Scalable machine learning for massive datasets: Fast summation algorithms

Getting good enough solutions as fast as possible

Vikas Chandrakant Raykar
vikas@cs.umd.edu

University of Maryland, CollegePark

March 8, 2007

Outline of the proposal

Supervised Learning

Learning from examples

- Classification [spam/ not spam]
- Regression [predicting the amount of snowfall]
- Ranking [predicting movie preferences]

Supervised Learning

Learning from examples

- Classification [spam/ not spam]
- Regression [predicting the amount of snowfall]
- Ranking [predicting movie preferences]

Learning can be viewed as function estimation $f : \mathcal{X} \to \mathcal{Y}$

- $\mathcal{X} = \mathbf{R}^d$ features/attributes.
- Classification $\mathcal{Y} = \{-1, +1\}$.
- Regression $\mathcal{Y} = \mathbf{R}$

Task	$\mathcal{X}$	$\mathcal{Y}$
spam filtering	word frequencies	spam(+1) not spam(-1)
snowfall prediction	temperature, humidity	inches of snow
movie preferences	rating by other users	movie 1 $\succeq$ movie 2

Three components of learning

Learning can be viewed as function estimation $f : \mathcal{X} \rightarrow \mathcal{Y}$.

Three tasks

- Training $\rightarrow$ Learning the function f from examples $\{x_i, y_i\}_{i=1}^{N}$.
- Prediction $\rightarrow$ Given a new x predict y.
- Model Selection $\rightarrow$ What kind on function f to use.

Two approaches to learning

Parametric approach

- Assumes a known parametric form for the function 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

- 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 prediction.
- Also known as memory based methods.

Scalable machine learning

Say we have N training examples

- Many state-of-the-art learning algorithms scale as $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$.
- They also have $\mathcal{O}(N^2)$ memory requirements.
- Huge data sets containing millions of training examples are relatively easy to gather.
- We would like to have algorithms that scale as $\mathcal{O}(N)$.

Scalable machine learning

Say we have N training examples

- Many state-of-the-art learning algorithms scale as $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$.
- They also have $\mathcal{O}(N^2)$ memory requirements.
- Huge data sets containing millions of training examples are relatively easy to gather.
- We would like to have algorithms that scale as $\mathcal{O}(N)$.

Example

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

Scalable machine learning

Say we have N training examples

- Many state-of-the-art learning algorithms scale as $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$.
- They also have $\mathcal{O}(N^2)$ memory requirements.
- Huge data sets containing millions of training examples are relatively easy to gather.
- We would like to have algorithms that scale as $\mathcal{O}(N)$.

Example

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

Previous approaches

- Use only a subset of the data.
- Online algorithms.

Goals of this dissertation

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

Goals of this dissertation

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 this dissertation

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. Demonstrate the speedup achieved on massive datasets.

Goals of this dissertation

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. Demonstrate the speedup achieved on massive datasets.
4. Realese the source code for the algorithms developed under LGPL.

Goals of this dissertation

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. Demonstrate the speedup achieved on massive datasets.
4. Realese the source code for the algorithms developed under LGPL.

Fast matrix-vector multiplication

The key computational primitive at the heart of various algorithms is a "structured" matrix-vector product (MVP).

Tools and applications

We use ideas and techniques from

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

to design these algorithms and have applied it to

Tools and applications

We use ideas and techniques from

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

to design these algorithms and have applied it to

- kernel density estimation [59,63,64,67]
- optimal bandwidth estimation [60,61]
- projection pursuit [60,61]
- implicit surface fitting
- Gaussian process regression [59,64]
- ranking [62,65,66]
- collaborative filtering [65,66]

Outline of the proposal

Key Computational tasks

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

Key Computational tasks

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

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

- Matrix-vector multiplication.
- Solving large linear systems.

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).

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).

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

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

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

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

- **Kernel machines** f is the regression/classification function. [Representer theorem]

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

- **Kernel machines** f is the regression/classification function. [Representer theorem]
- **Density estimation** f is the kernel density estimate

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

- **Kernel machines** f is the regression/classification function. [Representer theorem]
- **Density estimation** f is the kernel density estimate
- **Gaussian processes** f is the mean 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.,

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

- 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

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

  - $\mathbf{K}$ 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.

N-body problems in statistical learning

- $\mathcal{O}(N^2)$ because computations involve considering pair-wise elements.
- N-body problems in statistical learning in analogy with the Coulombic N-body problems occurring in computational physics.
- These are potential based problems involving forces or charges.
- In our case the potential corresponds to the kernel function.

Fast Matrix Vector Multiplication

We need a fast algorithm to compute

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

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 $f = \mathbf{K}q$

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

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 $\mathcal{O}(MN)$.

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?

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?

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)
Depends only on x_i and y_j.

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

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.

K is a structured matrix.

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

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 $\mathcal{O}(MN)$.

Motivating toy example

- Factorize and regroup

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

Motivating toy example

- Factorize and regroup

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

Motivating toy example

- Factorize and regroup

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

Motivating toy example

- Factorize and regroup

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

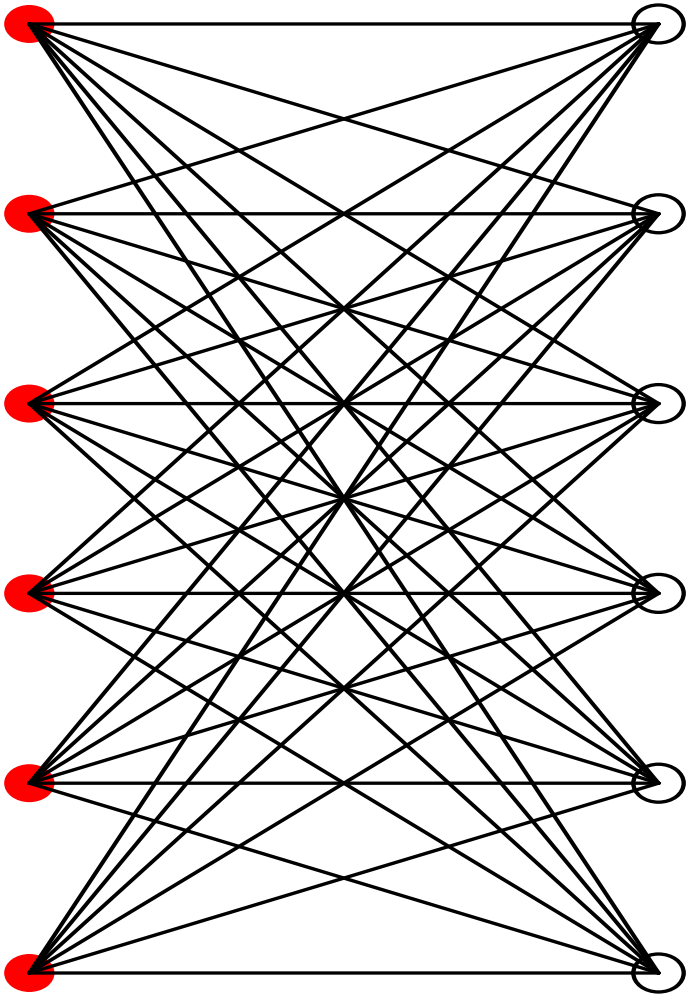
Motivating toy example

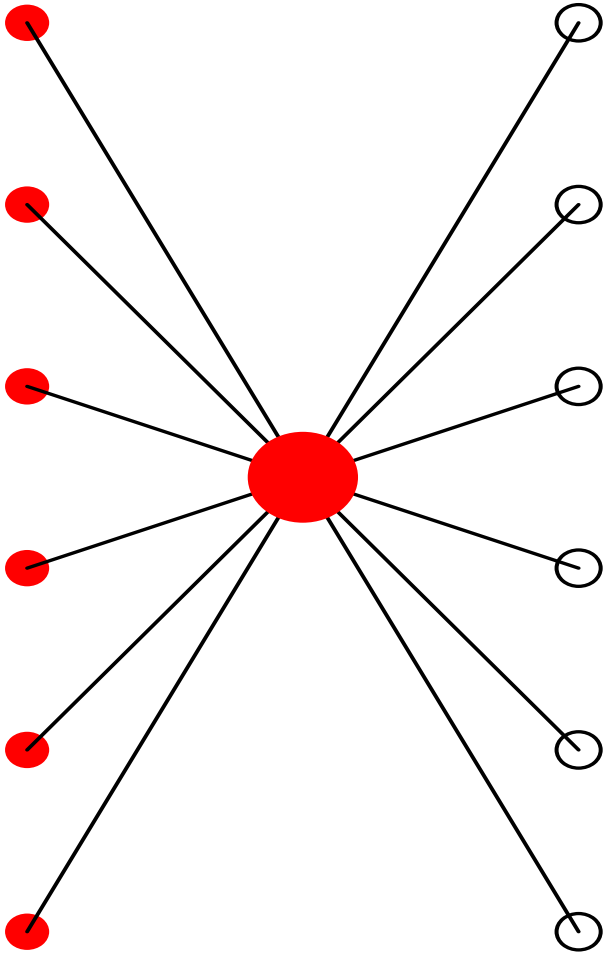
- Factorize and regroup

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

- 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.

Notion of ϵ-exact approximation

Notion of ϵ-exact approximation

- Direct computation is $\mathcal{O}(MN)$.
- We will compute $G(y_j)$ 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 $G(y_j)$ 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 $G(y_j)$ 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 $\hat{G}(y_j)$ such that $|\hat{G}(y_j) - G(y_j)| < \epsilon$.

Notion of ϵ-exact approximation

- Direct computation is $\mathcal{O}(MN)$.
- We will compute $G(y_j)$ 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 $\hat{G}(y_j)$ such that $|\hat{G}(y_j) - G(y_j)| < \epsilon$.

- 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 $G(y_j)$ 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 $\hat{G}(y_j)$ such that $|\hat{G}(y_j) - G(y_j)| < \epsilon$.

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

Two aspects of the problem

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

A class of techniques using only good space division schemes called dual tree methods have been proposed.

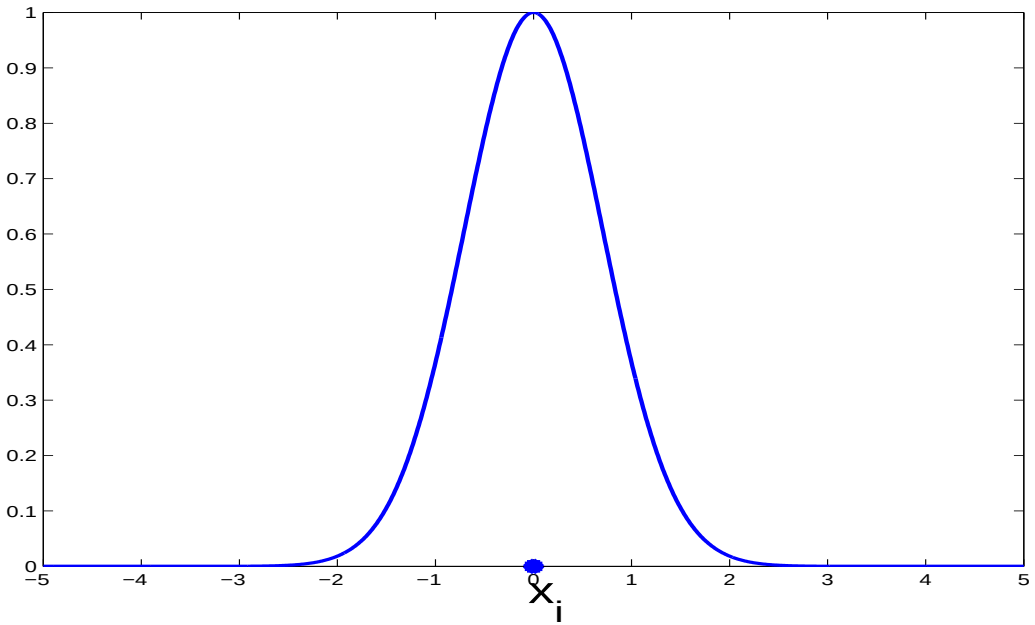
Outline of the proposal

Algorithm 1: Sums of Gaussians

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.



Discrete Gauss Transform

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

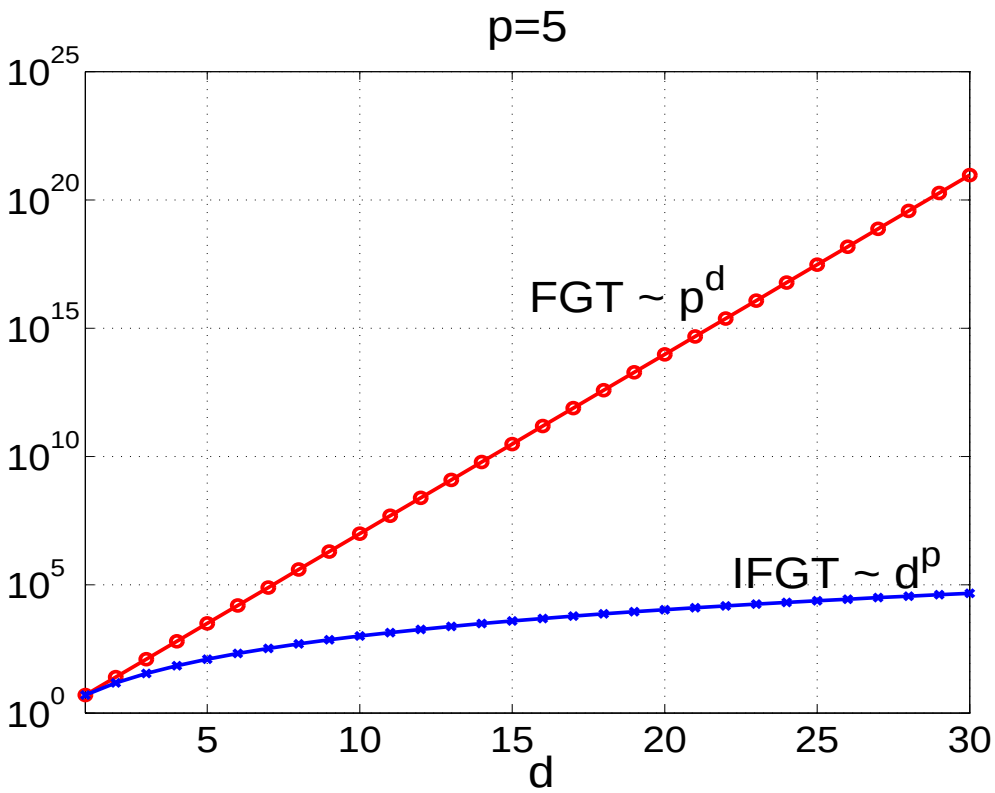
- $\{q_i \in \mathbf{R}\}_{i=1,\ldots,N}$ are the N source weights.
- $\{x_i \in \mathbf{R}^d\}_{i=1,\ldots,N}$ are the N source points.
- $\{y_j \in \mathbf{R}^d\}_{j=1,\ldots,M}$ are the M target points.
- $h \in \mathbf{R}^+$ is the source scale or bandwidth.

Fast Gauss Transform (FGT)

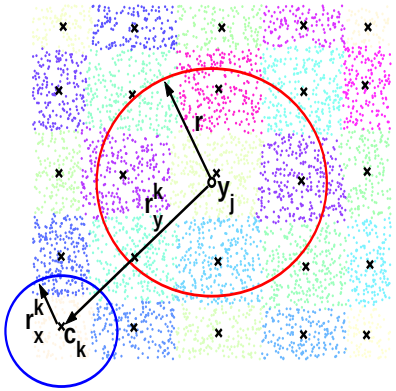
- $\epsilon -$ *exact* approximation algorithm.
- Computational complexity is $\mathcal{O}(M+N)$.
- Proposed by Greengard and Strain and applied successfully to a few lower dimensional applications in mathematics and physics.
- However the algorithm has not been widely used much in statistics, pattern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

Constants are important

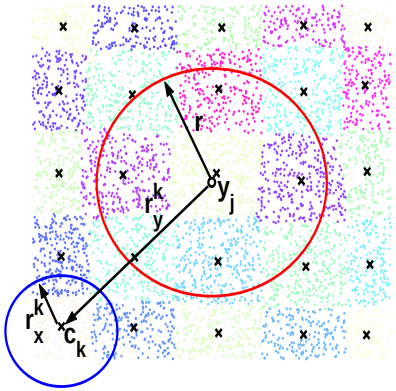
- FGT $\sim \mathcal{O}(p^d(M+N))$.
- We propose a method Improved FGT (IFGT) which scales as $\sim \mathcal{O}(d^p(M+N))$.



Brief idea of IFGT

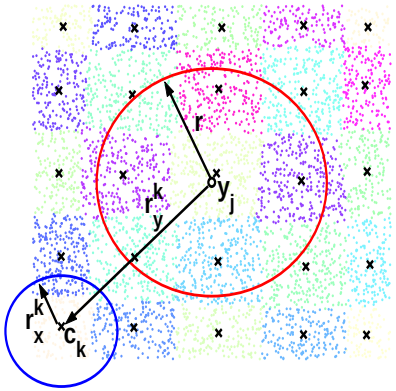


Brief idea of IFGT



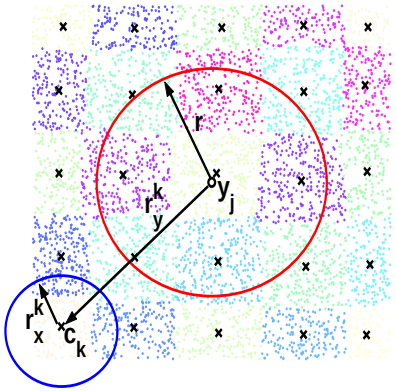
- **Step 0** Determine parameters of algorithm based on specified error bound, kernel bandwidth, and data distribution.

Brief idea of IFGT



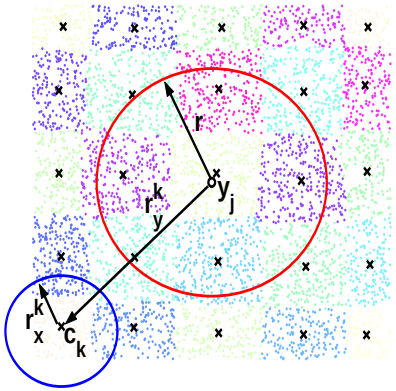
- **Step 0** Determine parameters of algorithm based on specified error bound, kernel bandwidth, and data distribution.
- **Step 1** Subdivide the d-dimensional space using a k-center clustering based geometric data structure ($\mathcal{O}(N \log K)$).

Brief idea of IFGT



- **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 ($\mathcal{O}(Nd^p)$).

Brief idea of IFGT



- **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 ($\mathcal{O}(Md^p)$).

Sample result

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

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

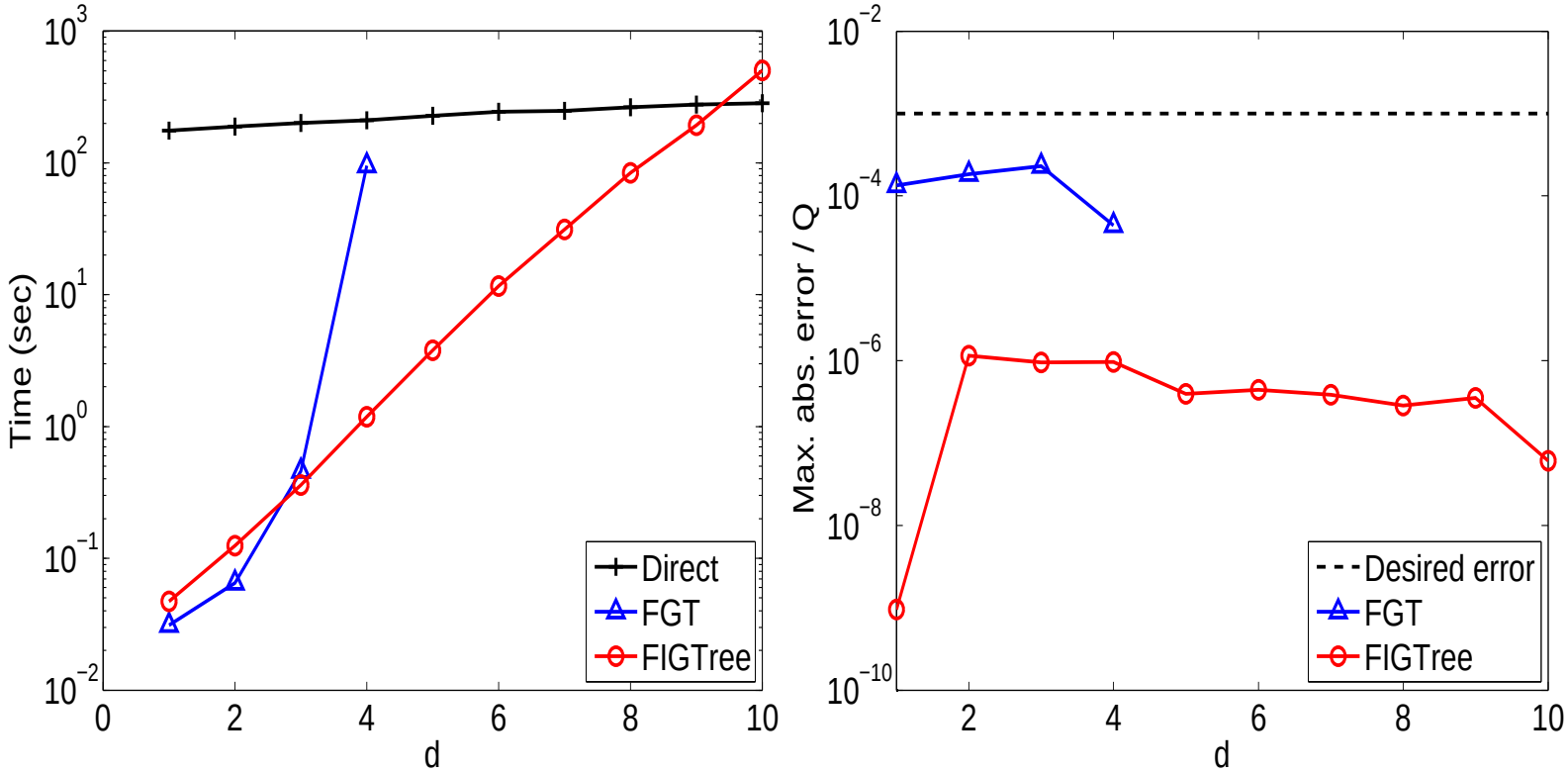
with an error of 10^{-8}.

FIGTree

We have also combined the IFGT with a *kd*-tree based nearest neighbor search algorithm.

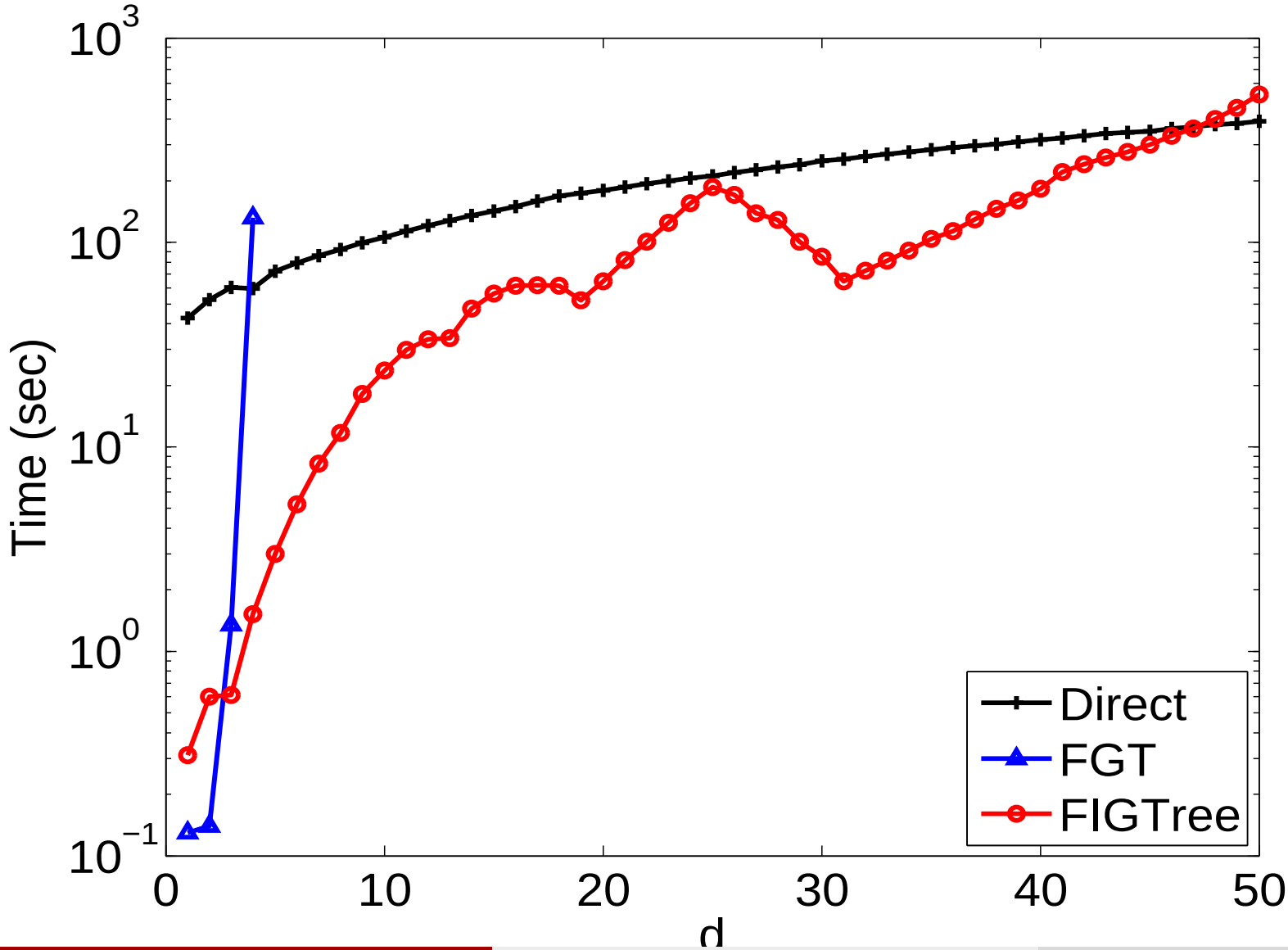
Speedup as a function of d $[h = 1.0]$

FGT cannot be run for $d > 3$



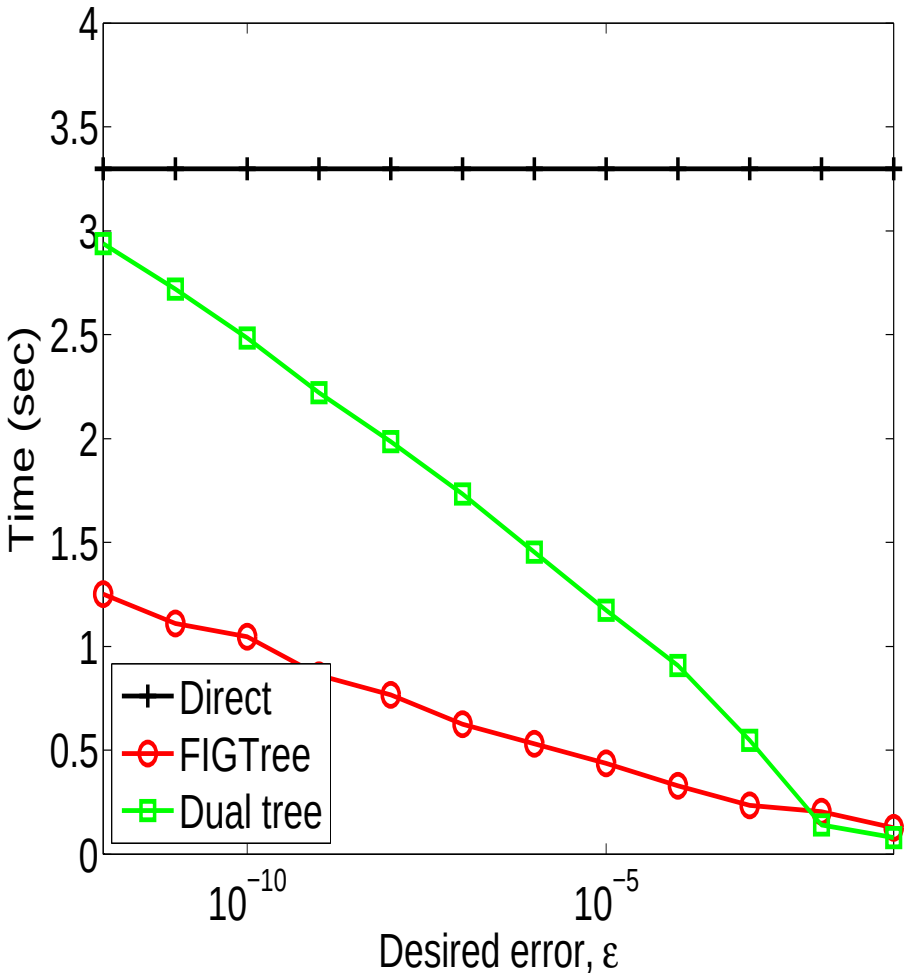
Speedup as a function of d $[h = 0.5\sqrt{d}]$

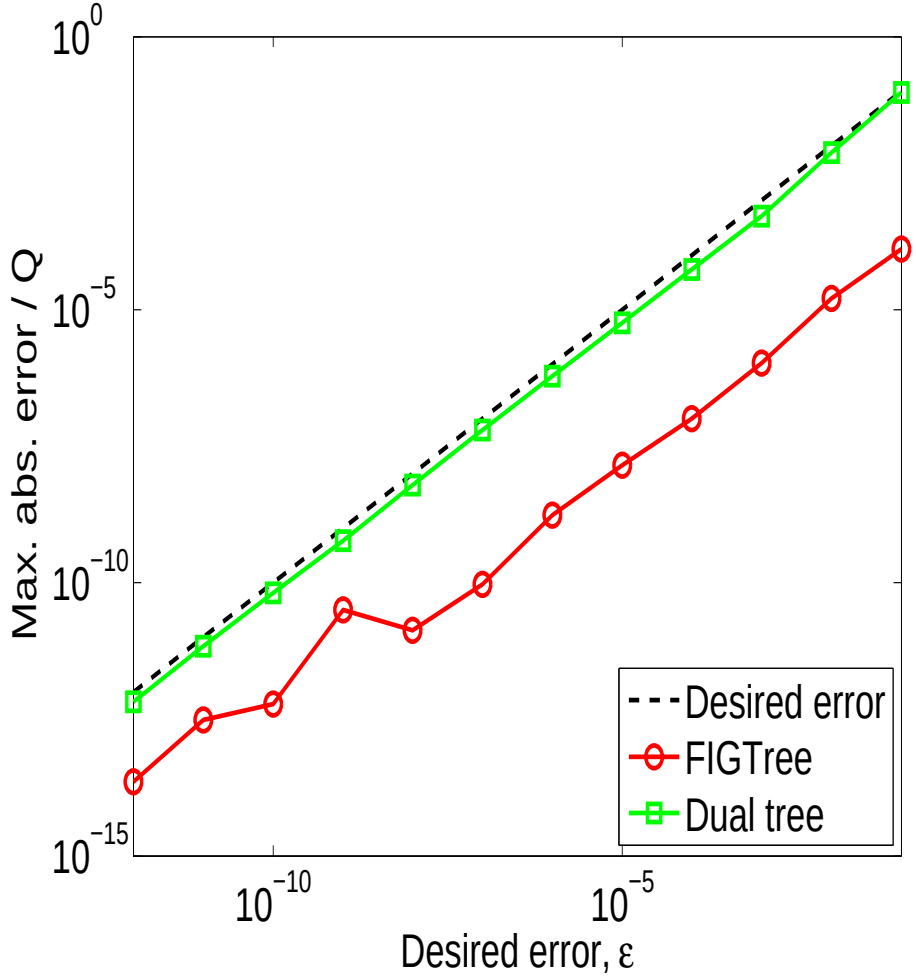
FIGTree scales well with d.



Speedup as a function of ϵ

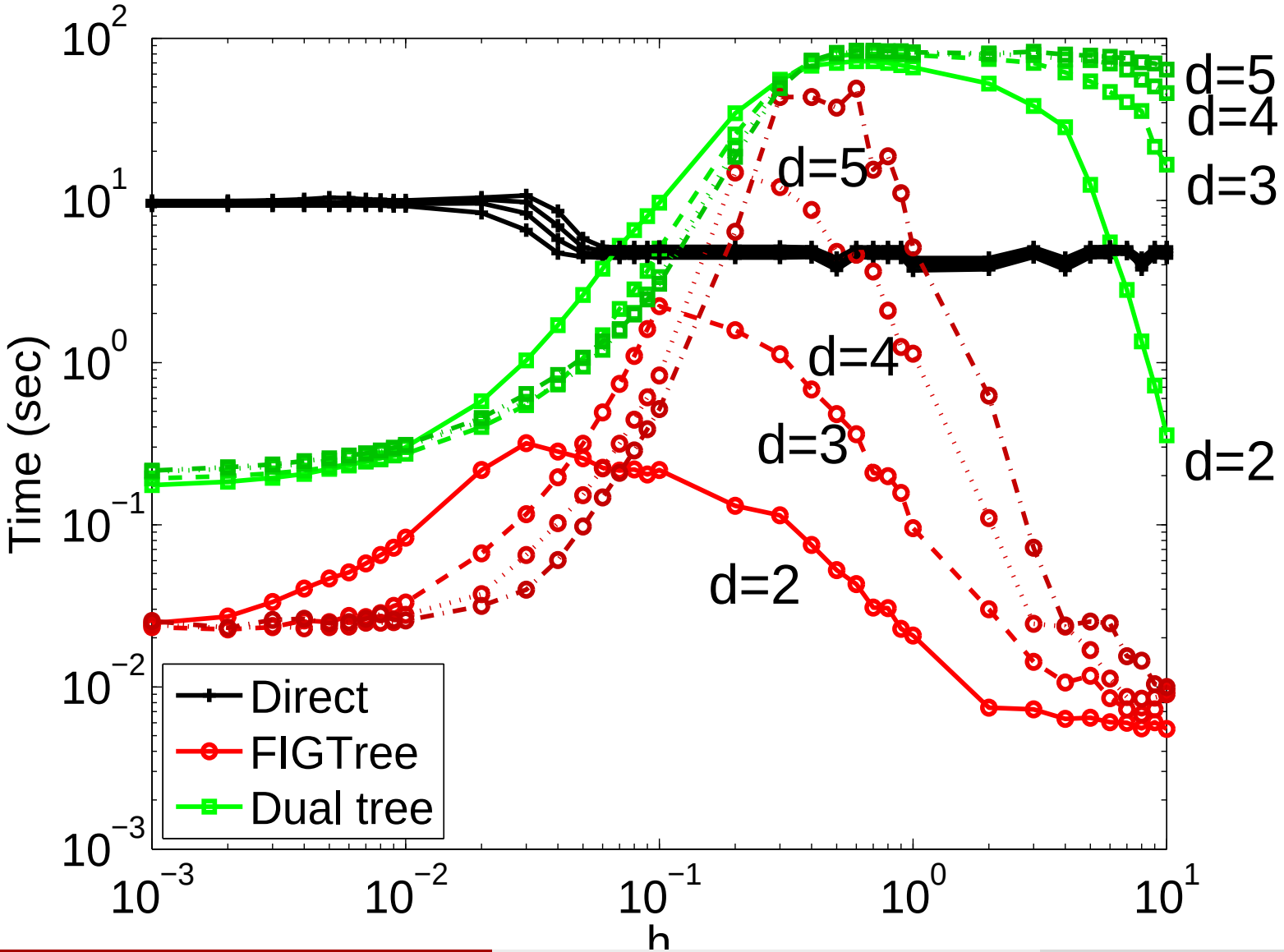
Better speedup for lower precision.





Speedup as a function of h

Scales well with bandwidth.



Applications

Direct application

- Kernel density estimation.
- Prediction in Gaussian process regression, SVM, RLS.

Applications

Direct application

- Kernel density estimation.
- Prediction in Gaussian process regression, SVM, RLS.

Embed in iterative or optimization methods

- Training of kernel machines and Gaussian processes.
- Computing eigen vector in unsupervised learning tasks.

Application 1: Kernel density Estimation

- Estimate the density $\widehat{p}$ from an i.i.d. sample $x_1, \ldots, x_N$ drawn from p.

Application 1: Kernel density Estimation

- Estimate the density $\widehat{p}$ from an i.i.d. sample $x_1, \ldots, x_N$ drawn from p.
- The most popular method is the kernel density estimator (also known as Parzen window estimator).
- $\widehat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$

Application 1: Kernel density Estimation

- Estimate the density $\widehat{p}$ from an i.i.d. sample $x_1, \ldots, x_N$ drawn from p.
- The most popular method is the kernel density estimator (also known as Parzen window estimator).
- $\widehat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$
- The widely used kernel is a Gaussian.

$$\widehat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi h^2)^{d/2}} e^{-\|x - x_i\|^2 / 2h^2}. \tag{3}$$

The computational cost of evaluating this sum at M points due to N data points is $\mathcal{O}(NM)$,

- The proposed FIGTree algorithm can be used to compute the sum approximately to ϵ precision in $\mathcal{O}(N + M)$ time.

KDE experimental results

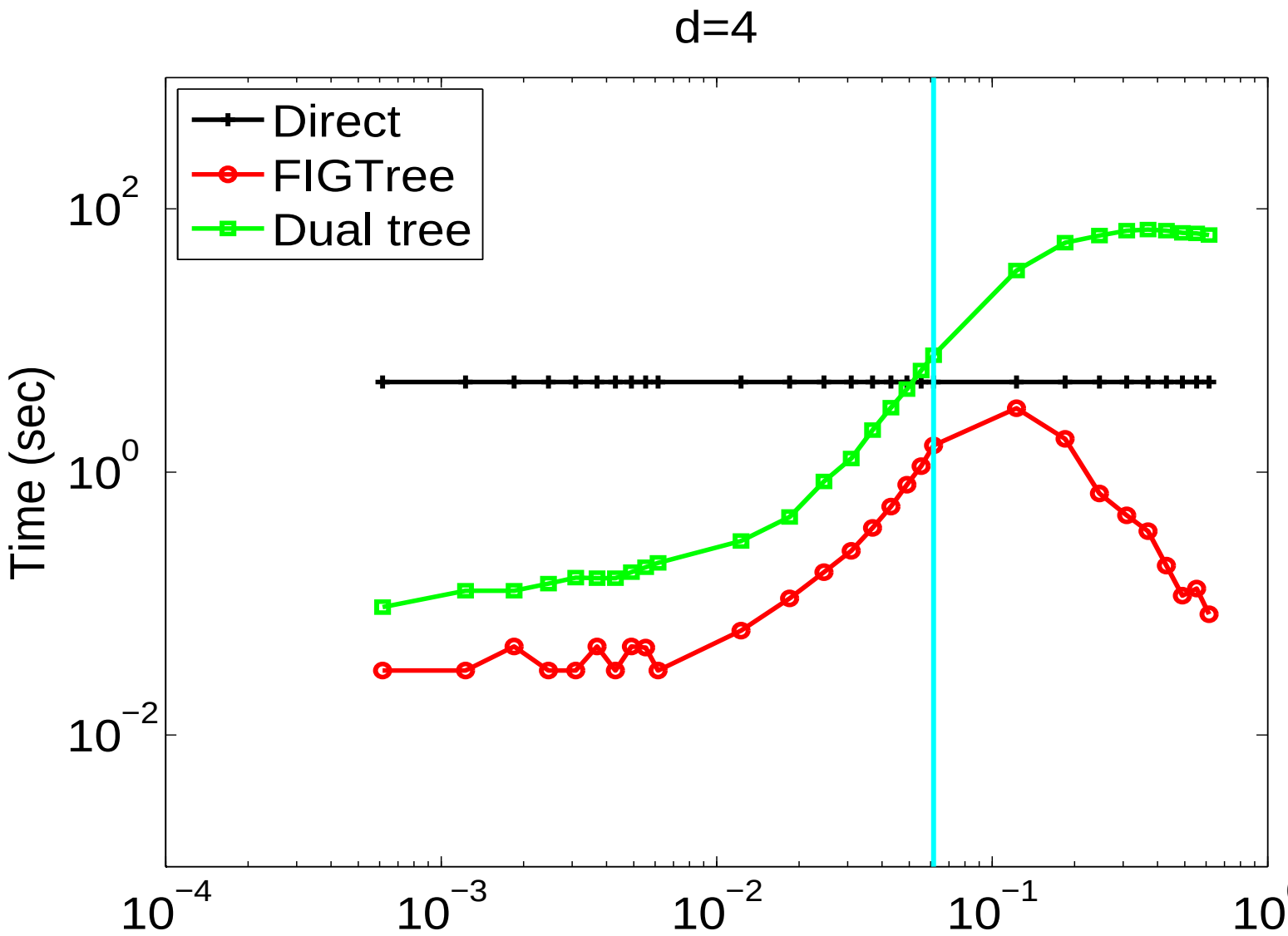
$N = M = 44,484$ $\epsilon = 10^{-2}$

SARCOS dataset

d	Optimal h	Direct time (sec.)	FIGTree time (sec.)	Speedup
1	0.024730	168.500	0.110	1531.818
2	0.033357	180.156	0.844	213.455
3	0.041688	189.438	6.094	31.0860
4	0.049527	196.375	19.047	10.310
5	0.056808	208.453	97.156	2.146
6	0.063527	221.906	130.250	1.704
7	0.069711	226.375	121.829	1.858
8	0.075400	236.781	106.203	2.230
9	0.080637	247.235	88.250	2.801
10	0.085465	254.547	98.718	2.579

KDE experimental results

$N = M = 7000$ $\epsilon = 10^{-2}$

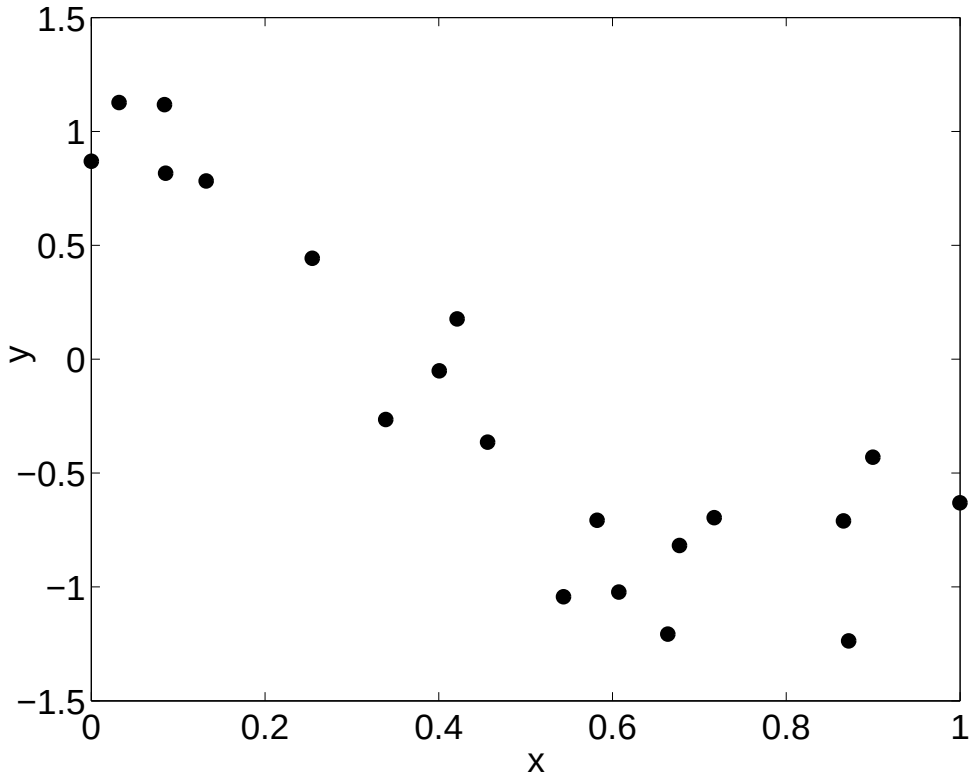


Application 2: Gaussian processes regression

Application 2: Gaussian processes regression

Regression problem

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



Gaussian processes regression

- Bayesian non-linear non-parametric regression.

Gaussian processes regression

- Bayesian non-linear non-parametric regression.
- The regression function is represented by an ensemble of functions, on which we place a Gaussian prior.

Gaussian processes regression

- Bayesian non-linear non-parametric regression.
- 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 regression

- Bayesian non-linear non-parametric regression.
- 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 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.

Infer the posterior

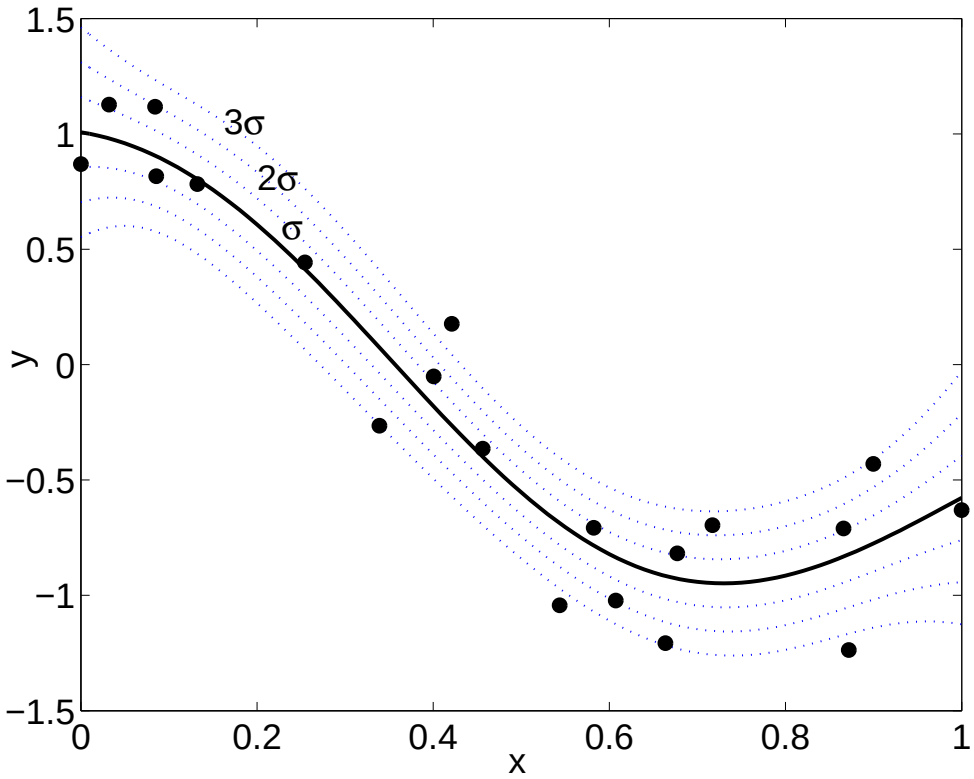
Given the training data $\mathfrak{D}$ and a new input x_* our task is to compute the posterior $p(f_*|x_*, \mathfrak{D})$.

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

- $(\mathbf{K} + \lambda \mathbf{I})\xi = \mathbf{y}$.
- The iterative method generates a sequence of approximate solutions ξ_k at each step which converge to the true solution ξ.
- Can use the conjugate-gradient method.

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.
- Can use the conjugate-gradient method.

Computational cost of conjugate-gradient

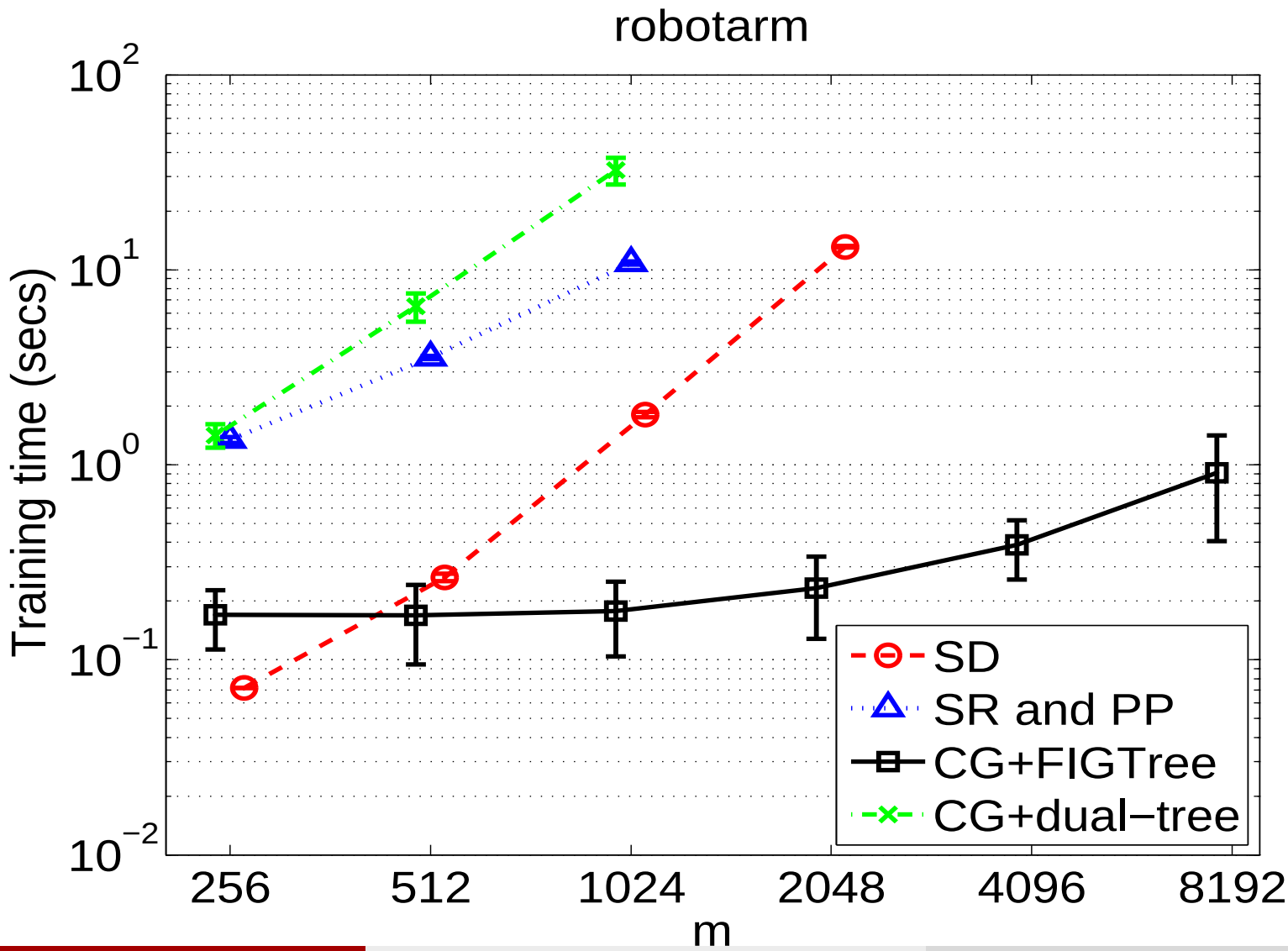
- Requires *one matrix-vector multiplication* and $5N$ flops per iteration.
- Four vectors of length N are required for storage.
- Hence computational cost now reduces to $\mathcal{O}(kN^2)$.
- For example N=25,600 takes around 17 minutes (compare to 10 hours).

CG+FIGTree

- 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]).

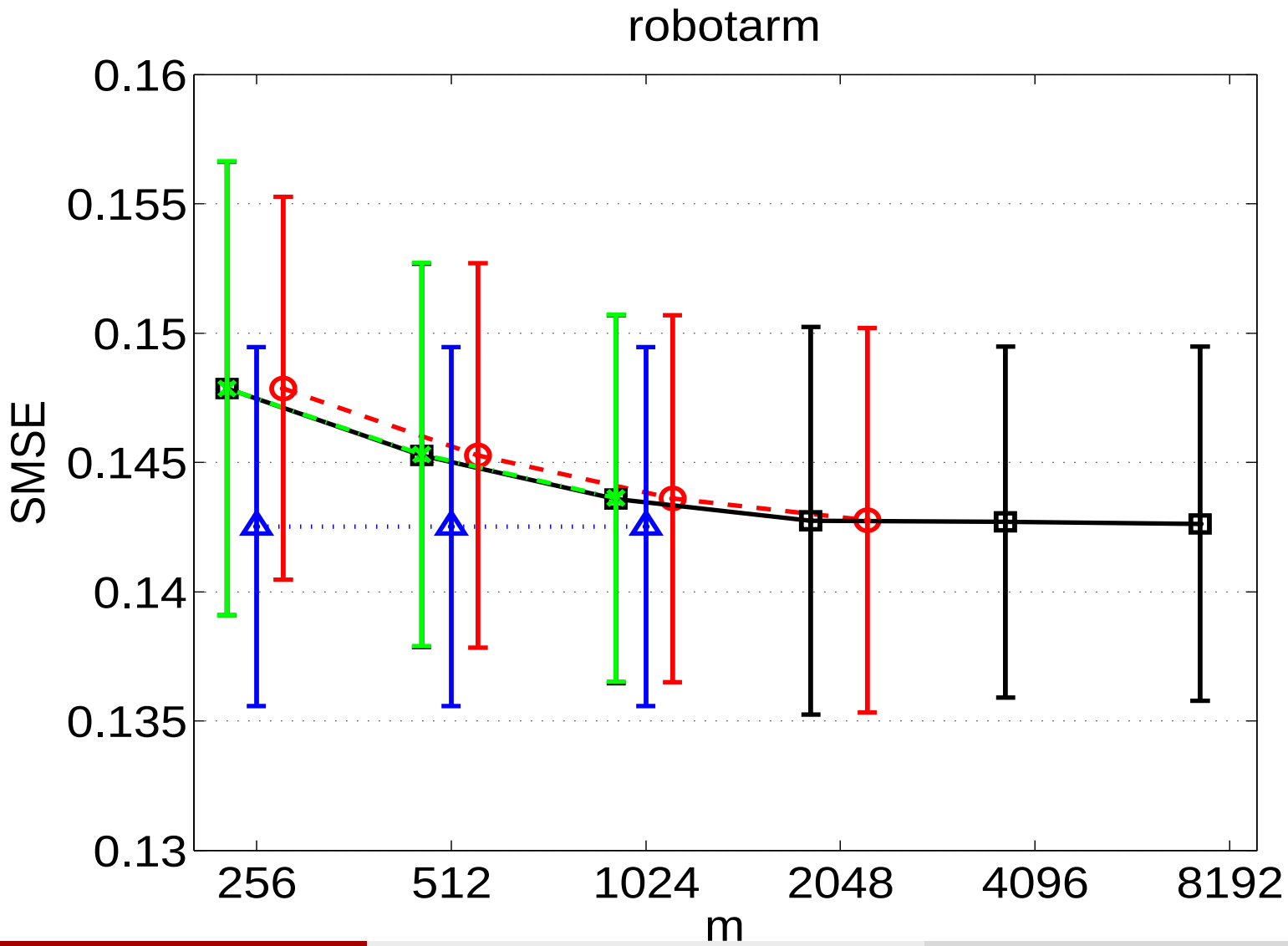
Results on the robotarm dataset

Training time



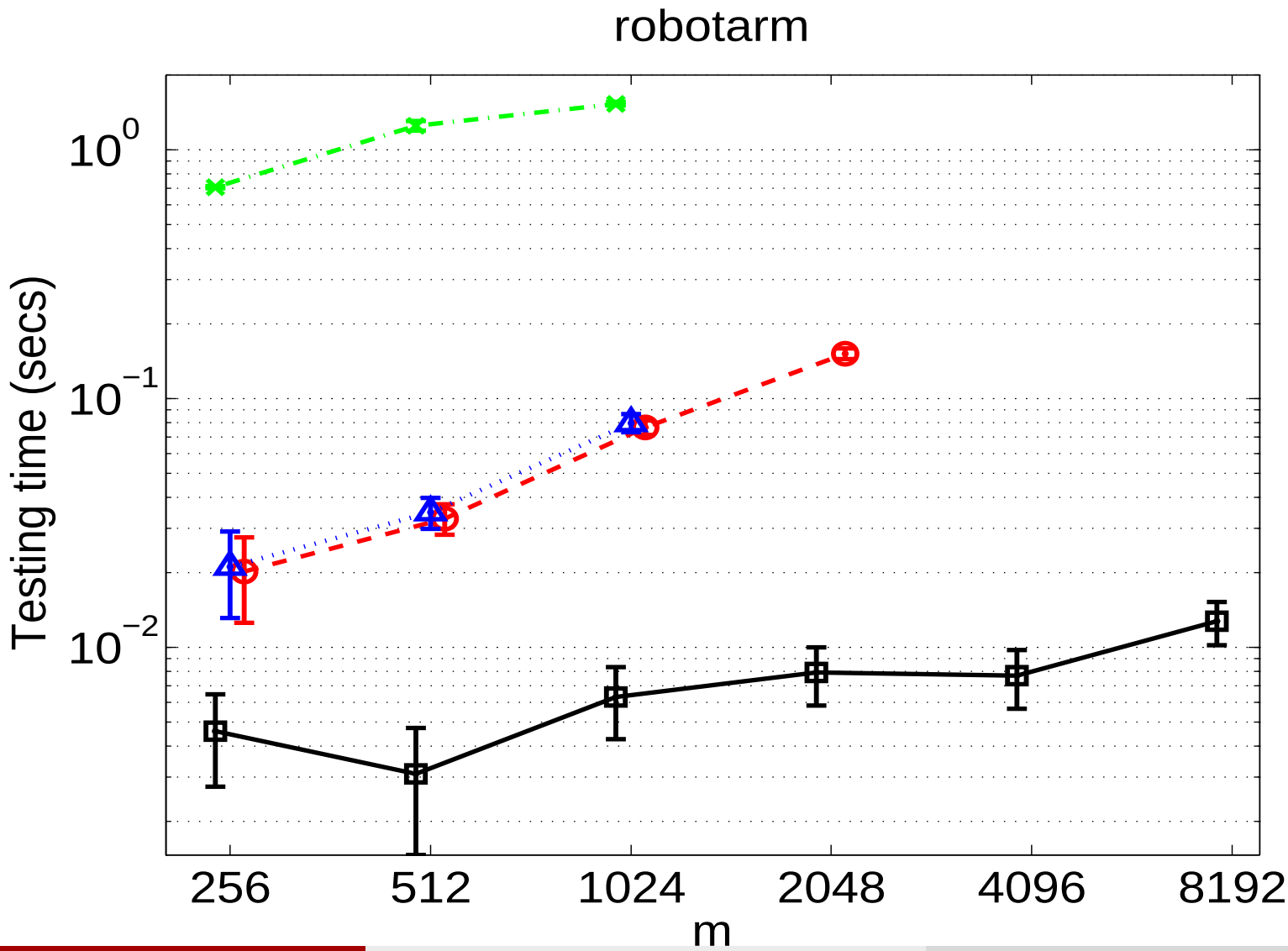
Results on the robotarm dataset

Test error



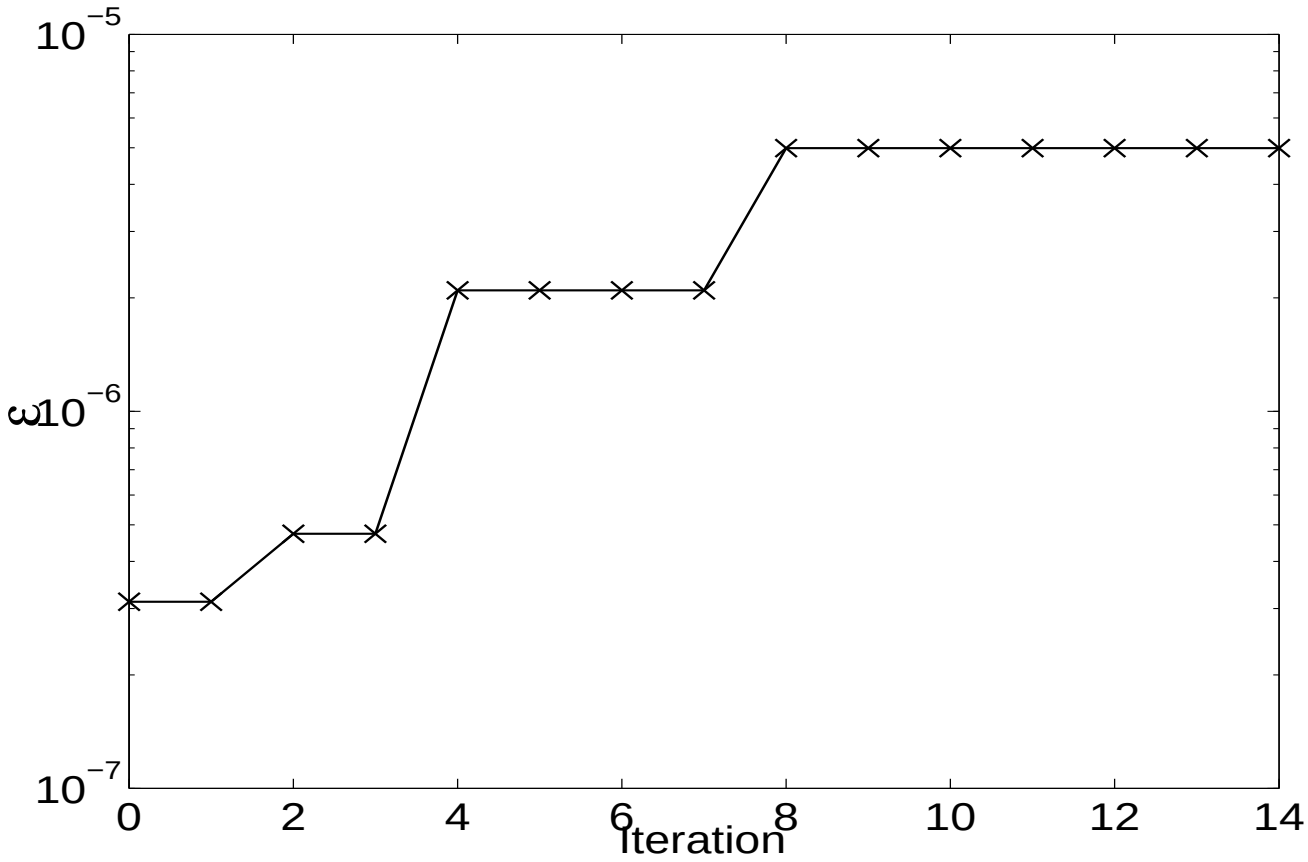
Results on the robotarm dataset

Test time

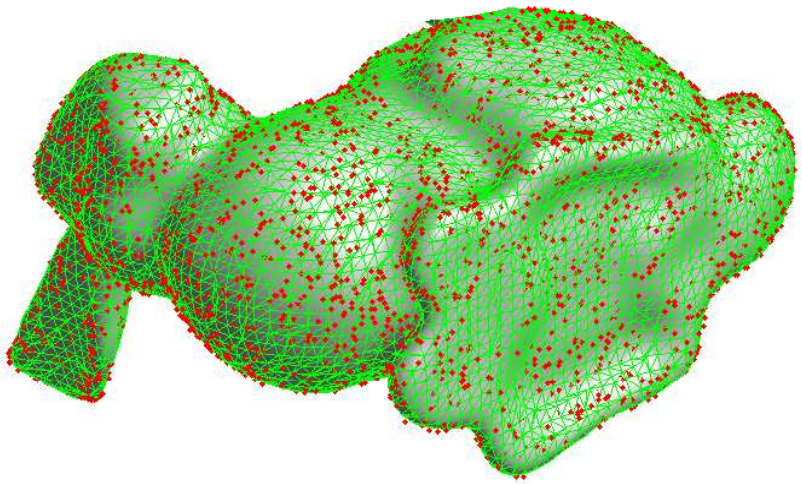


How to choose ϵ for inexact CG?

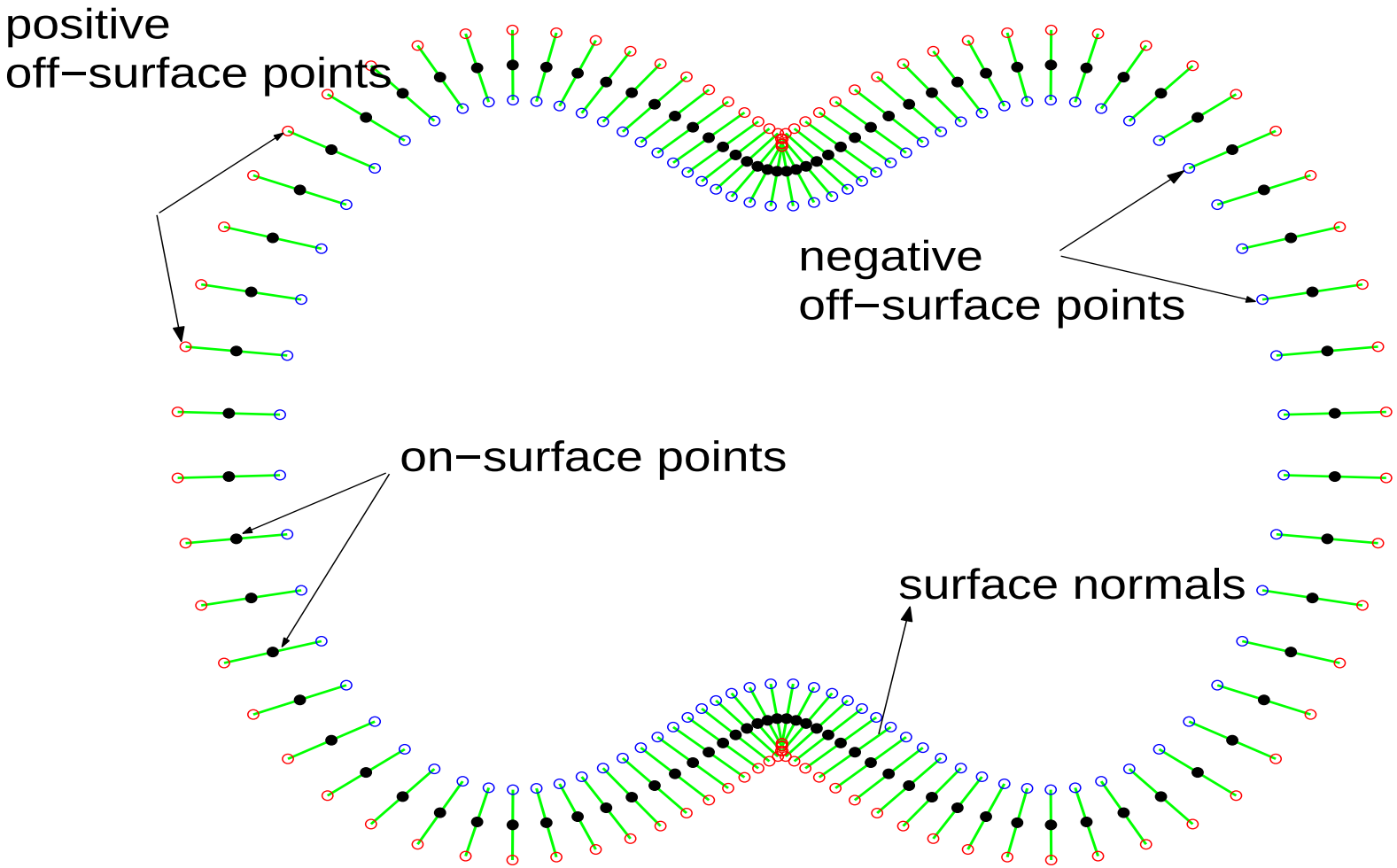
Matrix-vector product may be performed in an increasingly inexact manner as the iteration progresses and still allow convergence to the solution.



Application 3:Implicit surface fitting



Implicit surface fitting as regression



Implicit surface fitting as regression

Using the proposed approach we can handle point clouds containing millions of points.

Algorithm 2: Sums of Hermite × Gaussians

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

Algorithm 2: Sums of Hermite × Gaussians

- The FIGTree 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).

Algorithm 2: Sums of Hermite × Gaussians

- The FIGTree 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 $\mathcal{O}(N^2)$.

Algorithm 2: Sums of Hermite × Gaussians

- The FIGTree 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.

Algorithm 2: Sums of Hermite × Gaussians

- The FIGTree 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.
- $G_r(y_j) = \sum_{i=1}^{N} q_i H_r\left(\frac{y_j - x_i}{h}\right) e^{-(y_j-x_i)^2/2h^2} \quad j = 1, \ldots, M.$

Kernel density estimation

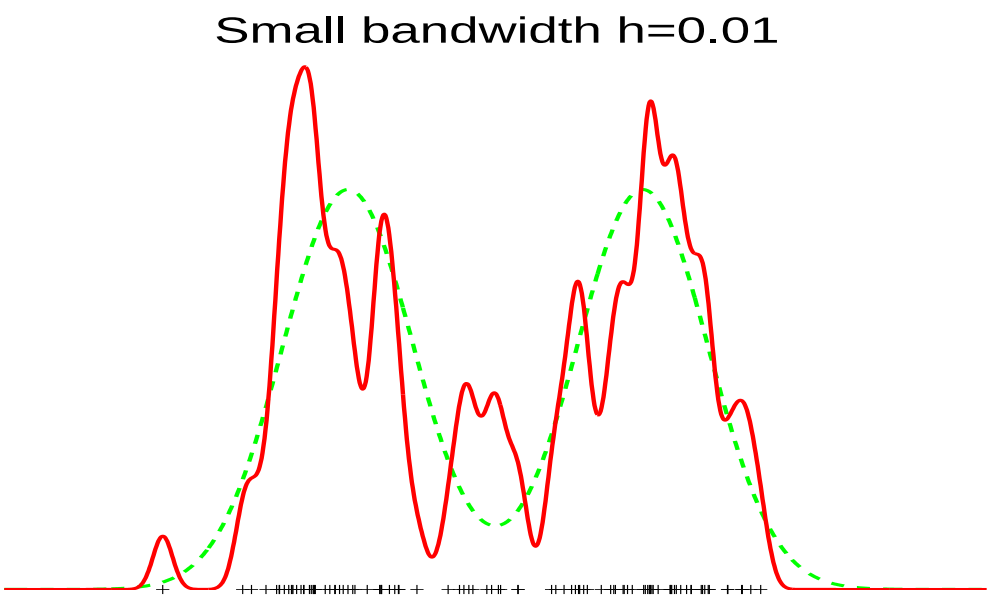
- The most popular method for density estimation is the kernel density estimator (KDE).

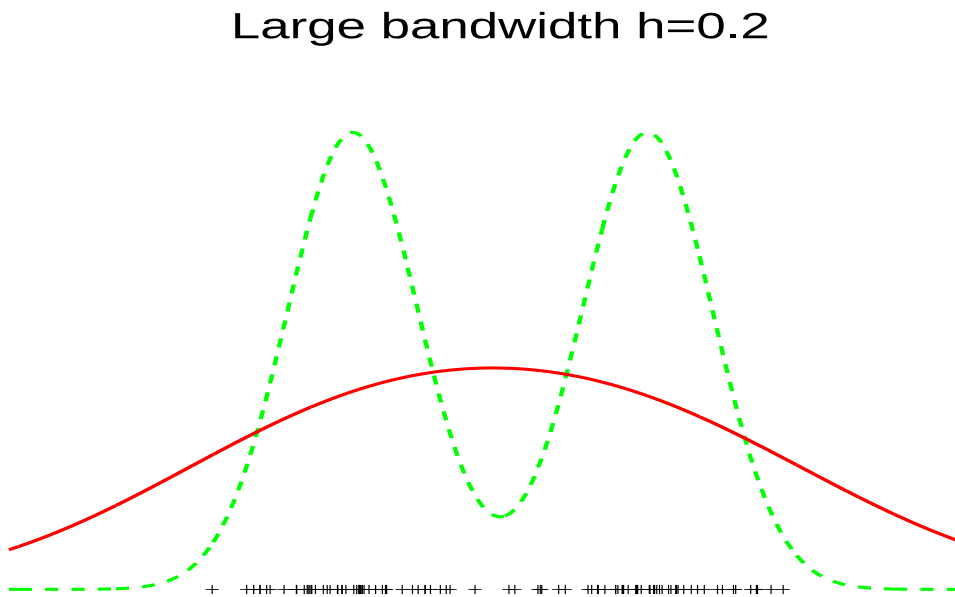
$$\widehat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

- FIGTree 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.





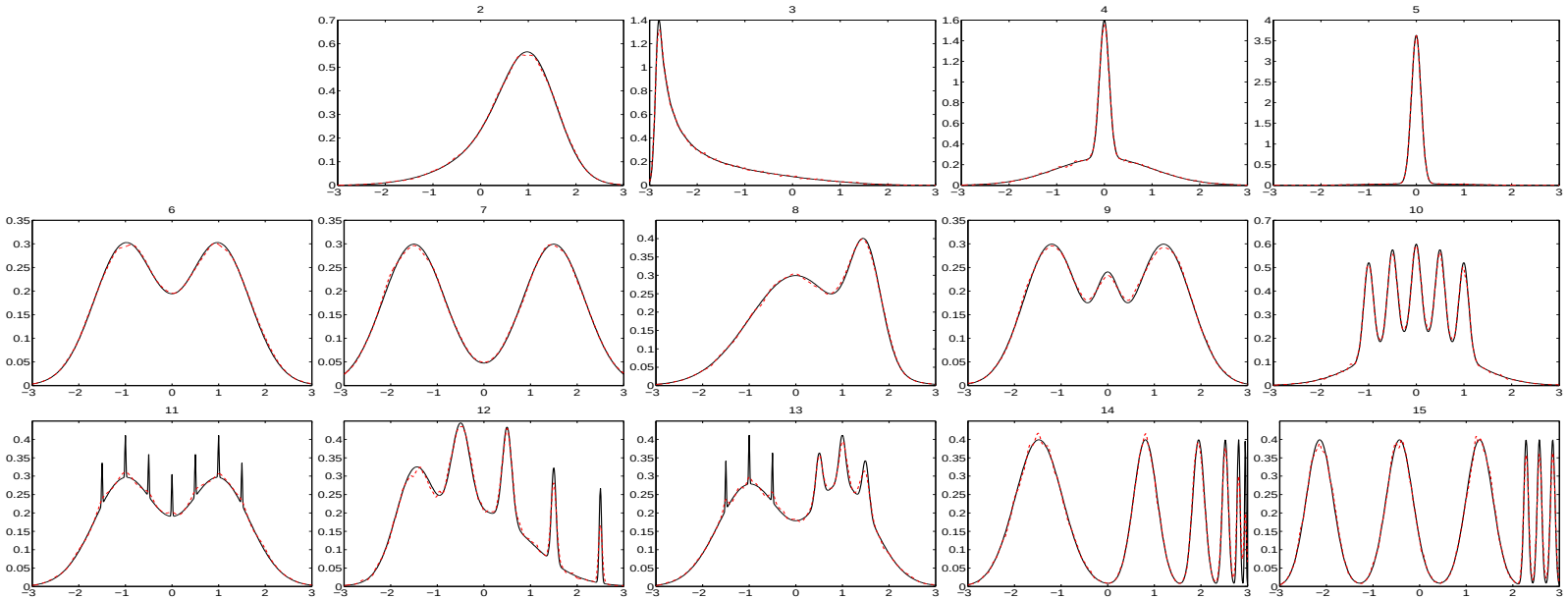
Application 1: Fast optimal bandwidth selection

- 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 $\mathcal{O}(N)$.
- The core part is the fast $\epsilon - \textit{exact}$ algorithm for kernel density derivative estimation which reduces the computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

Application 1: 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 $\mathcal{O}(N)$.
- The core part is the fast $\epsilon - \textit{exact}$ 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 $\longrightarrow$ 12.76 **hours**.
  - Fast evaluation $\longrightarrow$ 65 **seconds** with an error of around 10^{-12}.

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

Projection pursuit

The idea of projection pursuit is to search for projections from high- to low-dimensional space that are most *interesting*.

1. Given N data points in a d dimensional space project each data point onto the direction vector $a \in \mathbf{R}^d$, i.e., $z_i = a^T x_i$.
2. Compute the univariate nonparametric kernel density estimate, $\widehat{p}$, of the projected points z_i.
3. Compute the projection index $I(a)$ based on the density estimate.
4. Locally optimize over the the choice of a, to get the *most interesting* projection of the data.

Projection index

- The projection index is designed to reveal specific structure in the data, like clusters, outliers, or smooth manifolds.
- The entropy index based on Rényi's order-1 entropy is given by

$$I(a) = \int p(z) \log p(z) dz.$$

- The density of zero mean and unit variance which uniquely minimizes this is the standard normal density.
- Thus the projection index finds the direction which is most non-normal.

Speedup

The computational burden is reduced in the following three instances.

1. Computation of the kernel density estimate.
2. Estimation of the optimal bandwidth.
3. Computation of the first derivative of the kernel density estimate, which is required in the optimization procedure.

Image segmentation via PP

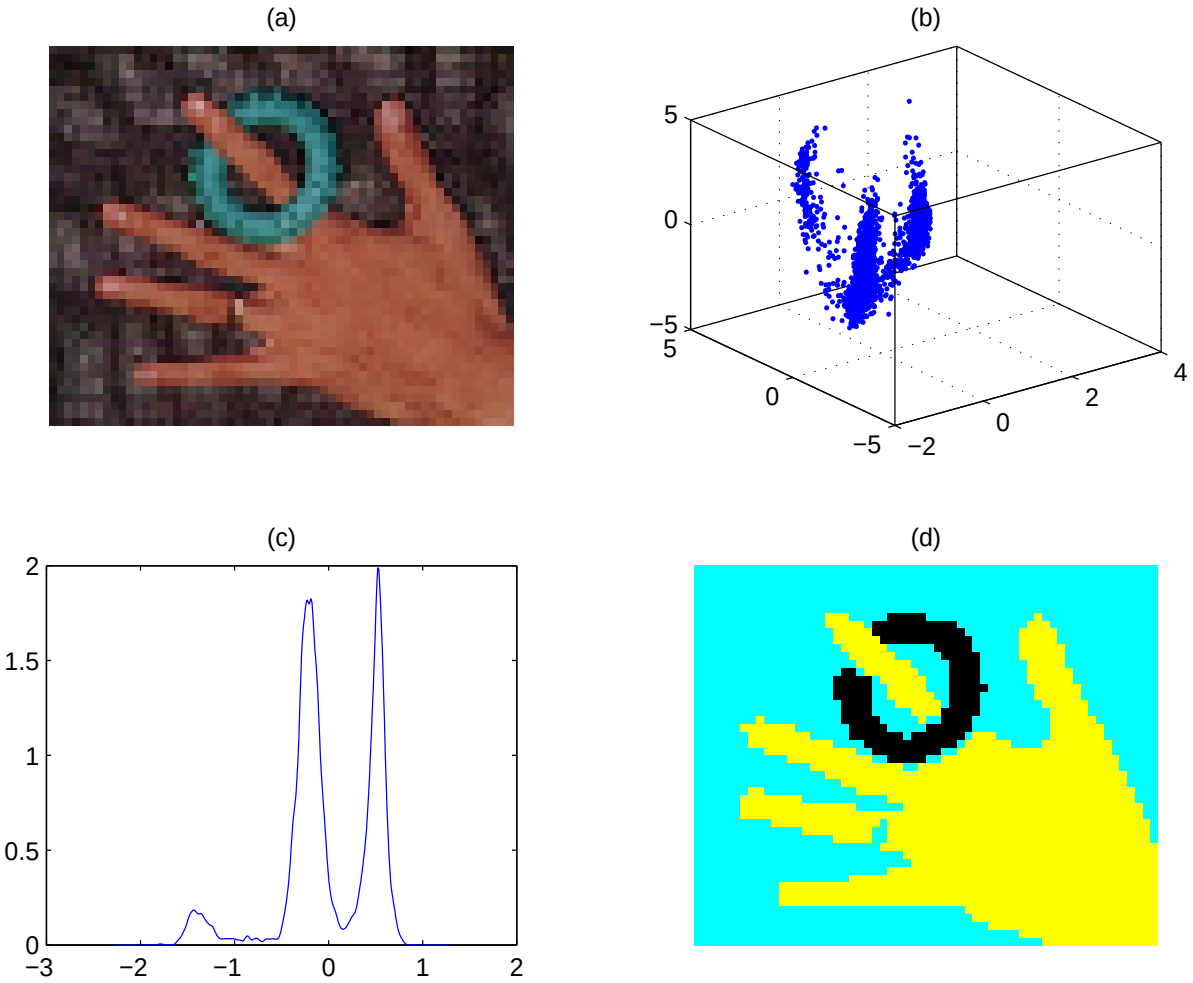
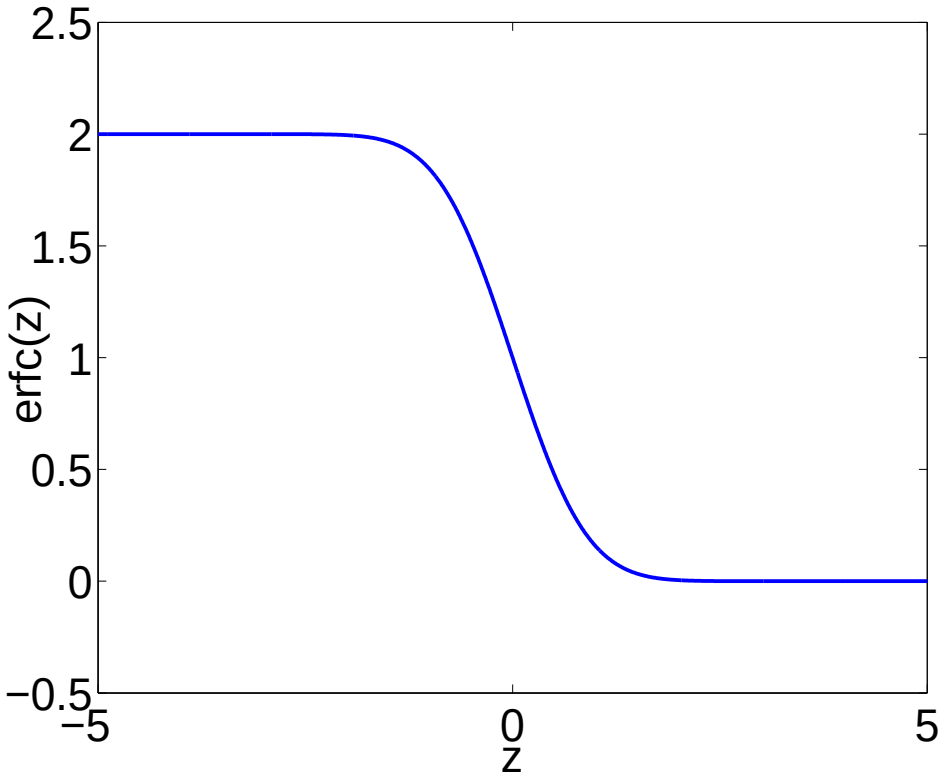


Image segmentation via PP with optimal KDE took 15 minutes while that using the direct method takes around 7.5 hours.

Algorithm 3: Sums of error functions

Another sum which we have encountered in ranking algorithms is

$$E(y) = \sum_{i=1}^{N} q_i \operatorname{erfc}(y - x_i).$$



Example

- $N = M = 51,200$.
- Direct evaluation takes about 18 hours.
- We specify $\epsilon = 10^{-6}$.
- Fast evaluation just takes 5 seconds.
- Actual error is around 10^{-10}

Application 1: Ranking

- For some applications ranking or ordering the elements is more important.
  - Information retrieval.
  - Movie recommendation.
  - Medical decision making.
- Compare two instances and predict which one is better.
- Various ranking algorithms train the models using pairwise preference relations.
- Computationally expensive to train due to the quadratic scaling in the number of pairwise constraints,

Fast ranking algorithm

- We propose a new ranking algorithm.
- Our algorithm also uses pairwise comparisons the runtime is still linear.
- This is made possible by fast approximate summation of erfc functions.
- The proposed algorithm is as accurate as the best available methods in terms of ranking accuracy.
- Several orders of magnitude faster.
- For a dataset with $4,177$ examples the algorithm took around 2 seconds.
- Direct took 1736 seconds and the best competitor RankBoost took 63 seconds.

Outline of the proposal

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.

Publications

Conference papers

- A fast algorithm for learning large scale preference relations. Vikas C. Raykar, Ramani Duraiswami, and Balaji Krishnapuram, In Proceedings of the AISTATS 2007, Peurto Rico, March 2007 [also submitted to PAMI]
- 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. [in preparation for JCGS]
- The Improved Fast Gauss Transform with applications to machine learning. Vikas C. Raykar and Ramani Duraiswami, To appear in Large Scale Kernel Machines, MIT Press 2006. [also submitted to JMLR]

Technical reports

- Fast weighted summation of erfc functions. Vikas C. Raykar, R. Duraiswami, and B. Krishnapuram, CS-TR-4848, Department of computer science, University of Maryland, CollegePark.
- Very fast optimal bandwidth selection for univariate kernel density estimation. Vikas C. Raykar and R. Duraiswami, CS-TR-4774, Department of computer science, University of Maryland, CollegePark.
- Fast computation of sums of Gaussians in high dimensions. Vikas C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov, CS-TR-4767, Department of computer science, University of Maryland, CollegePark.

Software releases under LGPL

- The FIGTree algorithm.
- Fast optimal bandwidth estimation.
- Fast erfc summation (coming soon)