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

IEEE Trans. On Pattern Analysis and Machine Intelligence, 2008

Dariush Hasanpoor

Isfahan University Of Technology

Dec. 2015

Outline

What is preference?

Definition

Preference Learning refers to the task of learning to predict an order relation on a collection of objects (alternatives).

- Preference information plays a key role in automated decision making and appears in various guises in AI researches:
  - Qualitative decision theory
  - Non-monotonic reasoning
  - Constraint satisfaction
  - Planning

Notations

Definition: *Weak Preference*

A weak *preference* relation $\succeq$ on a set $\mathcal{A}$ is a reflexive and transitive binary relation.

Definition: *Strict Preference*

$$a \succ b \longleftrightarrow (a \succeq b) \wedge (b \not\succeq a)$$

- In agreement with preference semantics

Notation	*Interpretation*
$a \succeq b$	"alternative a is at least as preferred as alternative b."
$a \succ b$	"alternative a is preferred over alternative b."

Notations

Definition: *Weak Preference*

A weak *preference* relation $\succeq$ on a set $\mathcal{A}$ is a reflexive and transitive binary relation.

Definition: *Strict Preference*

$$a \succ b \longleftrightarrow (a \succeq b) \wedge (b \not\succeq a)$$

- In agreement with preference semantics

Notation	*Interpretation*
$a \succeq b$	"alternative a is at least as preferred as alternative b."
$a \succ b$	"alternative a is preferred over alternative b."

Notations

Definition: *Weak Preference*

A weak *preference* relation $\succeq$ on a set $\mathcal{A}$ is a reflexive and transitive binary relation.

Definition: *Strict Preference*

$$a \succ b \longleftrightarrow (a \succeq b) \wedge (b \not\succeq a)$$

- In agreement with preference semantics

Notation	*Interpretation*
$a \succeq b$	"alternative a is at least as preferred as alternative b."
$a \succ b$	"alternative a is preferred over alternative b."

Preference Structure

Definition: *Total Strict Order (Ranking)*

If $\mathcal{A}$ is a finite set of objects/alternatives $\{a_1, \ldots, a_m\}$ a ranking of $\mathcal{A}$ can be defined with a permutation τ of $\{1, \ldots, m\}$ which $a_i \succ a_j \leftrightarrow \tau(i) < \tau(j)$.

- $\mathcal{S}_m$ is a set of all permutation of τ.
- The task of *preference learner* is to search in $\mathcal{S}_m$ space which is *learning to rank*.

Preference Structure

Definition: *Total Strict Order (Ranking)*

If $\mathcal{A}$ is a finite set of objects/alternatives $\{a_1, \ldots, a_m\}$ a ranking of $\mathcal{A}$ can be defined with a permutation τ of $\{1, \ldots, m\}$ which $a_i \succ a_j \leftrightarrow \tau(i) < \tau(j)$.

- $\mathcal{S}_m$ is a set of all permutation of τ.
- The task of *preference learner* is to search in $\mathcal{S}_m$ space which is *learning to rank*.

Types of Ranking

- The tasks are categorized as three main problems:
  - **Label ranking**
  - **Object ranking**
  - **Instance ranking**

Types of Ranking

Object Ranking

Task

The task of this model is to find a preference ranking order among instances.

Given

- A (potentially infinite) set X of objects (each object typically represented by a feature vector).
- A finite set of pairwise preferences $x_i \succ x_j$, $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$.

Find

- A ranking function that, given a set of objects $O \subset X$ as input, returns a permutation(ranking) of these objects.

- In the training phase, preference learning algorithms have access to examples for which the order relation is **(partially)** known.

Types of Ranking

Object Ranking

Task

The task of this model is to find a preference ranking order among instances.

Given

- A (potentially infinite) set X of objects (each object typically represented by a feature vector).
- A finite set of pairwise preferences $x_i \succ x_j$, $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$.

Find

- A ranking function that, given a set of objects $O \subset X$ as input, returns a permutation(ranking) of these objects.

- In the training phase, preference learning algorithms have access to examples for which the order relation is (partially) known.

Types of Ranking

Object Ranking

Task

The task of this model is to find a preference ranking order among instances.

Given

- A (potentially infinite) set X of objects (each object typically represented by a feature vector).
- A finite set of pairwise preferences $x_i \succ x_j$, $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$.

Find

- A ranking function that, given a set of objects $O \subset X$ as input, returns a permutation(ranking) of these objects.

- In the training phase, preference learning algorithms have access to examples for which the order relation is **(partially)** known.

Types of Ranking

Object Ranking

Task

The task of this model is to find a preference ranking order among instances.

Given

- A (potentially infinite) set X of objects (each object typically represented by a feature vector).
- A finite set of pairwise preferences $x_i \succ x_j$, $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$.

Find

- A ranking function that, given a set of objects $O \subset X$ as input, returns a permutation(ranking) of these objects.

- In the training phase, preference learning algorithms have access to examples for which the order relation is **(partially)** known.

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 $x \succ y$ and $y \prec x$.

- One way of describing preference relations is by means of a ranking function.

  - A function $f : \mathcal{X} \to \mathbb{R}$ 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.

Preference Relation and Ranking Function

- The learnt *ranking function* should be *rational*, a *preference* relation $\succeq$ is called rational if it satisfies the following two properties:
  - *Completeness*. $\forall x, y, \in \mathcal{X}$, we have that $x \succeq y$ or $y \succeq x$.
  - *Transitivity*. $\forall x, y, \in \mathcal{X}$, 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.

The paper considers the learning of a preference relation as a problem of learning a rational ranking function.

Problem Statement

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 $\mathcal{A} = \cup_{j=1}^{S}(\mathcal{A}^j = \{x_i^j \in \mathbb{R}^d\}_{i=1}^{m_j})$ 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 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 A^i, that is, $\forall(x_k^i \in \mathcal{A}^i, x_l^j \in \mathcal{A}^j),\ \ x_l^j \succ x_k^i$

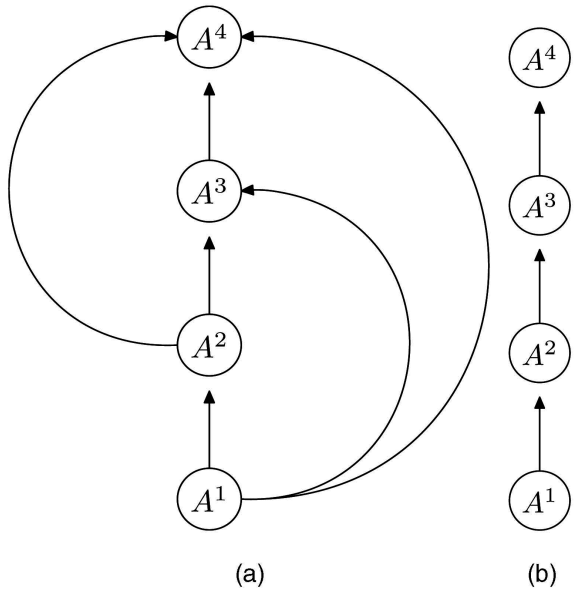


Figure 1

Problem Statement

Cont.

The Goal

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 mant 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 $\{x_k \in \mathbb{R}^d\}$.

Generalized Wilcoxon-Mann-Whitney Statistic

- The Wilcoxon-Mann-Whitney(WMW) statistic 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.

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

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

- The WMW statistic is thus an estimate of $Pr[f(x_1) \geq f(x_2)]$ for a randomly drawn pair of samples (x_1, x_0) such that $x_1 \succeq x_0$.
- For a perfect ranking function, the WMW statistic is 1, and for a completely random assignment, the expected WMW statistic is 0.5.

The MAP Estimator for Learning Ranking Functions

- The paper considered the family of linear ranking functions: $\mathcal{F} = \{f_w\}$, where for any $x, w \in \mathbb{R}^d$, $f_w(x) = w^T x$.

Objective:

Choose w in order to maximize the generalize $WMW(f_w, \mathcal{A}, \mathcal{G})$.

- For computational efficiency, the paper maximizes a continues surrogate via the log likelihood:

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

.

- In 4 we have assumed that every pair (x_l^j, x_k^i) is drawn independently, whereas only the original samples are drawn independently.

The MAP Estimator for Learning Ranking Functions

Cont.

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

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

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

$$\hat{w}_{\text{MAP}} = \underset{w}{\text{argmax}}\, L(w) \tag{6}$$

- Where $L(w)$ is the penalized log likelihood:

$$L(w) = -\frac{\lambda}{2}\|w\|^2 + \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} \log \sigma[w^T(x_l^j - x_k^i)] \tag{7}$$

The MAP Estimator for Learning Ranking Functions

Lower Bounding the WMW Statistic

- Comparing the log-likelihood $L(w)7$ to the WMW statistic (eq.2), we can lower bounding the 0-1 indicator function in the WMW statistic by a log-sigmoid function:

$$\mathbf{1}_{z>0} \geq 1 + (\frac{\log \sigma(z)}{\log_2}) \tag{8}$$

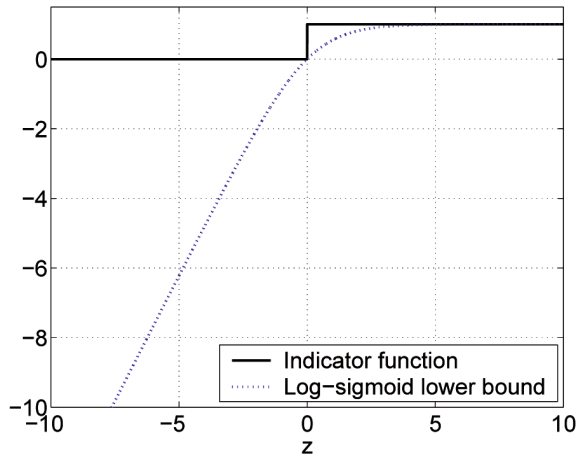


Figure 2

Therefore:

Maximizing the penalized log likelihood is equivalent to maximizing a lower bound on the WMW statistic.

The Optimization Algorithm

- **Objective:** Maximizing the penalized log likelihood, with respect to w.
- We use the Polak-Ribière variant of nonlinear conjugate gradients(CG) algorithm.
  - 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).

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

$$g(w_i) = \frac{\partial L(w)}{\partial w_i} = -\lambda w_i - \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} (x_l^i - x_k^j)\sigma[w^T(x_l^j - x_k^i)] \tag{9}$$

The Optimization Algorithm

What cost us so much!?

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

- The evaluation of the penalized log likelihood or its gradient requires $\mathcal{M}^2 = \sum_{\mathcal{E}_{ij}} m_i m_j$ operations — $\mathcal{O}(\mathcal{M}^2)$.
- The main contribution of the paper is an extremely fast method to compute the gradient *approximately*.

The Optimization Algorithm

Gradient Approximation Using the Error Function

- We shall rely on the approximation:

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

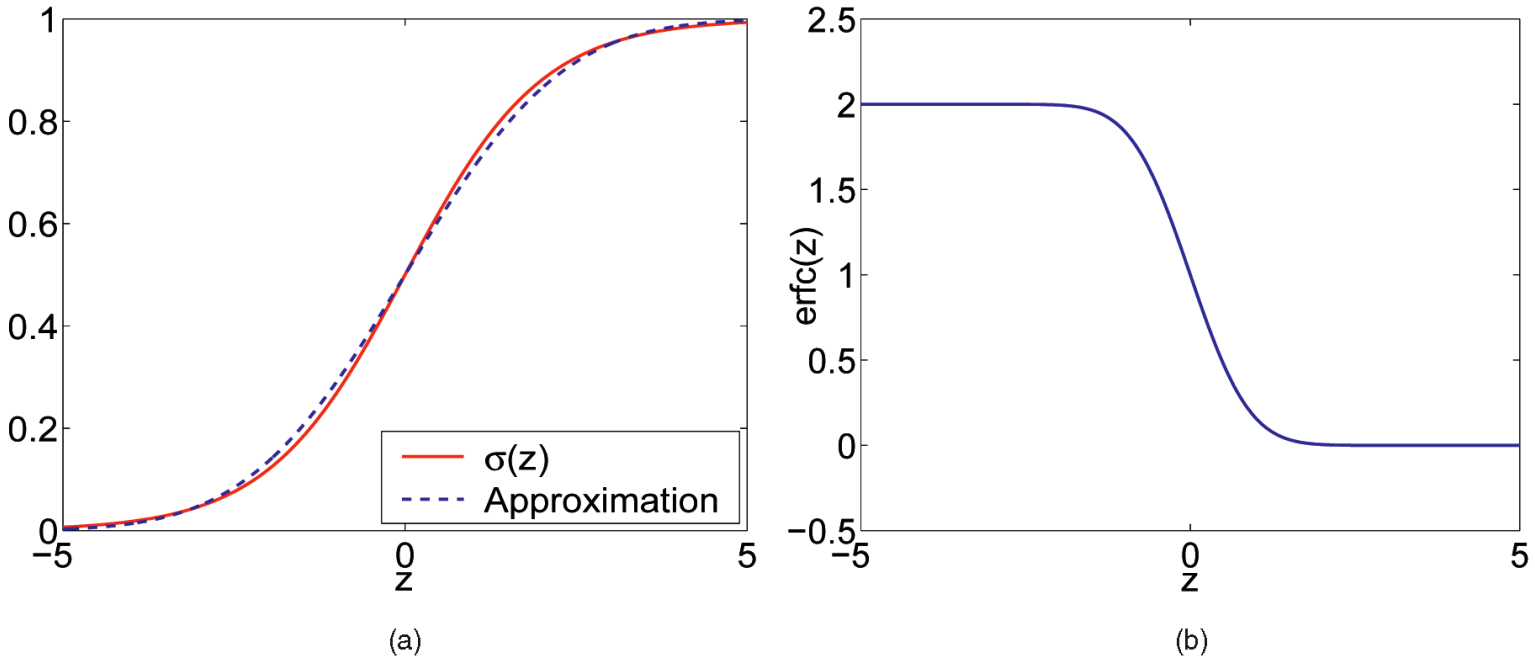


Figure 3 : (a) Approximation of the sigmoid function $\sigma(z) \approx 1 - \frac{1}{2} erfc\left(\frac{\sqrt{3}z}{\sqrt{2}\pi}\right)$. (b) The erfc function.

The Optimization Algorithm

Gradient Approximation Using the Error Function

- Where the *erfc* defined as:

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

- considering the $\sigma(z) \approx 1 - \frac{1}{2} erfc\left(\frac{\sqrt{3}z}{\sqrt{2}\pi}\right)$ the eq.9 can be approximated as:

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

The Optimization Algorithm

Quadratic Complexity of Gradient Evaluation

To summarize variables

- 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 for edges in the preference graph.
- $\mathcal{M}^2 = \sum_{\mathcal{E}_{ij}} m_i m_j$ is the total number of pairwise preference relations.
- **Definition:** $\forall x \in \mathcal{A},\ z = \frac{\sqrt{3} w^T x}{\sqrt{2}\pi}$ is scalar and for a given w can be computed in $\mathcal{O}(dm)$ operations for the entire training set.

The Optimization Algorithm

Quadratic Complexity of Gradient Evaluation(Cont.)

- We will isolate the key computational primitive contributing to the quadratic complexity in the gradient computation.

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

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

Where the *vectors* Δ_1, Δ_2 and Δ_3 are defined as follows:

$$\Delta_1 = \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} (x_k^i - x_l^j)$$

$$\Delta_2 = \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} x_k^i \, \mathit{erfc}(z_k^i - z_l^j) \tag{15}$$

$$\Delta_3 = \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} x_l^j \, \mathit{erfc}(z_k^i - z_l^j)$$

The Optimization Algorithm

Quadratic Complexity of Gradient Evaluation(Cont.)

- The vector $\Delta_1 = \sum_{\mathcal{E}_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} (x_k^i - x_l^j)$ is independent of w and can be written as follows:

$$\Delta_1 = \sum_{\mathcal{E}_{ij}} m_i m_j (x^i_{\text{mean}} - x^j_{\text{mean}})$$

  where $x^i_{\text{mean}} = \frac{1}{m_i} \sum_{k=1}^{m_i} x_k^i$ is the mean of all the training instances in the ith class.

- Δ_1 can be precomputed in $\mathcal{O}(|\mathcal{E}|d + dm)$ operations.

The Optimization Algorithm

Quadratic Complexity of Gradient Evaluation(Cont.)

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

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

where

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

- Each of Δ_2 and Δ_3 can be computed in $\mathcal{O}(dSm + \mathcal{M}^2)$ operations.

The Optimization Algorithm

Quadratic Complexity of Gradient Evaluation(Cont.)

- The core computational primitive contributing to the $\mathcal{O}(\mathcal{M}^2)$ cost is the summation of *erfc* functions.
- The paper showed that the 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 the result the Δ_2 and Δ_3 can be computed in linear $\mathcal{O}(dSm + (S-1)m)$ time.

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 centred at $z_i \in R$ with weights $q_i \in R$.

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

- Direct computation of eq.18 at M points $\{y_j \in \mathbb{R}\}_{j=1}^{M}$ is $\mathcal{O}(MN)$.

- we will derive an ϵ-accurate approximation algorithm to compute this in $\mathcal{O}(M+N)$ time.

Fast Weighted Summation Of erfc Functions

ϵ-Accurate Approximation

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

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

- The constant in $\mathcal{O}(M + N)$ for the algorithm depends on the desired accuracy ϵ, which however can be *arbitrary*.

- 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).

Fast Weighted Summation Of erfc Functions

Series Expansion for *erfc*Function

- The paper uses the following truncated Fourier series representation:

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

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

  where 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.

- This series converges rapidly, especially as $z \to 0$.

Fast Weighted Summation Of erfc Functions

Series Expansion for *erfc*Function(Cont.)

- Figure 4 shows the maximum absolute error between the actual value of *erfc*and the truncated series representation as a function of p.

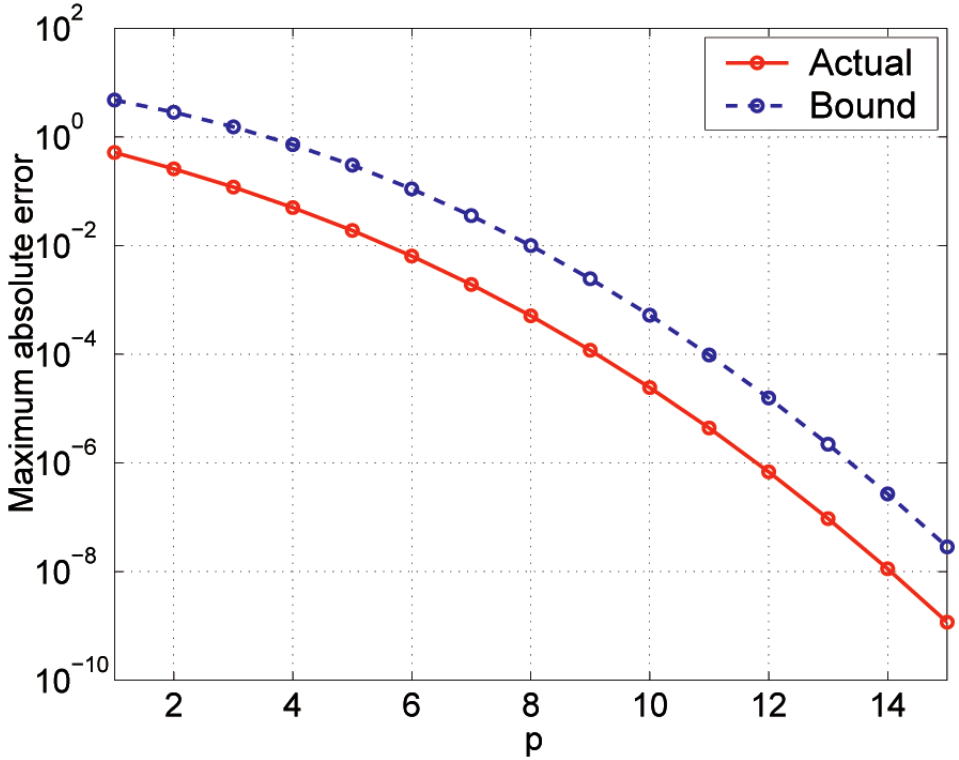


Figure 4 : The maximum absolute error between the actual value of *erfc*and the truncated series representation eq.20 as a function of the truncation number p for any $z \in [-4, 4]$. The error bound eq.**??** is also shown as adotted line

Fast Weighted Summation Of erfc Functions

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 eq. 21 as follows:

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

Fast Weighted Summation Of erfc Functions

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 eq. 21 as follows:

$$
\begin{aligned}
& \left| \frac{4}{\pi} \sum_{n=2p+1}^{\infty} \frac{e^{-n^2h^2}}{n} \sin(2nhx) \right| \\
& \leq \frac{4}{\pi} \sum_{n=2p+1}^{\infty} \frac{e^{-n^2h^2}}{n} |\sin(2nhx)| \\
\xrightarrow{|\sin(2nhx)| \leq 1} & \leq \frac{4}{\pi} \sum_{n=2p+1}^{\infty} \frac{e^{-n^2h^2}}{n} \\
\xrightarrow{\frac{1}{n} \leq 1} & < \frac{4}{\pi} \sum_{n=2p+1}^{\infty} e^{-n^2h^2} \\
\xrightarrow{\sum \to \int} & < \frac{4}{\pi} \int_{2p+1}^{\infty} e^{-x^2h^2} dx \\
& < \frac{2}{\sqrt{\pi}h} \left[\frac{2}{\sqrt{\pi}} \int_{(2p+1)h}^{\infty} e^{-t^2} dt \right] \\
& = \frac{2}{\sqrt{\pi}h} \mathit{erfc}((2p+1)h)
\end{aligned}
$$

Fast Weighted Summation Of erfc Functions

Error Bound

- Hence, the final error bound is of the form:

$$|error(z)| < \frac{2}{\sqrt{\pi}h}\, erfc((2p+1)h) + erfc(\frac{\pi}{2h} - |z|) \tag{22}$$

Fast Weighted Summation Of erfc Functions

Fast Summation Algorithm

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

$$E(y) = \sum_{i=1} N q_i \, \mathit{erfc}(y - z_i)$$

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

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

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

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.

Fast Weighted Summation Of erfc Functions

Fast Summation Algorithm

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

$$\hat{E}(y) = Q - \frac{4}{\pi} \sum_{i=1}^{N} q_i \sum_{\substack{n=1 \\ \text{n odd}}}^{2p-1} \frac{e^{-n^2 h^2}}{n} \sin\{2nh(y - z_i)\} \tag{25}$$

  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:

$$\begin{aligned} &\sin\{2nh(y - z_i)\} \\ &= \sin\{2nh(y - z_*) - 2nh(z_i - z_*)\} \\ &= \sin\{2nh(y - z_*)\} \cos\{2nh(z_i - z_*)\} - \cos\{2nh(y - z_*)\} \sin\{2nh(z_i - z_*)\} \end{aligned} \tag{26}$$

Fast Weighted Summation Of erfc Functions

Fast Summation Algorithm

$$\hat{E}(y) = Q - \frac{4}{\pi}\sum_{i=1}^{N} q_i \sum_{\substack{n=1 \\ \text{n odd}}}^{2p-1} \frac{e^{-n^2h^2}}{n} * \Big[\sin\{2nh(y-z_*)\}\cos\{2nh(z_i-z_*)\} - \cos\{2nh(y-z_*)\}\sin\{2nh(z_i-z_*)\}\Big] \tag{27}$$

- Note that we have shifted all the points by z_*.
  - The reason for this is that we cluster the points and use the series representation around the different cluster centres.

Fast Weighted Summation Of erfc Functions

Fast Summation Algorithm

- Substituting the separated representation and exchanging the order of summation and regrouping the terms, we have the following expression:

$$\begin{aligned}\hat{E}(y) = Q - \frac{4}{\pi} \sum_{\substack{n=1 \\ \text{n odd}}}^{2p-1} A_n \sin\{2nh(y - z_*)\} \\ + \frac{4}{\pi} \sum_{\substack{n=1 \\ \text{n odd}}}^{2p-1} B_n \cos\{2nh(y - z_*)\}\end{aligned} \tag{28}$$

$$\begin{aligned}A_n = \frac{e^{-n^2h^2}}{n} \sum_{i=1}^{N} q_i \cos\{2nh(z_i - z_*)\} \\ B_n = \frac{e^{-n^2h^2}}{n} \sum_{i=1}^{N} q_i \sin\{2nh(z_i - z_*)\}\end{aligned} \tag{29}$$

Fast Weighted Summation Of erfc Functions

Computational and Space Complexity

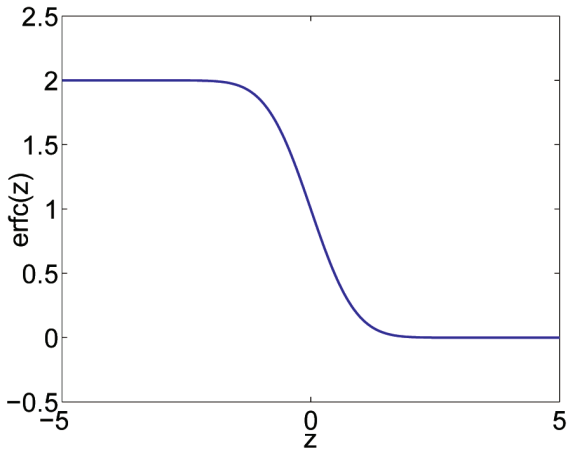
- The coefficients $\{A_n, B_n\}$ do not depend on y. Hence, each of A_n and B_n can be evaluated separately in $\mathcal{O}(N)$ times.
- Since there are p such coefficients the total complexity to computed A and B is $\mathcal{O}(pN)$.
- The term $Q = \sum_{i=1}^{N} q_i$ can also be pre-computed in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

Fast Weighted Summation Of erfc Functions

Direct Inclusion and Exclusion of Far Away Points

$$|error(z)| < \frac{2}{\sqrt{\pi}h} \mathit{erfc}((2p+1)h) + \mathit{erfc}(\frac{\pi}{2h} - |z|)$$

- For a fixed p and h as $|z|$ increases the error increases. Therefor as $|z|$ increases, h should decreases, 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 $\mathit{erfc}(s) \to 2$ as $s \to -\infty$ and $\mathit{erfc}(s) \to 0$ as $s \to \infty$ very quickly.



(b)

Fast Weighted Summation Of erfc Functions

Direct Inclusion and Exclusion of Far Away Points

$$|error(z)| < \frac{2}{\sqrt{\pi}h}\, erfc((2p+1)h) + erfc(\frac{\pi}{2h} - |z|)$$

- For a fixed p and h as $|z|$ increases the error increases. Therefor as $|z|$ increases, h should decreases, 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 $erfc(s) \to 2$ as $s \to -\infty$ and $erfc(s) \to 0$ as $s \to \infty$ very quickly.
- We only want an accuracy of ϵ, we can use the approximation:

$$erfc(c) \approx \begin{cases} 2 & s < -r \\ \text{p-truncated series} & -r \le s \le r \\ 0 & s > r \end{cases} \tag{30}$$

The bound r and the truncation number p have to be chosen such that for any s, the error is always less than ϵ.

Fast Weighted Summation Of erfc Functions

Choosing the Parameters

$$h = \frac{\pi}{3(r + \mathit{erfc}^{-1}(\frac{\epsilon}{2}))} \tag{31}$$

$$p = \left\lceil \frac{1}{2h} \mathit{erfc}^{-1}\left(\frac{\sqrt{\pi} h \epsilon}{4}\right) \right\rceil \tag{32}$$

$$r = \mathit{erfc}^{-1}(\epsilon) + 2r_x \tag{33}$$

Ranking Experiments

Datasets

	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

Table 1 : N is the size of the data set. d is the number of attributes. S is the number of classes. M is the average total number of pairwise relations per fold of the training set.

Ranking Experiments

Results

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

	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]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	62.91 [± 0.59]
12	6731.09 [± 312.41]	19.10 [± 1.76]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	147.04 [± 0.16]
13	2556.93 [± 15.03]	3.59 [± 0.41]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	133.42 [± 1.14]
14	$\star$ [± $\star$]	46.86 [± 1.06]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]

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

Ranking Experiments

Results

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

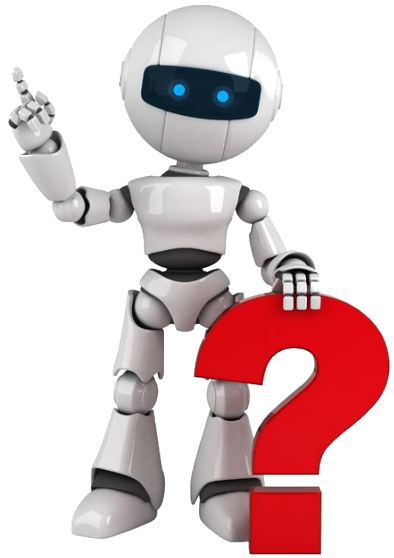
	RankNCG direct	**RankNCG fast**	RankNet linear	RankNet two layer	RankSVM linear	RankSVM quadratic	RankBoost
1	0.677 [± 0.233]	0.650 [± 0.210]	0.579 [± 0.096]	0.479 [± 0.284]	0.545 [± 0.236]	0.400 [± 0.276]	0.675 [± 0.173]
2	0.987 [± 0.019]	0.948 [± 0.077]	0.872 [± 0.088]	0.968 [± 0.038]	0.973 [± 0.048]	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]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	0.535 [± 0.014]
12	0.917 [± 0.005]	0.917 [± 0.005]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	0.845 [± 0.006]
13	0.623 [± 0.008]	0.623 [± 0.008]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	0.607 [± 0.010]
14	$\star$ [± $\star$]	0.979 [± 0.001]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]	$\star$ [± $\star$]

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

Conclusion

- This paper, 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.
- Experimental results demonstrate that despite the order of magnitude speedup, the accuracy was almost identical to exact method and other algorithms proposed in literature.

Thank You!



References

V. C. Raykar, R. Duraiswami, and B. Krishnapuram, "A fast algorithm for learning a ranking function from large-scale data sets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 7, pp. 1158–1170, 2008.

J. Furnkranz and E. Hullermeier, *Encyclopedia of Machine Learning*. Springer, 2010, ch. Preference Learning, pp. 789–795.

Johannes Furnkranz and Eyke Hullermeier, "Preference Learning," *Kunstliche Intelligenz*, pp. 60–61, 2005.

E. Hullermeier, J. Furnkranz, W. Cheng, and K. Brinker, *Artificial Intelligence*. ScienceDirect, 2008, ch. Label ranking by learning pairwise preferences.