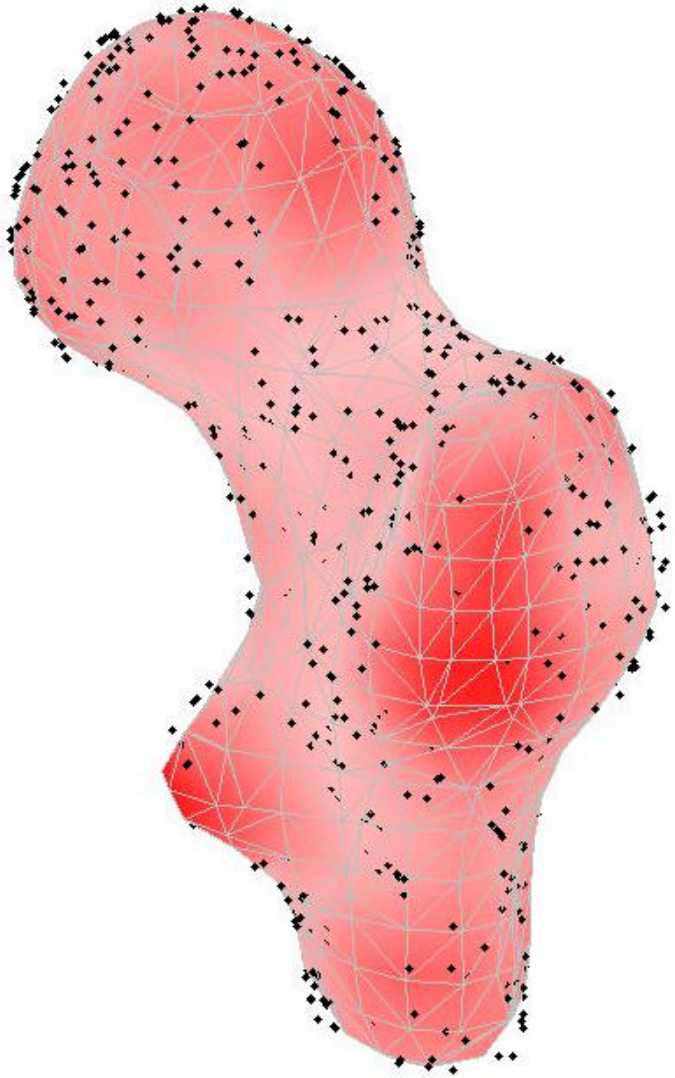


3D Point cloud data

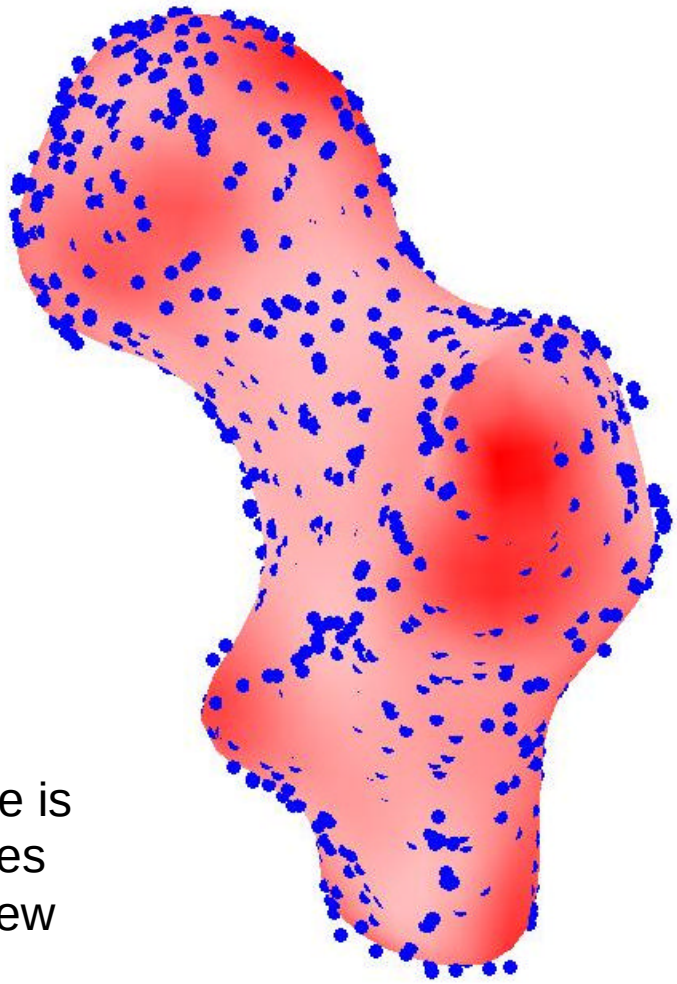
Fitted isosurface



Variance at mesh [sampling effect]



Note variance is more at places which have few datapoints



Outline of the proposal

Future work

Future work

1. Combine with dual-tree methods.

Future work

1. Combine with dual-tree methods.
2. Inexact eigenvalue methods.

Future work

1. Combine with dual-tree methods.
2. Inexact eigenvalue methods.
3. Preconditioners for the Gram matrix.

Future work

1. Combine with dual-tree methods.
2. Inexact eigenvalue methods.
3. Preconditioners for the Gram matrix.
4. Support Vector Machines in the primal.

Future work

1. Combine with dual-tree methods.
2. Inexact eigenvalue methods.
3. Preconditioners for the Gram matrix.
4. Support Vector Machines in the primal.
5. Fast large scale linear SVMs.

Future work

1. Combine with dual-tree methods.
2. Inexact eigenvalue methods.
3. Preconditioners for the Gram matrix.
4. Support Vector Machines in the primal.
5. Fast large scale linear SVMs.
6. Hyperparameter selection for different methods.

Future work

1. Combine with dual-tree methods.
2. Inexact eigenvalue methods.
3. Preconditioners for the Gram matrix.
4. Support Vector Machines in the primal.
5. Fast large scale linear SVMs.
6. Hyperparameter selection for different methods.
7. The paradox of the curse of dimensionality.

Future work

1. Combine with dual-tree methods.
2. Inexact eigenvalue methods.
3. Preconditioners for the Gram matrix.
4. Support Vector Machines in the primal.
5. Fast large scale linear SVMs.
6. Hyperparameter selection for different methods.
7. The paradox of the curse of dimensionality.
8. Structure, Inference, and Computation.

Conclusions

- Identified the key computationally intensive primitives in machine learning.

Conclusions

- Identified the key computationally intensive primitives in machine learning.
- We presented linear time algorithms.

Conclusions

- Identified the key computationally intensive primitives in machine learning.
- We presented linear time algorithms.
- We gave high accuracy guarantees.

Conclusions

- Identified the key computationally intensive primitives in machine learning.
- We presented linear time algorithms.
- We gave high accuracy guarantees.
- Unlike methods which rely on choosing a subset of the dataset we use all the available points and still achieve $\mathcal{O}(N)$ complexity.

Conclusions

- Identified the key computationally intensive primitives in machine learning.
- We presented linear time algorithms.
- We gave high accuracy guarantees.
- Unlike methods which rely on choosing a subset of the dataset we use all the available points and still achieve $\mathcal{O}(N)$ complexity.
- Applied it to a few machine learning tasks.

Conclusions

- Identified the key computationally intensive primitives in machine learning.
- We presented linear time algorithms.
- We gave high accuracy guarantees.
- Unlike methods which rely on choosing a subset of the dataset we use all the available points and still achieve $\mathcal{O}(N)$ complexity.
- Applied it to a few machine learning tasks.
- Open Issue: Handling fat data (Possibly thousands of attributes).
- Can we do better than naive?