A Fast Algorithm for Learning a Ranking Function from Large-Scale Data Sets

Vikas C. Raykar, Ramani Duraiswami, *Member*, *IEEE*, and
Balaji Krishnapuram, *Member*, *IEEE*

Abstract—We consider the problem of learning a ranking function that maximizes a generalization of the Wilcoxon-Mann-Whitney statistic on the training data. Relying on an ϵ-accurate approximation for the error function, we reduce the computational complexity of each iteration of a conjugate gradient algorithm for learning ranking functions from $\mathcal{O}(m^2)$ to $\mathcal{O}(m)$, where m is the number of training samples. Experiments on public benchmarks for ordinal regression and collaborative filtering indicate that the proposed algorithm is as accurate as the best available methods in terms of ranking accuracy, when the algorithms are trained on the same data. However, since it is several orders of magnitude faster than the current state-of-the-art approaches, it is able to leverage much larger training data sets.

Index Terms—Ranking, preference relations, fast erfc summation.

1 Introduction

The problem of *ranking* has recently received significant attention in the statistical machine learning and information retrieval communities. In a typical ranking formulation, we compare two instances and determine which one is *better* or *preferred*. Based on this, a set of instances can be ranked according to a desired *preference relation*. The study of ranking has largely been motivated by applications in search engines, information retrieval, collaborative filtering, and recommender systems. For example, in search engines, rather than returning a document as relevant or not (classification), the ranking formulation allows one to sort the documents in the order of their relevance.

1.1 Preference Relation and Ranking Function

Consider an instance space $\mathcal{X}$. For any $(x, y) \in \mathcal{X} \times \mathcal{X}$, we interpret the *preference relation* $x \succeq y$, as "x *is at least as good as* y." We say that "x *is indifferent to* y" $(x \sim y)$ if $x \succeq y$ and $y \succeq x$. For *learning a ranking*, we are provided with a set of pairwise preferences based on which we have to learn a preference relation. In general, an ordered list of instances can always be decomposed down to a set of pairwise preferences. One way of describing preference relations is by means of a ranking function. A function $f : \mathcal{X} \rightarrow \mathbb{R}$ is a *ranking/scoring function* representing the preference relation $\succeq$ if

$$\forall x, y \in \mathcal{X}, \quad x \succeq y \Leftrightarrow f(x) \geq f(y). \tag{1}$$

The ranking function f provides a numerical score to the instances based on which the instances can be ordered. The function f is not unique. For any strictly increasing function $g : \mathbb{R} \rightarrow \mathbb{R}$, $g(f(.))$ is a new ranking function representing the same preference relation. It may be noted that $x \sim y \Leftrightarrow f(x) = f(y)$.

The ranking function is similar to the *utility function* used in microeconomic theory [1], where utility is a measure of the satisfaction gained by consuming commodities. A consequence of using a ranking function is that the learnt preference relation is *rational*. In economics, a preference relation $\succeq$ is called rational if it satisfies the following two properties [1]

- *Completeness*. $\forall x, y \in \mathcal{X}$, we have that $x \succeq y$ or $y \succeq x$.
- *Transitivity*. $\forall x, y, z \in \mathcal{X}$, if $x \succeq y$ and $y \succeq z$, then $x \succeq z$.

A preference relation can be represented by a ranking function only if it is rational: For all x, $y \in \mathcal{X}$ either $f(x) \geq f(y)$ or $f(y) \geq f(x)$. This proves the completeness property. For all x, y, $z \in \mathcal{X}$, $f(x) \geq f(y)$ and $f(y) \geq f(z)$, implies that $f(x) \geq f(z)$. Hence, transitivity is satisfied.

A central tenet of microeconomic theory is that many of the human preferences can be assumed to be rational [1]. In the training data, we may have preferences that do not obey transitivity. However, the learnt ranking function will correspond to a rational preference relation. For the rest of the paper, we shall simply treat the learning of a preference relation as a problem of learning a rational ranking function.

1.2 Problem Statement

In the literature, the problem of learning a ranking function has been formalized in many ways. We adopt a general formulation based on directed preference graphs [2], [3].

We are given training data $\mathcal{A}$, a directed preference graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ encoding the preference relations, and a function class $\mathcal{F}$ from which we choose our ranking function f.

- The training data $\mathcal{A} = \bigcup_{j=1}^{S} (\mathcal{A}^j = \{x_i^j \in \mathbb{R}^d\}_{i=1}^{m_j})$ contains S classes (sets). Each class $\mathcal{A}^j$ contains m_j samples, and there are a total of $m = \sum_{j=1}^{S} m_j$ samples in $\mathcal{A}$.
- Each vertex of the directed order graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ corresponds to a class $\mathcal{A}^j$. The existence of a directed

- *V.C. Raykar and B. Krishnapuram are with CAD and Knowledge Solutions (IKM CKS), Siemens Medical Solutions Inc., 51 Valley Stream Pkwy, Malvern, PA 19355.*
  E-mail: {vikas.raykar, balaji.krishnapuram}@siemens.com.
- *R. Duraiswami is with the Department of Computer Science, University of Maryland, A.V. Williams Building, Room 3365, College Park MD 20742.*
  E-mail: ramani@umiacs.umd.edu.

Manuscript received 21 Oct. 2006; revised 16 May 2007; accepted 6 Aug. 2007; published online 6 Sept. 2007.
Recommended for acceptance by H. Shum.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-0745-1006.
Digital Object Identifier no. 10.1109/TPAMI.2007.70776.

0162-8828/08/$25.00 © 2008 IEEE Published by the IEEE Computer Society

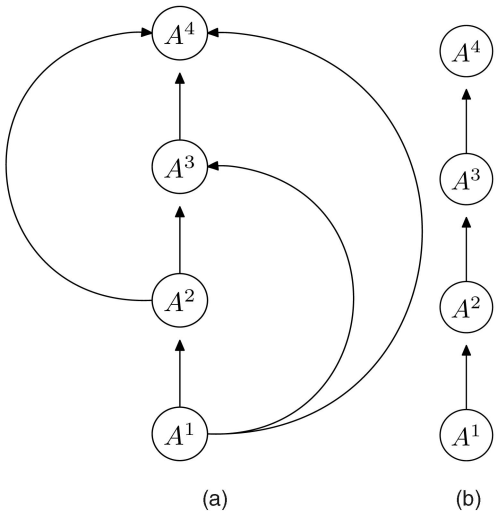


Fig. 1. (a) A full preference graph and (b) chain preference graph for a ranking problem with four classes.

edge $\mathcal{E}_{ij}$ from $\mathcal{A}^i \rightarrow \mathcal{A}^j$ means that all training samples in $\mathcal{A}^j$ are *preferred* or *ranked higher* than any training sample in $\mathcal{A}^i$, that is, $\forall(x_k^i \in \mathcal{A}^i, x_l^j \in \mathcal{A}^j), x_l^j \succeq x_k^i$ (see Fig. 1).

The goal is to learn a ranking function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ such that $f(x_l^j) \succeq f(x_k^i)$ for as many pairs as possible in the training data $\mathcal{A}$ and also to perform well on unseen examples. The output $f(x_k)$ can be sorted to obtain a rank ordering for a set of test samples $\{x_k \in \mathbb{R}^d\}$.

This general formulation gives us the flexibility to learn different kinds of preference relations by changing the preference graph. Fig. 1 shows two different ways to encode the preferences for a ranking problem with four classes. The first one containing all possible relations is called the full preference graph.

Although a ranking function can be obtained by learning classifiers or ordinal regressors, it is more advantageous to learn the ranking function directly due to two reasons:

- First, in many scenarios, it is more natural to obtain training data for pairwise preference relations rather than the actual labels for individual samples.
- Second, the loss function used for measuring the accuracy of classification or ordinal regression—for example, the 0-1 loss function—is computed for every sample individually, and then averaged over the training or the test set. In contrast, to asses the quality of the ranking for arbitrary preference graphs, we will use a generalized version of the *Wilcoxon-Mann-Whitney* statistic [2], [4], [5], that is, averaged over *pairs* of samples.

1.3 Generalized Wilcoxon-Mann-Whitney Statistic

The Wilcoxon-Mann-Whitney (WMW) statistic [4], [5] is frequently used to assess the performance of a classifier because of its equivalence to the area under the Receiver Operating Characteristics (ROC) curve (AUC). It is equal to the probability that a classifier assigns a higher value to the positive example than to the negative example, for a randomly drawn *pair of samples*. The generalized version of the WMW statistic for our ranking problem is defined as follows [2]:

$$\mathrm{WMW}(f, \mathcal{A}, \mathcal{G}) = \frac{\sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} \mathbf{1}_{f(x_l^j) \geq f(x_k^i)}}{\sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} 1}, \tag{2}$$

$$\text{where} \quad \mathbf{1}_{a \geq b} = \begin{cases} 1 & \text{if } a \geq b, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The numerator counts the number of correct pairwise orderings. The denominator is the total number of pairwise preference relations available. The WMW statistic is thus an estimate of $\Pr[f(x_1) \geq f(x_0)]$ for a randomly drawn pair of samples (x_1, x_0) such that $x_1 \succeq x_0$. This is a generalization of the area under the ROC curve (often used to evaluate bipartite rankings) to arbitrary preference graphs between many classes of samples. For a perfect ranking function, the WMW statistic is 1, and for a completely random assignment, the expected WMW statistic is 0.5.

A slightly more general formulation can be found in [3], [6], [7], where each edge in the graph has an associated weight, which indicates the strength of the preference relation. In such a case, each term in the WMW statistic must be suitably weighted.

Although the WMW statistic has been used widely to evaluate a learned model, it has only recently been used as an objective function to learn the model. Since maximizing the WMW statistic is a discrete optimization problem, most previous algorithms optimize a continuous relaxation instead. Previous algorithms often incurred $\mathcal{O}(m^2)$ effort in order to evaluate the relaxed version or its gradient. This led to very large training times for massive data sets.

1.4 Our Proposed Approach

In this paper, we directly maximize the relaxed version of the WMW statistic using a conjugate gradient (CG) optimization procedure. The gradient computation scales as $\mathcal{O}(m^2)$, which is computationally intractable for large data sets. Inspired by the fast multipole methods in computational physics [8], we develop a new algorithm that allows us to compute the gradient approximately to ϵ accuracy in $\mathcal{O}(m)$ time. This enables the learning algorithm to scale well to massive data sets.

1.5 Organization

The rest of the paper is structured as follows: In Section 2, we describe the previous work in ranking and place our method in context. The cost function that we optimize is described in Section 3. We also show that the cost function derived from a probabilistic framework can be considered as a regularized lower bound on the WMW statistic (see Section 3.1). The computational complexity of the gradient computation is analyzed in Section 4.2. In Section 5, we describe the fast summation of erfc functions–a main contribution of this paper—which makes the learning algorithm scalable for large data sets. Experimental results are presented in Section 6 and 7.

2 Previous Literature on Learning Ranking Functions

Many ranking algorithms have been proposed in the literature. Most learn a ranking function from pairwise relations and as a consequence are computationally expensive to train as the number of pairwise constraints is quadratic in the number of samples.

2.1 Methods Based on Pairwise Relations

The problem of learning rankings was first treated as a classification problem on pairs of objects by Herbrich et al. [9] and, subsequently, used on a Web page ranking task by Joachims [10]. The positive and negative examples are constructed from pairs of training examples—for example, Herbrich et al. [9] use the difference between the feature vectors of two training examples as a new feature vector for that pair. Algorithms similar to SVMs were used to learn the ranking function.

Burges et al. [6] proposed the RankNet, which uses a neural network to model the underlying ranking function. Similar to our approach, it uses gradient descent techniques to optimize a probabilistic cost function—the cross entropy. The neural net is trained on pairs of training examples using a modified version of the backpropagation algorithm.

Several boosting-based algorithms have been proposed for ranking. With collaborative filtering as an application Freund et al. [7] proposed the RankBoost algorithm for combining preferences. Dekel et al. [3] present a general framework for label ranking by means of preference graphs and graph decomposition procedure. A log-linear model is learnt using a boosting algorithm.

A probabilistic kernel approach to preference learning based on Gaussian processes was proposed by Chu and Ghahramani [11].

2.2 Fast Approximate Algorithms

The naive optimization strategy proposed in all the above algorithms suffer from the $\mathcal{O}(m^2)$ growth in the number of constraints. Fast approximate methods have only recently been investigated. An efficient implementation of the RankBoost algorithm for two class problems was presented in [7]. A convex-hull-based relaxation scheme was proposed in [2]. In a recent paper, Yan and Hauptmann [12] proposed an approximate margin-based rank learning framework by bounding the pairwise risk function. This reduced the computational cost of computing the risk function from quadratic to linear. Recently, an extension of RankNet, called LambdaRank, was proposed [13], which speeds up the algorithm by reducing the pairwise part of the computation to a loop, which can be computed very quickly. Although they showed good experimental evidence for the speedup obtained, the method still has a pairwise dependence.

2.3 Other Approaches

A parallel body of literature has considered online algorithms and sequential update methods, which find solutions in single passes through the data. PRank [14], [15] is a perceptron-based online ranking algorithm that learns using one example at a time. RankProp [16] is a neural net ranking model that is trained on individual examples rather than pairs. However, it is not known whether the algorithm converges. All gradient-based learning methods can also be trained using stochastic gradient descent techniques.

2.4 WMW Statistic Maximizing Algorithms

Our proposed algorithm directly maximizes the WMW statistic. Previous algorithms that explicitly try to maximize the WMW statistic come in two different flavors. Since the WMW statistic is not a continuous function, various approximations have been used.

A class of these methods have a Support Vector Machine (SVM)-type flavor, where the hinge loss is used as a convex upper bound for the 0-1 indicator function [7], [17], [18], [19]. Algorithms similar to the SVMs were used to learn the ranking function.

Another class of methods use a sigmoid [20] or a polynomial approximation [17] to the 0-1 loss function. Similar to our approach, they use a gradient-based learning algorithm.

2.5 Relationship to the Current Paper

Similar to the papers mentioned, our algorithm is also based on the common approach of trying to correctly arrange pairs of samples, treating them as independent. However, our algorithm differs from the previous approaches in the following ways:

- Most of the proposed approaches [3], [6], [9], [10], [11], [21] are computationally expensive to train due to the quadratic scaling in the number of pairwise constraints. Although the number of pairwise constraints is quadratic, the proposed algorithm is still linear. This is achieved by an efficient algorithm for the fast approximate summation of erfc functions, which allows us to factor the computations.
- There are no approximations in our ranking formulation, as in [12], where in order to reduce the quadratic growth, a bound on the risk functional is used. It should be noted that we use approximations only in the gradient computation of the optimization procedure. As a result, the optimization will converge to the same solution but will take a few more iterations.
- The other approximate algorithm [2] scales well to large data sets computationally, but it makes very coarse approximations by summarizing the slack variables for an entire class by a single common scalar value.
- The cost function that we optimize is a lower bound on the WMW statistic—the measure that is frequently used to assess the quality of rankings. Previous approaches that try to maximize the WMW statistic [7], [17], [18], [19], [20] consider only a classification problem and also incur the quadratic growth in the number of constraints.
- Also, to optimize our cost function, we use the nonlinear conjugate gradient algorithm—which converges much more rapidly than the steepest gradient method used for instance by the backpropagation algorithm in RankNet [6].

3 The MAP Estimator for Learning Ranking Functions

In this paper, we will consider the family of linear ranking functions: $\mathcal{F} = \{f_w\}$, where for any x, $w \in \mathbb{R}^d$, $f_w(x) = w^T x$.

Although we want to choose w to maximize the generalized $\mathrm{WMW}(f_w, \mathcal{A}, \mathcal{G})$, for computational efficiency, we shall instead maximize a continuous surrogate via the log likelihood

$$\begin{aligned}\mathcal{L}(f_w, \mathcal{A}, \mathcal{G}) &= \log \Pr[\text{correct ranking}|w] \\ &\approx \log \prod_{\mathcal{E}_{ij}} \prod_{k=1}^{m_i} \prod_{l=1}^{m_j} \Pr\left[f_w(x_l^j) > f_w(x_k^i)|w\right].\end{aligned} \tag{4}$$

Note that in (4), in common with most papers [6], [9], [11], we have assumed that every pair (x_l^j, x_k^i) is drawn independently, whereas only the original samples are drawn independently.

We use the sigmoid function to model the pairwise probability, that is,

$$\Pr\left[f_w(x_l^j) > f_w(x_k^i)|w\right] = \sigma\left[w^T(x_l^j - x_k^i)\right], \tag{5}$$

$$\text{where} \quad \sigma(z) = \frac{1}{1+e^{-z}}, \tag{6}$$

is the sigmoid function (see Fig. 3a). The sigmoid function has been previously used in [6] to model pairwise posterior probabilities. However, the cost function used was the cross-entropy.

We will assume a spherical Gaussian prior $p(w) = \mathcal{N}(w|0, \lambda^{-1}\mathbf{I})$ on the weights w. This encapsulates our prior belief that the individual weights in w are independent and close to zero with a variance parameter $1/\lambda$. The optimal *maximum a posteriori* (MAP) estimator is of the form

$$\widehat{w}_{\text{MAP}} = \arg\max_w L(w), \tag{7}$$

where $L(w)$ is the penalized log likelihood:

$$L(w) = -\frac{\lambda}{2}\|w\|^2 + \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} \log \sigma\left[w^T(x_l^j - x_k^i)\right]. \tag{8}$$

The parameter λ is also known as the regularization parameter. A similar objective function was also derived in [11] based on a Gaussian process framework.

3.1 Lower Bounding the WMW Statistic

Comparing the log-likelihood $L(w)$ (8) to the WMW statistic (2), we can see that this is equivalent to lower bounding the 0-1 indicator function in the WMW statistic by a log-sigmoid function (see Fig. 2), that is,

$$\mathbf{1}_{z>0} \geq 1 + (\log \sigma(z)/\log 2). \tag{9}$$

The log-sigmoid is appropriately scaled and shifted to make the bound tight at the origin. The log-sigmoid bound was also used in [3] along with a boosting algorithm. Therefore, maximizing the penalized log likelihood is equivalent to maximizing a lower bound on the WMW statistic. The prior acts as a regularizer.

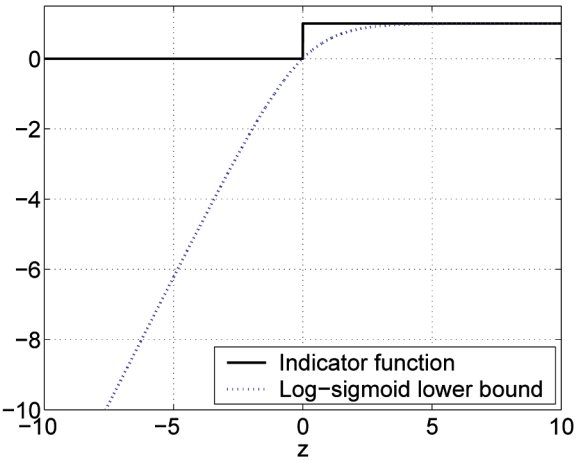


Fig. 2. Log-sigmoid lower bound for the 0-1 indicator function.

4 The Optimization Algorithm

In order to find the w that maximizes the penalized log likelihood, we use the Polak-Ribière variant of nonlinear *conjugate gradients* (CG) algorithm [22]. The CG method only needs the gradient $g(w)$ and does not require evaluation of $L(w)$. It also avoids the need for computing the second derivatives (Hessian matrix). The gradient vector is given by (using the fact that $\sigma'(z) = \sigma(z)\sigma(-z)$ and $\sigma(-z) = 1 - \sigma(z)$):

$$g(w) = -\lambda w - \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} (x_k^i - x_l^j)\sigma\left[w^T(x_k^i - x_l^j)\right]. \tag{10}$$

Notice that the evaluation of the penalized log likelihood or its gradient requires $\mathcal{M}^2 = \sum_{\mathcal{E}_{ij}} m_i m_j$ operations—this quadratic scaling can be prohibitively expensive for large data sets. The main contribution of this paper is an extremely fast method to compute the gradient approximately (Section 5).

4.1 Gradient Approximation Using the Error Function

We shall rely on the approximation (see Fig. 3a):

$$\sigma(z) \approx 1 - \frac{1}{2}\text{erfc}\left(\frac{\sqrt{3}z}{\sqrt{2}\pi}\right), \tag{11}$$

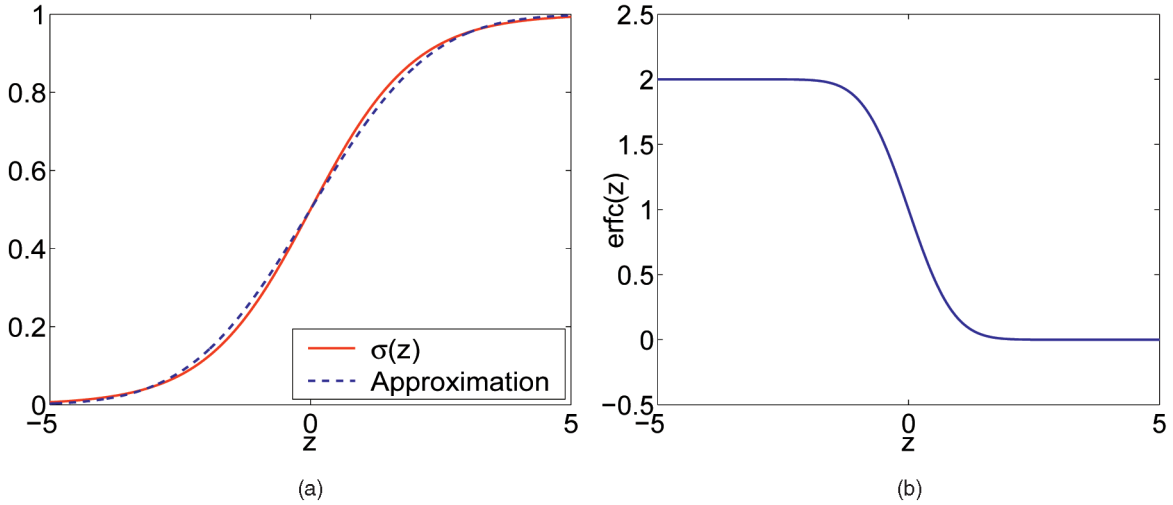


Fig. 3. (a) Approximation of the sigmoid function $\sigma(z) \approx 1 - \frac{1}{2}\text{erfc}(\frac{\sqrt{3}z}{\sqrt{2}\pi})$. (b) The erfc function.

where the complementary error function (Fig. 3b) is defined in [23]

$$\mathrm{erfc}(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2} dt. \tag{12}$$

Note that $\mathrm{erfc}(z) = 1 - \mathrm{erf}(z)$, where $\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$ is the error function encountered in integrating the normal distribution. As a result, the approximate gradient can be computed—still with $\mathcal{M}^2$ operations—as

$$g(w) \approx -\lambda w - \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} (x_k^i - x_l^j) \left[1 - \frac{1}{2} \mathrm{erfc}\left(\frac{\sqrt{3} w^T (x_k^i - x_l^j)}{\sqrt{2}\pi} \right) \right]. \tag{13}$$

4.2 Quadratic Complexity of Gradient Evaluation

We will isolate the key computational primitive contributing to the quadratic complexity in the gradient computation. The following summarizes the different variables in analyzing the computational complexity of evaluating the gradient.

- We have S classes with m_i training instances in the ith class.
- Hence, we have a total of $m = \sum_{i=1}^S m_i$ training examples in d dimensions.
- $|\mathcal{E}|$ is the number of edges in the preference graph.
- $\mathcal{M}^2 = \sum_{\mathcal{E}_{ij}} m_i m_j$ is the total number of pairwise preference relations.

For any x, we will define $z = \sqrt{3} w^T x / (\pi\sqrt{2})$. Note that z is a scalar and, for a given w, can be computed in $\mathcal{O}(dm)$ operations for the entire training set. We will now rewrite the gradient as

$$g(w) = -\lambda w - \Delta_1 + \frac{1}{2}\Delta_2 - \frac{1}{2}\Delta_3, \tag{14}$$

where the vectors Δ_1, Δ_1, and Δ_3 are defined as follows:

$$\begin{aligned} \Delta_1 &= \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} (x_k^i - x_l^j), \\ \Delta_2 &= \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} x_k^i \mathrm{erfc}(z_k^i - z_l^j), \\ \Delta_3 &= \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} x_l^j \mathrm{erfc}(z_k^i - z_l^j). \end{aligned} \tag{15}$$

The vector Δ_1 is independent of w and can be written as follows:

$$\Delta_1 = \sum_{\mathcal{E}_{ij}} m_i m_j (x_{\mathrm{mean}}^i - x_{\mathrm{mean}}^j), \text{where } x_{\mathrm{mean}}^i = \frac{1}{m_i} \sum_{k=1}^{m_i} x_k^i.$$

is the mean of all the training instances in the ith class. Hence, Δ_1 can be precomputed in $\mathcal{O}(|\mathcal{E}|d + dm)$ operations.

The other two terms Δ_2 and Δ_3 can be written as follows:

$$\Delta_2 = \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} x_k^i E_-^j(z_k^i) \quad \Delta_3 = \sum_{\mathcal{E}_{ij}} \sum_{l=1}^{m_j} x_l^j E_+^i(-z_l^j), \tag{16}$$

where

$$\begin{aligned} E_-^j(y) &= \sum_{l=1}^{m_j} \mathrm{erfc}(y - z_l^j), \\ E_+^i(y) &= \sum_{k=1}^{m_i} \mathrm{erfc}(y + z_k^i). \end{aligned} \tag{17}$$

Note that $E_-^j(y)$ in the sum of m_j erfc functions centered at z_l^j and evaluated at y—which requires $\mathcal{O}(m_j)$ operations. In order to compute Δ_2, we need to evaluate it at m_i points, thus requiring $\mathcal{O}(m_i m_j)$ operations. Hence, each of Δ_2 and Δ_3 can be computed in $\mathcal{O}(dSm + \mathcal{M}^2)$ operations.

Hence, the core computational primitive contributing to the $\mathcal{O}(\mathcal{M}^2)$ cost is the summation of erfc functions. In the next section, we will show how this sum can be computed in linear $\mathcal{O}(m_i + m_j)$ time, at the expense of reduced accuracy, which however can be arbitrary. As a result of this, Δ_2 and Δ_3 can be computed in linear $\mathcal{O}(dSm + (S-1)m)$ time.

In terms of the optimization algorithm since the gradient is computed approximately, the number of iterations required to converge may increase. However, this is more than compensated by the cost per iteration, which is drastically reduced.

5 Fast Weighted Summation of ERFC Functions

In general, $E_-^j(y)$ and $E_+^i(y)$ can be written as the weighted summation of N erfc functions centered at $z_i \in \mathcal{R}$ with weights $q_i \in \mathcal{R}$

$$E(y) = \sum_{i=1}^N q_i \, \mathrm{erfc}(y - z_i). \tag{18}$$

Direct computation of (18) at M points $\{y_j \in \mathcal{R}\}_{j=1}^M$ is $\mathcal{O}(MN)$. In this section, we will derive an ϵ-accurate approximation algorithm to compute this in $\mathcal{O}(M+N)$ time.

5.1 ϵ-Accurate Approximation

For any given $\epsilon > 0$, we define $\widehat{E}$ to be an ϵ-accurate approximation to E if the maximum absolute error relative to the total weight $Q_{abs} = \sum_{i=1}^N |q_i|$ is upper bounded by a specified ϵ, that is,

$$\max_{y_j} \left[\frac{|\hat{E}(y_j) - E(y_j)|}{Q_{abs}} \right] \le \epsilon. \tag{19}$$

The constant in $\mathcal{O}(M+N)$ for our algorithm depends on the desired accuracy ϵ, which however can be *arbitrary*. In fact, for machine precision accuracy, there is no difference between the direct and the fast methods. The algorithm we present is inspired by the fast multipole methods proposed in computational physics [8]. The fast algorithm is based on using an infinite series expansion for the erfc function and retaining only the first few terms (whose contribution is at the desired accuracy).

5.2 Series Expansion for erfc Function

Several series exist for the erfc function (see Chapter 7 in [23]). Some are applicable only to a restricted interval, whereas others need a large number of terms to converge. We use the following truncated Fourier series representation derived by Beauliu [24], [25]:

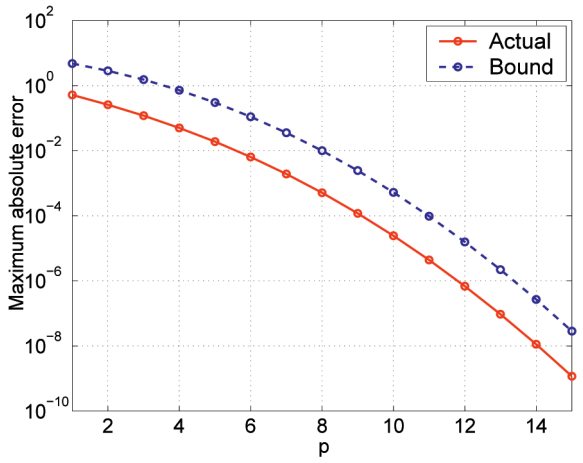


Fig. 4. The maximum absolute error between the actual value of erfc and the truncated series representation (20) as a function of the truncation number p for any $z \in [-4, 4]$. The error bound (22) is also shown as a dotted line.

$$\operatorname{erfc}(z) = 1 - \frac{4}{\pi} \sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} \frac{e^{-n^2h^2}}{n} \sin(2nhz) + \text{error}(z), \tag{20}$$

$$|\text{error}(z)| < \left| \frac{4}{\pi} \sum_{\substack{n=2p+1 \\ n \text{ odd}}}^{\infty} \frac{e^{-n^2h^2}}{n} \sin(2nhz) \right| + \operatorname{erfc}\left(\frac{\pi}{2h} - |z|\right). \tag{21}$$

Here, p is known as the *truncation number*, and h is a real number related to the sampling interval. The series is derived by applying a Chernoff bound to an approximate Fourier series expansion of a periodic square waveform [24]. This series converges rapidly, especially as $z \to 0$. Fig. 4 shows the maximum absolute error between the actual value of erfc and the truncated series representation as a function of p. For example, for any $z \in [-4, 4]$ with $p = 12$, the error is less than 10^{-6}.

5.3 Error Bound

We will have to choose p and h such that the error is less than the desired ϵ. For this purpose, we further bound the first term in (21) as follows:

$$\begin{aligned}
&\left| \frac{4}{\pi} \sum_{\substack{n=2p+1 \\ n \text{ odd}}}^{\infty} \frac{e^{-n^2h^2}}{n} \sin(2nhx) \right| \\
&\leq \frac{4}{\pi} \sum_{\substack{n=2p+1 \\ n \text{ odd}}}^{\infty} \frac{e^{-n^2h^2}}{n} |\sin(2nhx)| \\
&\leq \frac{4}{\pi} \sum_{\substack{n=2p+1 \\ n \text{ odd}}}^{\infty} \frac{e^{-n^2h^2}}{n} \quad [\text{Since} |\sin(2nhx)| \leq 1] \\
&< \frac{4}{\pi} \sum_{\substack{n=2p+1 \\ n \text{ odd}}}^{\infty} e^{-n^2h^2} \quad [\text{Since } 1/n \leq 1] \\
&< \frac{4}{\pi} \int_{2p+1}^{\infty} e^{-x^2h^2} dx \quad \left[\text{Replacing } \sum \text{ by } \int \right] \\
&< \frac{2}{\sqrt{\pi}h} \left[\frac{2}{\sqrt{\pi}} \int_{(2p+1)h}^{\infty} e^{-t^2} dt \right] \\
&= \frac{2}{\sqrt{\pi}h} \operatorname{erfc}((2p+1)h).
\end{aligned}$$

Hence, the final error bound is of the form:

$$|\text{error}(z)| < \frac{2}{\sqrt{\pi}h} \operatorname{erfc}((2p+1)h) + \operatorname{erfc}\left(\frac{\pi}{2h} - |z|\right). \tag{22}$$

The error bound is shown as a dotted line in Fig. 4.

5.4 Fast Summation Algorithm

We now derive a fast algorithm to compute $E(y)$ based on the series (20).

$$\begin{aligned}
E(y) &= \sum_{i=1}^{N} q_i \operatorname{erfc}(y - z_i) \\
&= \sum_{i=1}^{N} q_i \left[1 - \frac{4}{\pi} \sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} \frac{e^{-n^2h^2}}{n} \sin\{2nh(y - z_i)\} + \text{error} \right].
\end{aligned} \tag{23}$$

Ignoring the error term for the time being, the sum $E(y)$ can be approximated as

$$\widehat{E}(y) = Q - \frac{4}{\pi} \sum_{i=1}^{N} q_i \sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} \frac{e^{-n^2h^2}}{n} \sin\{2nh(y - z_i)\}, \tag{24}$$

where $Q = \sum_{i=1}^{N} q_i$. The terms y and z_i are entangled in the argument of the sin function, leading to a quadratic complexity. The crux of the algorithm is to separate them using the trigonometric identity

$$\begin{aligned}
&\sin\{2nh(y - z_i)\} \\
&= \sin\{2nh(y - z_*) - 2nh(z_i - z_*)\} \\
&= \sin\{2nh(y - z_*)\} \cos\{2nh(z_i - z_*)\} \\
&\quad - \cos\{2nh(y - z_*)\} \sin\{2nh(z_i - z_*)\}.
\end{aligned} \tag{25}$$

Note that we have shifted all the points by z_*. The reason for this will be clearer later in Section 5.7, where we cluster the points and use the series representation around different cluster centers. Substituting the separated representation in (24), we have

$$\begin{aligned}
\widehat{E}(y) = Q &- \frac{4}{\pi} \sum_{i=1}^{N} q_i \sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} \frac{e^{-n^2h^2}}{n} \sin\{2nh(y - z_*)\} \cos\{2nh(z_i - z_*)\} \\
&+ \frac{4}{\pi} \sum_{i=1}^{N} q_i \sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} \frac{e^{-n^2h^2}}{n} \cos\{2nh(y - z_*)\} \sin\{2nh(z_i - z_*)\}.
\end{aligned} \tag{26}$$

Exchanging the order of summation and regrouping the terms, we have the following expression:

$$\begin{aligned}
\widehat{E}(y) = Q &- \frac{4}{\pi} \sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} A_n \sin\{2nh(y - z_*)\} \\
&+ \frac{4}{\pi} \sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} B_n \cos\{2nh(y - z_*)\},
\end{aligned} \tag{27}$$

where

$$A_n = \frac{e^{-n^2h^2}}{n}\sum_{i=1}^{N} q_i \cos\{2nh(z_i - z_*)\}, \quad \text{and}$$
$$B_n = \frac{e^{-n^2h^2}}{n}\sum_{i=1}^{N} q_i \sin\{2nh(z_i - z_*)\}. \tag{28}$$

5.5 Computational and Space Complexity

Note that the coefficients $\{A_n, B_n\}$ do not depend on y. Hence, each of A_n and B_n can be evaluated separately in $\mathcal{O}(N)$ time. Since there are p such coefficients the total complexity to compute A and B is $\mathcal{O}(pN)$. The term $Q = \sum_{i=1}^{N} q_i$ can also be precomputed in $\mathcal{O}(N)$ time. Once A, B, and Q have been precomputed, evaluation of $\widehat{E}(y)$ requires $\mathcal{O}(p)$ operations. Evaluating at M points is $\mathcal{O}(pM)$. Therefore, the computational complexity has reduced from the quadratic $\mathcal{O}(NM)$ to the linear $\mathcal{O}(p(N+M))$. We need space to store the points and the coefficients A and B. Hence, the storage complexity is $\mathcal{O}(N+M+p)$.

5.6 Direct Inclusion and Exclusion of Far Away Points

From (22), it can be seen that for a fixed p and h as $|z|$ increases the error increases. Therefore, as $|z|$ increases, h should decrease, and consequently, the series converges slower leading to a large truncation number p.

Note that $s = (y - z_i) \in [-\infty, \infty]$. The truncation number p required to approximate $\operatorname{erfc}(s)$ can be quite large for large $|s|$. Luckily, $\operatorname{erfc}(s) \to 2$ as $s \to -\infty$, and $\operatorname{erfc}(s) \to 0$ as $s \to \infty$ very quickly (see Fig. 3b). Since we only want an accuracy of ϵ, we can use the approximation

$$\operatorname{erfc}(s) \approx \begin{cases} 2 & \text{if } s < -r, \\ p\text{-truncated series} & \text{if } -r \le s \le r, \\ 0 & \text{if } s > r. \end{cases} \tag{29}$$

The bound r and the truncation number p have to be chosen such that for any s, the error is always less than ϵ. For example, for error of the order 10^{-15}, we need to use the series expansion for $-6 \le s \le 6$. However, we cannot check the value of $(y - z_i)$ for all pairs of z_i and y. This would lead us back to the quadratic complexity. To avoid this, we subdivide the points into clusters.

5.7 Space Subdivision

We uniformly subdivide the domain into K intervals of length $2r_x$. The N source points are assigned into K clusters S_k, for $k = 1, \ldots, K$ with c_k being the center of each cluster. The aggregated coefficients are computed for each cluster, and the total contribution from all the influential clusters is summed up. For each cluster, if $|y - c_k| \le r_y$, we will use the series coefficients. If $(y - c_k) < -r_y$, we will include a contribution of $2Q_k$; if $(y - c_k) > r_y$, we will ignore that cluster. The cutoff radius r_y has to be chosen to achieve a given accuracy. Hence,

$$\begin{aligned} \widehat{E}(y) = &\sum_{|y-c_k|\le r_y} Q_k \\ &- \sum_{|y-c_k|\le r_y} \frac{4}{\pi}\sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} A_n^k \sin\{2nh(y-c_k)\} \\ &+ \sum_{|y-c_k|\le r_y} \frac{4}{\pi}\sum_{\substack{n=1 \\ n \text{ odd}}}^{2p-1} B_n^k \cos\{2nh(y-c_k)\} \\ &+ \sum_{(y-c_k)<-r_y} 2Q_k, \end{aligned} \tag{30}$$

where

$$\begin{aligned} A_n^k &= \frac{e^{-n^2h^2}}{n}\sum_{i=1}^{N} q_i \cos\{2nh(z_i - c_k)\}, \\ B_n^k &= \frac{e^{-n^2h^2}}{n}\sum_{i=1}^{N} q_i \sin\{2nh(z_i - c_k)\}, \text{and} \\ Q_k &= \sum_{\forall z_i \in S_k} q_i. \end{aligned} \tag{31}$$

The computational complexity to compute A, B, and Q is still $\mathcal{O}(pN)$ since each z_i belongs to only one cluster. Let l be the number of influential clusters, that is, the clusters for which $|y - c_k| \le r_y$. Evaluating $\widehat{E}(y)$ at M points due to these l clusters is $\mathcal{O}(plM)$. Let m be the number of clusters for which $(y - c_k) < -r_y$. Evaluating $\widehat{E}(y)$ at M points due to these m clusters is $\mathcal{O}(mM)$. Hence, the total computational complexity is $\mathcal{O}(pN + (pl + m)M)$. The storage complexity is $\mathcal{O}(N + M + pK)$.

5.8 Choosing the Parameters

Given any $\epsilon > 0$, we want to choose the following parameters, r_x (the interval length), r_y (the cut off radius), p (the truncation number), and h such that for any target point y:

$$\left|\frac{\hat{E}(y) - E(y)}{Q_{abs}}\right| \le \epsilon, \tag{32}$$

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

Let us define Δ_i to be the pointwise error in $\widehat{E}(y)$ contributed by the ith source z_i. We now require that

$$|\hat{E}(y) - E(y)| = \left|\sum_{i=1}^{N} \Delta_i\right| \le \sum_{i=1}^{N} |\Delta_i| \le \sum_{i=1}^{N} |q_i|\epsilon. \tag{33}$$

One way to achieve this is to let $|\Delta_i| \le |q_i|\epsilon \; \forall i = 1, \ldots, N$. For all z_i such that $|y - z_i| \le r$, we have (22)

$$|\Delta_i| < \underbrace{|q_i|\frac{2}{\sqrt{\pi}h}\operatorname{erfc}((2p+1)h)}_{T_e} + \underbrace{|q_i|\operatorname{erfc}\Big(\frac{\pi}{2h} - r\Big)}_{S_e}. \tag{34}$$

We have to choose the parameters such that $|\Delta_i| < |q_i|\epsilon$. We will let $S_e < |q_i|\epsilon/2$. This implies that

$$\frac{\pi}{2h} - r > \operatorname{erfc}^{-1}(\epsilon/2). \tag{35}$$

Hence, we have to choose

$$h < \frac{\pi}{2\big(r + \operatorname{erfc}^{-1}(\epsilon/2)\big)}. \tag{36}$$

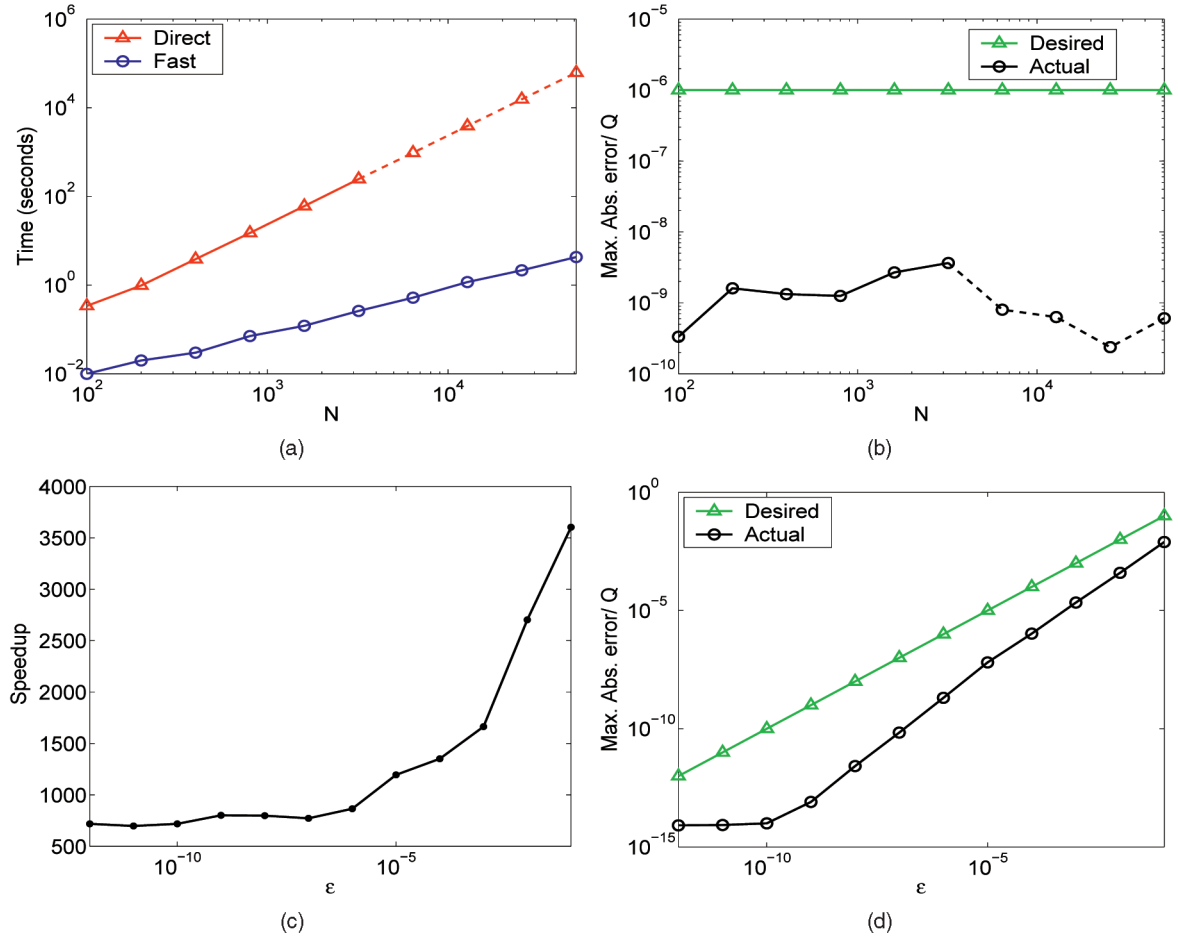


Fig. 5. (a) The running time in seconds, and (b) maximum absolute error relative to Q_{abs} for the direct and the fast methods as a function of $N(=M)$. For $N > 3,200$, the timing results for the direct evaluation were obtained by evaluating the sum at $M = 100$ points and then extrapolating (shown as dotted line). (c) The speedup achieved and (d) maximum absolute error relative to Q_{abs} for the direct and the fast methods as a function of ϵ for $N(=M) = 3,000$. Results are on a 1.6-GHz Pentium M processor with 512 Mbytes of RAM.

We will choose

$$h = \frac{\pi}{3\big(r + \text{erfc}^{-1}(\epsilon/2)\big)}. \tag{37}$$

We will choose p such that $T_e < |q_i|\epsilon/2$. This implies that

$$2p + 1 > \frac{1}{h}\text{erfc}^{-1}\left(\frac{\sqrt{\pi}h\epsilon}{4}\right). \tag{38}$$

We choose

$$p = \left\lceil \frac{1}{2h}\text{erfc}^{-1}\left(\frac{\sqrt{\pi}h\epsilon}{4}\right)\right\rceil. \tag{39}$$

Note that as r increases, h decreases, and, consequently, p increases. If $s \in (r, \infty]$, we approximate $\text{erfc}(s)$ by 0, and if $s \in [-\infty, -r)$, then approximate $\text{erfc}(s)$ by 2. If we choose

$$r > \text{erfc}^{-1}(\epsilon), \tag{40}$$

then the approximation will result in a error $< \epsilon$. In practice, we choose

$$r = \text{erfc}^{-1}(\epsilon) + 2r_x, \tag{41}$$

where r_x is the cluster radius. For a target point y, the number of influential clusters

$$(2l + 1) = \left\lceil \frac{2r}{2r_x} \right\rceil. \tag{42}$$

Let us choose $r_x = 0.1\text{erfc}^{-1}(\epsilon)$. This implies $2l + 1 = 12$. Therefore, we have to consider six clusters on either side of the target point. Summarizing, the parameters are given by

- $r_x = 0.1\text{erfc}^{-1}(\epsilon)$.
- $r = \text{erfc}^{-1}(\epsilon) + 2r_x$.
- $h = \pi/3\big(r + \text{erfc}^{-1}(\epsilon/2)\big)$.
- $p = \left\lceil \frac{1}{2h}\text{erfc}^{-1}\left(\frac{\sqrt{\pi}h\epsilon}{4}\right)\right\rceil$.
- $(2l + 1) = \lceil r/r_x \rceil$.

5.9 Numerical Experiments

We present experimental results for the core computational primitive of erfc functions. Experiments when this primitive is embedded in the optimization routine will be provided in Section 6.

We present numerical studies of the speedup and error as a function of the number of data points and the desired error ϵ. The algorithm was programmed in C++ with MATLAB bindings and was run on a 1.6 GHz Pentium M processor with 512 Mbytes of RAM. Figs. 5a and 5b shows the running time and the maximum absolute error relative to Q_{abs} for both the direct and the fast methods as a function of $N(=M)$. The points were normally distributed with zero

TABLE 1
Benchmark Data Sets Used in the Ranking Experiments

	Dataset name	N	d	S	$\mathcal{M}$		Dataset name	N	d	S	$\mathcal{M}$
1	Diabetes	43	3	2	272	8	Airplane Companies	950	10	5	217301
2	Pyrimidines	74	28	3	1113	9	RandNet	1000	50	6	195907
3	Triazines	186	61	4	7674	10	RandPoly	1000	50	6	225131
4	Wisconsin Breast Cancer	194	33	4	8162	11	Abalone	4177	9	3	3713729
5	Machine-CPU	209	7	4	9820	12	RandNet	5000	50	6	6269910
6	Auto-MPG	392	8	3	30057	13	RandPoly	5000	50	6	5367241
7	Boston Housing	506	14	2	33693	14	California Housing	20640	9	3	82420255

N is the size of the data set. d is the number of attributes. S is the number of classes. $\mathcal{M}$ is the average total number of pairwise relations per fold of the training set.

mean and unit variance. The weights q_i were set to 1. We see that the running time of the fast method grows linearly, whereas that of the direct evaluation grows quadratically. We also observe that the error is well below the permissible error, thus validating our bound. For example, for $N = M = 51,200$ points, although the direct evaluation takes around 17.26 hours, the fast evaluation requires only 4.29 sec with an error of around 10^{-10}. Fig. 5c shows the trade-off between precision and speedup. An increase in speedup is obtained at the cost of slightly reduced accuracy.

6 Ranking Experiments

6.1 Data Sets

We used two artificial data sets and 10 publicly available benchmark data sets[1] in Table 1, previously used for evaluating ranking [2] and ordinal regression [26]. Since these data sets are originally designed for regression, we discretize the continuous target values into S equal sized bins as specified in Table 1. For each data set, the number of classes S was chosen such that none of them were empty. The two data sets RandNet and RandPoly are artificial data sets generated as described in [6]. The ranking function for RandNet is generated using a random two layer neural net with 10 hidden units and RandPoly using a random polynomial.

6.2 Evaluation Procedure

For each data set, 80 percent of the examples were used for training and the remaining 20 percent were used for testing. The results are shown for a fivefold cross validation experiment. In order to choose the regularization parameter λ, on each fold, we used the training split and performed a fivefold cross validation on the training set. The performance is evaluated in terms of the generalized WMW statistic (A WMW of one implies perfect ranking). We used a full order graph to evaluate the ranking performance.

We compare the performance and the time taken for the following methods:

1. *RankNCG.* The proposed nonlinear conjugate-gradient ranking procedure. The tolerance for the conjugate gradient procedure was set to 10^{-3}. The nonlinear conjugate gradient optimization procedure was randomly initialized. We compare the following two versions:
   - *RankNCG direct.* This uses the exact gradient computations.
   - *RankNCG fast.* This uses the fast approximate gradient computation. The accuracy parameter ϵ for the fast gradient computation was set to 10^{-6}.
2. *RankNet* [6]. A neural network, which is trained using pairwise samples based on cross-entropy cost function. For training in addition to the preference relation $x_i \succeq x_j$, each pair also has an associated target posterior $\Pr[x_i \succeq x_j]$. In our experiments, we used hard target probabilities of 1 for all pairs. The best learning rate for the net was chosen using WMW as the cross validation measure. Training was done in a batch mode for around 500–1,000 epochs or until there are no function decrease in the cost function. We used two versions of the RankNet:
   - *RankNet two layer.* A two layer neural network with 10 hidden units.
   - *RankNet linear.* A single layer neural network.
3. *RankSVM* [9], [10]. A ranking function is learnt by training an SVM classifier[2] over pairs of examples. The trade-off parameter was chosen by cross validation. We used two version of the RankSVM:
   - *RankSVM linear.* The SVM is trained using a linear kernel.
   - *RankSVM quadratic.* The SVM is trained using a polynomial kernel $k(x, y) = (x.y + c)^p$ of order $p = 2$.
4. *RankBoost* [7]. A boosting algorithm, which effectively combines a set of weak ranking functions. We used {0, 1}-valued weak rankings that use the

1. The data sets were downloaded from http://www.liacc.up.pt/~ltorgo/Regression/DataSets.html.

2. Using the SVM-light packages available at http://svmlight.joachims.org.

TABLE 2
The Mean Training Time and Standard Deviation in Seconds for the Various Methods and All the Data Sets Shown in Table 1

	RankNCG direct	**RankNCG fast**	RankNet linear	RankNet two layer	RankSVM linear	RankSVM quadratic	RankBoost
1	0.11 [± 0.02]	0.06 [± 0.01]	1.79 [± 0.03]	3.32 [± 0.11]	0.09 [± 0.04]	0.10 [± 0.01]	1.70 [± 0.09]
2	0.63 [± 0.13]	0.12 [± 0.03]	7.11 [± 0.27]	13.55 [± 0.30]	0.10 [± 0.02]	0.62 [± 0.13]	1.72 [± 0.02]
3	17.63 [± 7.27]	0.70 [± 0.39]	58.14 [± 0.78]	131.41 [± 2.19]	0.55 [± 0.28]	13.96 [± 0.48]	6.70 [± 0.06]
4	13.41 [± 9.35]	0.33 [± 0.43]	48.13 [± 0.85]	97.24 [± 1.05]	0.64 [± 0.03]	23.17 [± 3.37]	1.88 [± 0.04]
5	20.38 [± 4.87]	0.97 [± 0.15]	57.99 [± 0.58]	111.14 [± 1.14]	1.14 [± 0.27]	24.46 [± 0.68]	1.24 [± 0.02]
6	28.05 [± 10.94]	0.40 [± 0.23]	175.63 [± 1.55]	333.49 [± 3.96]	0.43 [± 0.02]	37.27 [± 3.10]	1.54 [± 0.04]
7	18.92 [± 0.63]	0.16 [± 0.01]	195.14 [± 4.75]	381.28 [± 7.93]	0.36 [± 0.03]	13.93 [± 2.15]	2.32 [± 0.04]
8	332.88 [± 26.66]	3.29 [± 0.88]	1264.58 [± 3.21]	2464.84 [± 10.94]	34.32 [± 4.05]	1332.79 [± 69.47]	5.56 [± 0.37]
9	250.37 [± 21.03]	5.08 [± 0.47]	1166.23 [± 17.47]	2380.62 [± 34.53]	83.62 [± 6.30]	13628.23 [± 210.10]	13.55 [± 0.07]
10	102.48 [± 0.59]	0.78 [± 0.04]	1341.20 [± 6.91]	2733.25 [± 23.11]	1656.52 [± 99.89]	14110.48 [± 121.98]	13.99 [± 0.05]
11	1736.47 [± 191.03]	1.47 [± 0.38]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	62.91 [± 0.59]
12	6731.09 [± 312.41]	19.10 [± 1.76]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	147.04 [± 0.16]
13	2556.93 [± 15.03]	3.59 [± 0.41]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	133.42 [± 1.14]
14	⋆ [± ⋆]	46.86 [± 1.06]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]

The results are shown for a five fold cross-validation experiment. The symbol ⋆ indicates that the particular method either crashed due to limited memory requirements or took a very large amount of time.

TABLE 3
The Corresponding Generalized WMW Statistic and the Standard Deviation on the Test Set for the Results Shown in Table 2

	RankNCG direct	**RankNCG fast**	RankNet linear	RankNet two layer	RankSVM linear	RankSVM quadratic	RankBoost
1	0.677 [± 0.233]	0.650 [± 0.210]	0.579 [± 0.096]	0.479 [± 0.284]	0.545 [± 0.236]	0.400 [± 0.276]	0.675 [± 0.173]
2	0.987 [± 0.019]	0.948 [± 0.077]	0.872 [± 0.088]	0.968 [± 0.038]	0.973 [± 0.048]	0.837 [± 0.142]	0.906 [± 0.144]
3	0.942 [± 0.044]	0.914 [± 0.047]	0.828 [± 0.030]	0.891 [± 0.064]	0.934 [± 0.019]	0.861 [± 0.088]	0.651 [± 0.045]
4	0.764 [± 0.028]	0.771 [± 0.046]	0.773 [± 0.046]	0.750 [± 0.035]	0.793 [± 0.018]	0.795 [± 0.035]	0.748 [± 0.056]
5	0.920 [± 0.015]	0.938 [± 0.020]	0.919 [± 0.035]	0.923 [± 0.040]	0.929 [± 0.026]	0.901 [± 0.014]	0.926 [± 0.018]
6	0.999 [± 0.002]	0.998 [± 0.002]	0.998 [± 0.003]	0.996 [± 0.003]	0.998 [± 0.002]	0.995 [± 0.008]	0.992 [± 0.004]
7	1.000 [± 0.000]	1.000 [± 0.000]	1.000 [± 0.000]	0.800 [± 0.400]	1.000 [± 0.000]	1.000 [± 0.000]	1.000 [± 0.000]
8	0.984 [± 0.004]	0.984 [± 0.003]	0.951 [± 0.004]	0.765 [± 0.245]	0.984 [± 0.004]	0.996 [± 0.001]	0.958 [± 0.003]
9	0.944 [± 0.012]	0.944 [± 0.012]	0.915 [± 0.017]	0.899 [± 0.028]	0.945 [± 0.013]	0.747 [± 0.005]	0.848 [± 0.015]
10	0.625 [± 0.025]	0.625 [± 0.025]	0.688 [± 0.032]	0.644 [± 0.054]	0.625 [± 0.026]	0.823 [± 0.008]	0.618 [± 0.024]
11	0.536 [± 0.011]	0.534 [± 0.008]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	0.535 [± 0.014]
12	0.917 [± 0.005]	0.917 [± 0.005]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	0.845 [± 0.006]
13	0.623 [± 0.008]	0.623 [± 0.008]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	0.607 [± 0.010]
14	⋆ [± ⋆]	0.979 [± 0.001]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]	⋆ [± ⋆]

ordering information provided by the features [7]. Training a weak ranking function involves finding the best feature and the best threshold for that feature. We boosted for 50-100 rounds.

6.3 Results

The results are summarized in Tables 2 and 3. All experiments were run on a 1.83 GHz machine with 1.00 Gbytes of RAM. The following observations can be made.

6.3.1 Quality of Approximation

The WMW is similar for 1) the proposed exact method (RankNCG direct) and the 2) the approximate method (RankNCG fast). The runtime of the approximate method is one to two magnitudes lower than the exact method, especially for large data sets. Thus, we are able to get very good speedups without sacrificing ranking accuracy.

6.3.2 Comparison with Other Methods

All the methods show very similar WMW scores. In terms of the training time, the proposed method clearly beats all the other methods. For small data sets, RankSVM linear is comparable in time to our methods. For large data sets, RankBoost shows the next best time.

6.3.3 Ability to Handle Large Data Sets

For data set 14, only the fast method completed execution. The direct method and all the other methods either crashed due to huge memory requirements or took an incredibly large amount of time. Further, since the accuracy of learning (that

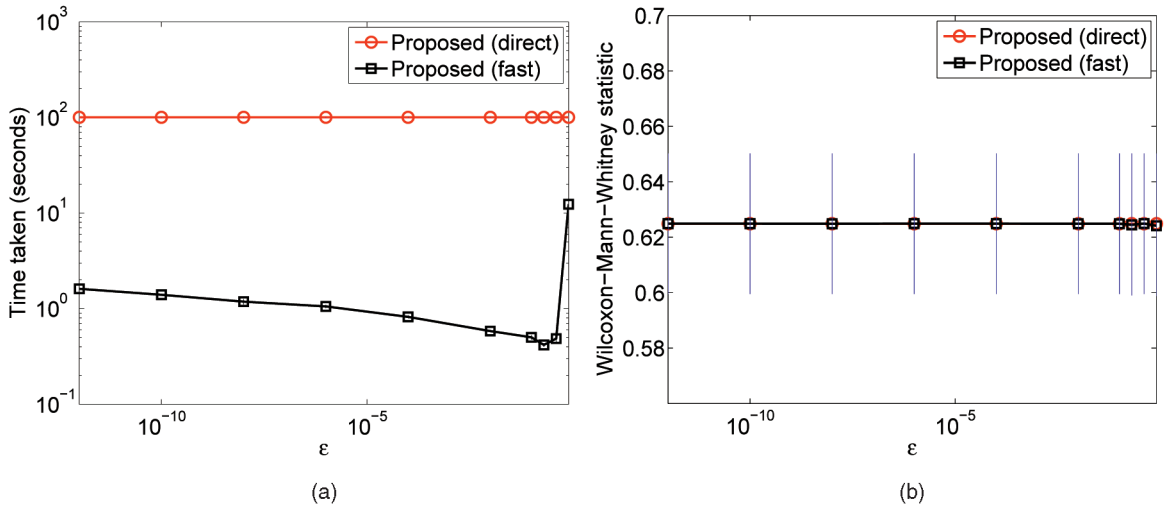


Fig. 6. *Effect of ϵ-accurate derivatives*. (a) The time taken and (b) the WMW statistic for the proposed method and the faster version of the proposed method as a function of ϵ. The CG tolerance was set to 10^{-3}. Results are for data set 10. The bars indicate $\pm$ one standard deviation.

is, estimation) clearly depends on the ability to leverage large data sets, in real life, the proposed methods are also expected to be more accurate on large-scale ranking problems.

6.4 Impact of the Gradient Approximation

Fig. 6 studies the accuracy and the runtime for data set 10 as a function of the gradient tolerance, ϵ. As ϵ increases, the time taken per-iteration (and hence, overall) decreases. However, if it is too large, the total time taken starts increasing (after $\epsilon = 10^{-2}$ in Fig. 6a). Intuitively, this is because the use of approximate derivatives slows the convergence of the conjugate gradient procedure by increasing the number of iterations required for convergence. The speedup is achieved because computing the approximate derivatives is extremely fast, thus compensating for the slower convergence. However, after a certain point the number of iterations dominates the runtime. Also, notice that ϵ has no significant effect on the WMW achieved, because the optimizer still converges to the optimal value albeit at a slower rate.

7 Application to Collaborative Filtering

As an application, we will show some results on a collaborative filtering task for movie recommendations. We use the MovieLens data set,[3] which contains approximately 1 million ratings for 3,592 movies by 6,040 users. Ratings are made on a scale of 1 to 5. The task is to predict the movie rankings for a user based on the rankings provided by other users. For each user, we used 70 percent of the movies rated by him for training and the remaining 30 percent for testing. The features for each movie consisted of the ranking provided by d other users. For each missing rating, we imputed a sample drawn from a Gaussian distribution with its mean and variance estimated from the available ratings provided by the other users. Tables 4 and 5 shows the time taken and the WMW score for this task for the two fastest methods. The results are averaged for over 100 users. The other methods took a large amount of time to train just for one user. The proposed method shows the best WMW and takes the least amount of time for training.

8 Conclusion and Future Work

In this paper, we presented an approximate ranking algorithm that directly maximizes (a regularized lower bound on) the generalized Wilcoxon-Mann-Whitney statistic. The algorithm was made computationally tractable using a novel fast summation method for calculating a weighted sum of erfc functions.[4] Experimental results demonstrate that despite the order of magnitude speedup, the accuracy was almost identical to exact method and other algorithms proposed in literature.

8.1 Future Work

Other applications for fast summation of erfc functions. The fast summation method proposed could be potentially useful in neural networks, probit regression, and in Bayesian models involving sigmoids.

Nonlinear kernelized variations. The main focus of the paper was to learn a linear ranking function. A nonlinear version of the algorithm can be easily derived using the *kernel trick* (see [9] for an SVM analog). We kernelize the algorithm by replacing the linear ranking function $f(x) = w^T x$, with $f(x) = \sum_{i=1}^{m} \alpha_i k(x, x_i) = \alpha^T \mathbf{k}(x)$, where k is the kernel used, and $\mathbf{k}(x)$ is a column vector defined by $\mathbf{k}(x) = [k(x, x_1), \ldots, k(x, x_m)]^T$. The penalized log likelihood for this problem changes to

$$L(\alpha) = -\frac{\lambda}{2}\|\alpha\|^2 + \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} \log \sigma\left[\alpha^T\left(\mathbf{k}(x_l^j) - \mathbf{k}(x_k^i)\right)\right]. \tag{43}$$

3. The data set was downloaded from http://www.grouplens.org/.

4. The software for the fast erfc summation is available on the first author's Web site at http://www.umiacs.umd.edu/~vikas/.

TABLE 4
Results for the EACHMOVIE Data Set: The Mean Training Time and the Standard Deviation in Seconds (Averaged over 100 Users) as a Function of the Number of Features d

d	RankNCG fast	RankBoost
50	0.48 [± 0.19]	6.68 [± 1.65]
100	0.44 [± 0.17]	12.67 [± 2.83]
200	0.42 [± 0.17]	27.53 [± 5.99]
400	0.41 [± 0.17]	68.08 [± 13.95]
800	0.45 [± 0.13]	193.18 [± 39.75]
1600	0.51 [± 0.15]	613.54 [± 124.93]

TABLE 5
The Corresponding Generalized WMW Statistic and the Standard Deviation on the Test Set for the Results Shown in Table 4

d	RankNCG fast	RankBoost
50	0.693 [± 0.054]	0.672 [± 0.056]
100	0.707 [± 0.049]	0.679 [± 0.050]
200	0.722[± 0.053]	0.685 [± 0.057]
400	0.720 [±0.054]	0.685 [± 0.051]
800	0.721 [± 0.050]	0.673 [± 0.058]
1600	0.719 [± 0.053]	0.682 [± 0.058]

The gradient vector is given by

$$g(\alpha) = \nabla L(\alpha) = -\lambda\alpha - \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} \big(\mathbf{k}(x_k^i) - \mathbf{k}(x_l^j)\big)\sigma\big[\alpha^T\big(\mathbf{k}(x_k^i) - \mathbf{k}(x_l^j)\big)\big]. \tag{44}$$

The gradient is now a column vector of length m, whereas it was of length d for the linear version. As a result, evaluating the gradient now requires roughly $\mathcal{O}(m^2 + \mathcal{M}^2)$ computations. The $\mathcal{O}(\mathcal{M}^2)$ part is due the weighted sum of sigmoid (or erfc) functions, for which we can use the fast approximation proposed in this paper. The $\mathcal{O}(m^2)$ part arises due to the multiplication of the $m \times m$ kernel matrix with a vector. Fast approximate matrix-vector multiplication techniques like dual-tree methods [27] and the improved fast Gauss transform [28], [29] can be used to speedup this computation. However, each of these methods have their own regions of applicability, and more experiments need to be done to evaluate the final speedups that can be obtained.

Independence of pairs of samples. In common with most papers following [9], we have assumed that every pair (x_l^j, x_k^i) is drawn independently, even though they are really correlated (actually, the samples x_k^i are drawn independently). In the future, we plan to correct this lack of independence using a statistical random-effects model.

Effect of ϵ on convergence rate. We plan to study the convergence behavior of the conjugate gradient procedure using approximate gradient computations. This would give us a formal mechanism to choose ϵ.

Other metrics. The paper considers only the WMW statistic, but many information retrieval metrics (for example, mean reciprocal rank, mean average precision, and normalized discounted cumulative gain) are more sophisticated. They try to weigh the items that appear at the top of the list more. In the future, we would like to extend the proposed method to other commonly used metrics.

ACKNOWLEDGMENTS

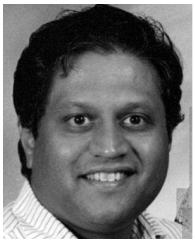
The authors would like to thank Dr. Chris Burges for his suggestions on implementing the RankNet algorithm. They would also like to thank the reviewers for their comments, which helped to improve the overall quality of the paper.

REFERENCES

[1] A. Mas-Colell, M. Whinston, and J. Green, *Microeconomic Theory.* Oxford Univ. Press, 1995.
[2] G. Fung, R. Rosales, and B. Krishnapuram, "Learning Rankings via Convex Hull Separation," *Advances in Neural Information Processing Systems 18,* Y. Weiss, B. Schölkopf, and J. Platt, eds. MIT Press, 2006.
[3] O. Dekel, C. Manning, and Y. Singer, "Log-Linear Models for Label Ranking," *Advances in Neural Information Processing Systems 16,* S. Thrun, L. Saul, and B. Schölkopf, eds. MIT Press, 2004.
[4] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bull.,* vol. 1, no. 6, pp. 80-83, Dec. 1945.
[5] H.B. Mann and D.R. Whitney, "On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other," *The Annals of Math. Statistics,* vol. 18, no. 1, pp. 50-60, 1947.
[6] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to Rank Using Gradient Descent," *Proc. 22nd Int'l Conf. Machine Learning,* 2005.
[7] Y. Freund, R. Iyer, and R. Schapire, "An Efficient Boosting Algorithm for Combining Preferences," *J. Machine Learning Research,* vol. 4, pp. 933-969, 2003.
[8] L. Greengard, "Fast Algorithms for Classical Physics," *Science,* vol. 265, no. 5174, pp. 909-914, 1994.
[9] R. Herbrich, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer, "Learning Preference Relations for Information Retrieval," *Proc. Int'l Conf. Machine Learning Workshop Learning for Text Categorization,* pp. 80-84, 1998.
[10] T. Joachims, "Optimizing Search Engines Using Clickthrough Data," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 133-142, 2002.
[11] W. Chu and Z. Ghahramani, "Preference Learning with Gaussian Processes," *Proc. 22nd Int'l Conf. Machine Learning,* pp. 137-144, 2005.
[12] R. Yan and A. Hauptmann, "Efficient Margin-Based Rank Learning Algorithms for Information Retrieval," *Proc. Int'l Conf. Image and Video Retrieval,* 2006.
[13] C. Burges, R. Ragno, and Q. Le, "Learning to Rank with Nonsmooth Cost Functions," *Advances in Neural Information Processing Systems 19,* B. Schölkopf, J. Platt, and T. Hoffman, eds. MIT Press, 2007.
[14] K. Crammer and Y. Singer, "Pranking with Ranking," *Advances in Neural Information Processing Systems,* vol. 14, pp. 641-647, 2002.
[15] E.F. Harrington, "Online Ranking/Collaborative Filtering Using the Perceptron Algorithm," *Proc. 20th Int'l Conf. Machine Learning,* 2003.
[16] R. Caruana, S. Baluja, and T. Mitchell, "Using the Future to 'Sort Out' the Present: Rankprop and Multitask Learning for Medical Risk Evaluation," *Advances in Neural Information Processing Systems,* 1995.
[17] L. Yan, R. Dodier, M. Mozer, and R. Wolniewicz, "Optimizing Classifier Performance via an Approximation to the Wilcoxon-Mann-Whitney Statistic," *Proc. 20th Int'l Conf. Machine Learning,* pp. 848-855, 2003.
[18] A. Rakotomamonjy, "Optimizing Area under the ROC Curve with SVMs," *ROC Analysis in Artificial Intelligence,* pp. 71-80, 2004.

[19] U. Brefeld and T. Scheffer, "AUC Maximizing Support Vector Learning," *Proc. ICML 2005 Workshop ROC Analysis in Machine Learning,* 2005.
[20] A. Herschtal and B. Raskutti, "Optimising Area under the ROC Curve Using Gradient Descent," *Proc. 21st Int'l Conf. Machine Learning,* 2004.
[21] R. Herbrich, T. Graepel, and K. Obermayer,, "Large Margin Rank Boundaries for Ordinal Regression," *Advances in Large Margin Classifiers,* pp. 115-132, MIT Press, 2000.
[22] J. Nocedal and S.J. Wright, *Numerical Optimization.* Springer, 1999.
[23] M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover, 1972.
[24] N.C. Beauliu, "A Simple Series for Personal Computer Computation of the Error Function $Q(.)$," *IEEE Trans. Comm.,* vol. 37, no. 9, pp. 989-991, Sept. 1989.
[25] C. Tellambura and A. Annamalai, "Efficient Computation of erfc(x) for Large Arguments," *IEEE Trans. Comm.,* vol. 48, no. 4, pp. 529-532, Apr. 2000.
[26] C. Wei and Z. Ghahramani, "Gaussian Processes for Ordinal Regression," *The J. Machine Learning Research,* vol. 6, pp. 1019-1041, 2005.
[27] A.G. Gray and A.W. Moore, "Nonparametric Density Estimation: Toward Computational Tractability," *Proc. SIAM Int'l Conf. Data Mining,* 2003.
[28] C. Yang, R. Duraiswami, and L. Davis, "Efficient Kernel Machines Using the Improved Fast Gauss Transform," *Advances in Neural Information Processing Systems 17,* L.K. Saul, Y. Weiss, and L. Bottou, eds, pp. 1561-1568, MIT Press, 2005.
[29] V.C. Raykar and R. Duraiswami, The Improved Fast Gauss Transform with Applications to Machine Learning, *Large Scale Kernel Machines,* pp. 175-201, MIT Press, 2007.

Vikas C. Raykar received the BE degree in electronics and communication engineering from the National Institute of Technology, Trichy, India, in 2001 and the MS degree in electrical engineering and the PhD degree in computer science from the University of Maryland, College Park, in 2003 and in 2007, respectively. He currently works as a scientist in Siemens Medical Solutions, Malvern, Pennsylvania. His current research interests include developing scalable algorithms for machine learning.



Ramani Duraiswami received the BTech degree in mechanical engineering from the Indian Institute of Technology, Bombay, India, in 1985 and the PhD degree in mechanical engineering and applied mathematics from the Johns Hopkins University, Baltimore, in 1991. He is currently a faculty member in the Department of Computer Science, Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, where he is the director of the Perceptual Interfaces and Reality Laboratory. His research interests are broad and currently include spatial audio, virtual environments, microphone arrays, computer vision, statistical machine learning, fast multipole methods, and integral equations. He is a member of the IEEE.

Balaji Krishnapuram received the BTech degree from the Indian Institute of Technology (IIT), Kharagpur, in 1999 and the PhD degree from Duke University in 2004, both in electrical engineering. He works as a scientist in Siemens Medical Solutions, Malvern, Pennsylvania. His research interests include statistical pattern recognition, Bayesian inference, and computational learning theory. He is also interested in applications in computer aided medical diagnosis, signal processing, computer vision, and bioinformatics. He is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.