An Empirical Comparison of Learning Algorithms for Nonparametric Scoring

Marine Depecker, Stéphan Clémençon, Nicolas Vayatis

Title Page

DRAFT manuscript No.
(will be inserted by the editor)

An Empirical Comparison of Learning Algorithms for Nonparametric Scoring

The TreeRank Algorithm and Other Methods

Stéphan Clémençon · Marine Depecker · Nicolas Vayatis

the date of receipt and acceptance should be inserted later

Stéphan Clémençon · Marine Depecker
Télécom ParisTech
LTCI - UMR CNRS 5141
Paris, France
Tel.: +33-1-45817807
Fax.: +33-1-45817158
E-mail: stephan.clemencon@telecom-paristech.fr

Nicolas Vayatis
ENS Cachan & UniverSud
CMLA - UMR CNRS 8536 Cachan, France

DRAFT manuscript No.
(will be inserted by the editor)

An Empirical Comparison of Learning Algorithms for Nonparametric Scoring

The TreeRank Algorithm and Other Methods

the date of receipt and acceptance should be inserted later

Abstract The TreeRank algorithm was recently proposed in [1] and [2] as a scoring-based method based on recursive partitioning of the input space. This tree induction algorithm builds orderings by recursively optimizing the Receiver Operating Characteristic (ROC) curve through a one-step optimization procedure called LeafRank. One of the aim of this paper is the in-depth analysis of the empirical performance of the variants of TreeRank/LeafRank method. Numerical experiments based on both artificial and real data sets are provided. Further experiments using resampling and randomization, in the spirit of bagging and random forests are developed ([3], [4]) and we show how they increase both stability and accuracy in bipartite ranking. Moreover, an empirical comparison with other efficient scoring algorithms such as RankBoost and RankSVM is presented on UCI benchmark data sets.

Keywords Scoring rules · Ranking trees · ROC curve · AUC Maximization · Resampling · Feature randomization

1 Introduction

In the last decade, learning to rank signals, web pages, individuals, organizations, has affected critical aspects of modern society. But behind the very concept of 'ranking', very diverse setups and numerous types of data may be considered. One of the subproblems in the ranking literature which has drawn a lot of attention is known as ranking with bipartite feedback or bipartite ranking [5]. In short, bipartite ranking can be cast as the problem of ordering instances x of a possibly high dimensional space $\mathcal{X}$ based on a binary feedback information $y \in \{-1, +1\}$ attached to every instance x. The type of applications we have in mind are those related to scoring high dimensional observation vectors with binary labels, such as medical diagnosis or credit-risk screening. Several methods and algorithms can be applied to produce scoring rules for bipartite ranking: plain (or Kernel) Logistic Regression [6], [7], LogitBoost [8], RankBoost (see [5]), RankSVM [9], RankLS [10], RankNet (see [11]), and many others. In the present paper, we will mainly consider a promising tree-based procedure called TreeRank recently developed and described in [1], [2]. We will explain how it differs from other algorithms in its very principles and explore its performance by providing an intensive experimental study, jointly with a fair comparison with other methods available in open source programming environments.

The guide for this study is the type of metrics used for performance assessment. In signal detection or medical diagnosis, the most complete description of the performance of a real-valued scoring rule is the ROC curve which plots the true positive rate as a function of the false positive rate as the detection threshold varies. However, as a function-like criterion, it only provides a partial order on the class of all possible scoring rules and it is quite difficult to optimize in practice. The mostly used reference performance measures are summaries of the ROC curve and the most popular certainly is the Area Under the ROC Curve, or AUC. The drawback of this type of measure lies in the fact that it is a global measure and does not take into account the errors among the top of the list (highest scores). Various approaches for providing local measures of scoring performance related to the ROC curve have been proposed [12], [13], [14] or [15] and such metrics will be considered for assessing performance. There are numerous different criteria such as the Discounted Cumulative Gain, the Hit Ratio@x%, the Average Precision@n, and their variants (we refer to [16] for an extensive enumeration and discussion regarding all these criteria). Interestingly, the main feature of modern learning algorithms consists of performance maximization (or risk minimization, equivalently) and one expects that the closer the optimization risk functional is to the actual target risk functional, the better the performance evaluated on test data sets. This observation is the foundation for many machine learning algorithms extending the principle of convex risk minimization developed in the classification setup to the case of data conceived as pairwise comparisons [17]. The answer to the standard questions of regularized learning methods (choice of cost functions, penalty or dynamics - boosting or SVM-type) then determines specific ranking algorithms. For instance, RankBoost implements AUC maximization in the spirit of AdaBoost, RankSVM proposes to fit an SVM algorithm with the hinge loss, RankLS uses the least-squares as a cost function, and RankNet follows the spirit of Neural Networks with an entropy-based cost function. The common feature of these algorithms is that they rely on pairwise comparisons (preferences) between labels and scoring functions. We emphasize the fact that the principle of performance maximization actually differs from the classical statistical methods used in scoring applications. Indeed, it appears that the mainstream scoring method of logistic regression is based on the conditional maximum likelihood maximization [6] and aims at modeling the unknown posterior probability

$$\eta(x) = \mathbb{P}\{Y = +1 \mid X = x\} \ .$$

This makes sense since the function η actually defines the optimal scoring rule in the sense of the ROC curve [1]. Besides, statistical learning theory has established connections between convex risk minimization with classification - calibrated costs and logistic regression ([8]) and this idea has been further developed in kernel logistic regression algorithms ([7] and references therein). In these algorithms, the parametric estimation step of logistic regression is replaced by nonparametric estimation where an SVM-type convex optimization problem is solved. Eventually, an estimate of the function η is provided by plugging-in the solution of the optimization step in a closed-form expression for η. Such methods seem to offer reasonable competitors to RankBoost. However, from the theoretical side, they present serious drawbacks. Indeed, in case that the statistical model used to represent the regression function is wrong, there are no guarantees about the stability of the estimated scoring rule. Such *plug-in* methods suffer from the curse of dimensionality and convex risk minimization methods are known to fail in the estimation of the regression function ([18],[19]). Hence, methods based on performance, such as RankBoost offers more guarantees than statistical estimation methods in leading to efficient ranking rules as their principle relies on the very optimization of one of the objective performance metrics for bipartite ranking. On the other hand scoring-based methods provide a richer output as they give a finer local description of the 'ranking'. Indeed, the limitation of RankBoost lies in the fact that AUC is a global criterion. Therefore there is no reason why methods like RankBoost should perform well on, say, the top 10% of instances in $\mathcal{X}$.

In the present paper, we propose to explore the empirical performance of TreeRank and a series of its variants (alternative splitting rule called LeafRank, and Ranking Forests, see [1], [2], [4]). The interesting feature of the TreeRank algorithm is that it takes the best of the two approaches previously described. Indeed, the TreeRank procedure uses decision trees to build a scoring rule which estimates the optimal ordering as induced by the regression function η, while maximizing the *Area Under the* ROC *Curve* (AUC) in a recursive manner. The one-step AUC maximization is implemented with a splitting rule called LeafRank which can be seen as an algorithm resolving a cost-sensitive classification problem with data-dependent cost (see [2]). The reason why this method can potentially perform better than RankBoost is that TreeRank/LeafRank method optimizes the ROC curve in a stronger sense than the AUC (in sup-norm namely, see [1]). It also avoids the drawbacks of statistical estimation methods as it does not aim at the direct estimation of the regression function (only the ordering induced matters here, and not the actual values of the scoring rule).

The structure of the paper is the following. In Section 2, the setup and the notations are introduced. Section 3 presents the ranking algorithms under study in this paper, including TreeRank/LeafRank, Ranking Forests, and competitors such as RankBoost, RankLS, or RankSVM. The main results can be found in the numerical experiments described in Section 4. The conclusions are gathered in the discussion of Section 5.

2 The scoring problem in presence of classification data

We first introduce concepts from probabilistic modeling which describe the nature of the bipartite ranking problem. We shall refer here to the optimal elements for bipartite ranking and to the performance measures that can be retained to capture the optimal decision rules.

2.1 Setup

Probabilistic model. The probabilistic setup for bipartite ranking is the same as for binary classification. The data appear as a sample $\mathcal{D}_n = \{(X_i, Y_i) : 1 \leq i \leq n\}$, and we assume that the (X_i, Y_i)'s are *i.i.d.* realizations of a random pair (X, Y), where X lies in a feature space $\mathcal{X}$ and Y is a binary label in $\{-1, +1\}$. The joint probability distribution of the pair (X, Y) can be described, for instance, by P_X the marginal distribution of X and the regression function $\eta(x) = \mathbb{P}(Y = +1 \mid X = x)$, $\forall x \in \mathcal{X}$. We also use the notation $p = \mathbb{P}(Y = +1)$.

Optimal scoring rules. It is well-known that the target of binary classification is the level set $R^* = \{x \in \mathcal{X} \;:\; \eta(x) > 1/2\}$ ([20]). Indeed, predicting $y = +1$ on any $x \in R^*$ and $y = -1$ otherwise is the best possible decision in this case. Now, if the problem is how to order the instances of $\mathcal{X}$ in such a way that positively labeled data appear on top of the list with high probability, then the optimal decision rules consist of real-valued scoring functions which imitate the ordering induced on $\mathcal{X}$ by the regression function η. In previous works ([1],[21]), we have showed that the following statements are true:

- *The class of optimal scoring functions are strictly increasing transforms of the regression function.*
  They are of the form: $s^* = T \circ \eta$, for some measurable function $T : \mathbb{R} \to \mathbb{R}$, whose restriction to the range of $\eta(X)$ is strictly increasing.
- *Optimal scoring rules can be obtained from the entire collection of level sets of the regression function η.*
  Indeed, they can be represented as: $\forall x \in \mathcal{X}$, $s^*(x) = c + \int_0^1 w(u) \cdot \mathbb{I}\{\eta(x) > u\}\, du$, where $c \in \mathbb{R}$, $w > 0$.
- *Bipartite ranking may be viewed as a continuum of classification problems.*
  This fact follows from the representation of s^*. Finding optimal scoring rules boils down to recovering the whole class of level sets: $\{\{\eta(x) > u\},\ u \in [0, 1]\}$.
- *The* ROC *curve fully describes the performance of scoring functions in the bipartite ranking setup.*
  As a matter of fact, for a given distribution P_X, optimal scoring rules s^* have an ROC curve above the ROC curve of any other scoring rule s. An easy bipartite ranking problem corresponds to the case where class-conditional distributions (distribution of X given the class label Y) are far from each other. On the opposite, in the case when there is an important overlap between the two class-conditional distributions, the optimal ROC curve becomes close to the diagonal line.

2.2 Performance measures for scoring

Most of machine learning algorithms involve a functional optimization step. One of the main contributions of statistical learning theory has been the understanding of the connection between the optimization performed and the performance metrics for assessing the quality of prediction ([17]). The choice of an objective criterion determines the optimal elements for a given problem and allows to have a baseline to compare prediction results for various methods. In the case of binary classification, considering that performance of a classifier $g : \mathcal{X} \to \{-1, +1\}$ is measured with the misclassification rate $L(g) = \mathbb{P}\big(Y \neq g(X)\big)$ implies that the optimal classifier is the indicator of the level set R^*. Now, if the criterion puts asymmetric weights on the two types of error

$$\begin{aligned} L_w(g) &= 2w\mathbb{P}\big(Y = +1, g(X) = -1\big) \\ &\quad + 2(1-w)\mathbb{P}\big(Y = -1, g(X) = +1\big) \ , \end{aligned}$$

then the optimal classifier is the indicator of a level set of the regression function whose level is equal to w. In addition, most of the efficient algorithmic approaches, such as boosting or Support Vector Machines, actually optimize surrogate (convex) risk functionals. Under rather weak assumptions, optimizing the surrogate criterion actually optimizes the misclassification rate and proving this fact was definitely one of the major achievements in the recent years (see [22]).

The ROC *curve and the* AUC. When it comes to scoring applications, the golden standard definitely is the ROC curve ([23]) as it represents the discrimination ability of a scoring rule at all levels. The ROC curve of a scoring rule s plots the true positive rate $\beta_s(t) = \mathbb{P}\big(s(X) > t \mid Y = +1\big)$ against the false positive rate $\alpha_s(t) = \mathbb{P}\big(s(X) > t \mid Y = -1\big)$ as the threshold t varies from $-\infty$ to $+\infty$. However, the ROC curve is a complex object to optimize as it represents a function-like criterion. It mainly serves as a visual display of performance used *a posteriori* (i.e. after training the scoring rule). A simpler object is the functional defined by the Area Under the ROC Curve, known as the AUC which can be interpreted as the probability of concordant pairs:

$$\begin{aligned} \text{AUC(s)} &= \mathbb{P}\big(s(X) > s(X') \mid Y = 1, Y' = -1\big) \\ &\quad + \frac{1}{2}\mathbb{P}\big(s(X) = s(X') \mid Y = 1, Y' = -1\big), \end{aligned}$$

where (X, Y) and (X', Y') are i.i.d. pairs of observations.

Remark 1 In the case of preference-based ranking, then individual labels are not available and there exists no ROC curve for a preference function. However, it is possible to extend the definition of AUC to this setup. Indeed, denote preference data by triples of the form (X, X', Z) where $X, X' \in \mathcal{X}$, and $Z \in \{-1, 0, +1\}$. Then a preference function $\pi : \mathcal{X} \times \mathcal{X} \to \{-1, 0, +1\}$ has an AUC defined by

$$\begin{aligned} \text{AUC}(\pi) &= \mathbb{P}\big(\pi(X, X') > 0 \mid Z = +1\big) \\ &\quad + \frac{1}{2}\mathbb{P}\big(\pi(X, X') > 0 \mid Z = 0\big) \ . \end{aligned}$$

Note that a scoring rule s leads to the obvious choice for a preference function : $\pi(x, x') = s(x) - s(x')$ (see [24] for details and optimal elements for preference data). The converse is less straightforward (see for example [25]).

An important limitation of the AUC is that it only measures the quality of a scoring / ordering in a global manner. Two scoring rules may present the same AUC and simultaneously have a very different behavior. Indeed, the relative weight of a discordant pair (x, x') is only a function of the difference $s(x) - s(x')$ of the scores between the two instances and not of their position over the range of s. This is a crucial remark, especially as in most applications involving ranking problems people often focus on the instances with highest scores (corresponding to the part of the ROC curve close to the origin). As highlighted in [13], page ranking, credit risk screening or medical diagnosis are good examples of fields in which priority is given to observations among the top scores. Indeed, in credit risk screening for instance, the main objective will be to identify potential risks among the best candidates for a credit. In a similar way, main objective in medical diagnosis will be to ensure correct diagnosis among the more risky patients of a cohort. There exist several criteria which emphasize the highest scores : the Average Precision (AP), the Predicted-Rank-of-Top (PROT), the Coverage, the Discounted Cumulative Gain (DCG), the p-norm push ([15]). Most of these criteria can be related to linear rank statistics as shown in [13] and [14], but they are not necessarily related to specific algorithms maximizing them.

Local AUC. In [13], a truncation of the AUC called the local AUC has been introduced which examines performance in the sense of the AUC but only at the top of the ranked list. The truncation is shown to be consistent with the bipartite ranking problem as the regression function η or any strictly increasing transform of it is still optimal. Fix $u \in (0,1)$ the rate of best instances and let $t^* = t(s,u)$ be the corresponding threshold: $u = \mathbb{P}\big(s(X) > t^*\big)$. We take the notations: $\alpha^* = \alpha_s(t^*)$ and $\beta^* = \beta_s(t^*)$, which are determined as the coordinates of the intersection of the *control line* $d_u : u = p\beta + (1-p)\alpha$ in the ROC space with coordinates (α, β) with the ROC curve (see Figure 1). Then, the Local AUC is defined by:

$$\mathrm{LocAUC(s,u)} = \beta^*(1-\alpha^*) + \int_{t=0}^{\alpha^*} \beta_s \circ \alpha_s^{-1}(t)\ dt,$$

where $\alpha_s^{-1}(t) = \inf\{u \in (0,1) : \ \alpha_s(u) \geq t\}$ for all $t \in (0,1)$. Instead of putting more weight on top-ranked instances, the local AUC only focuses on the top of the list with a truncation.

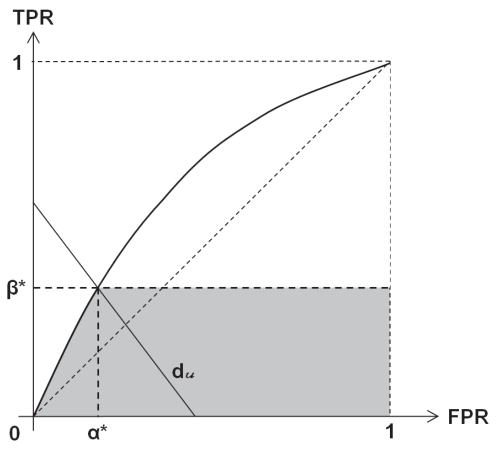


Fig. 1 The ROC curve, the control line and the LocAUC criterion.

Other measures. The tree-based approach considered in this paper focuses on the ROC curve and the AUC. In the experimental section, we will also consider other criteria for completeness. We shall provide the Average Precision as well as the Hit Rate.

3 TreeRank and other scoring methods

Our focus in the paper lies in the scoring-based method proposed in [1] and [2], using recursive partitioning in the spirit of decision trees. In the present section, we introduce the main ideas and the algorithms in this approach, as well as some of its competitors.

3.1 Presentation of TreeRank and its variants

Ranking trees. Scoring rules output by TreeRank/LeafRank can be described as oriented binary trees, called ranking trees. Ranking trees are defined by two elements: (i) a partition of the input space $\mathcal{X}$ into cells, and (ii) a permutation on the cells which indicates the ordering induced by the rule. Denote by $\mathcal{T}_D$ the binary tree of depth $D \geq 0$ which represents a recursive partition of the input space $\mathcal{X}$. Let us equip the tree with a left-to-right orientation. We then obtain a collection of scoring rules stored in a tree-like structure (see Figure 2). The root of the tree is $C_{0,0} = \mathcal{X}$, and each internal node $C_{d,k}$, with $0 \leq d < D$ and $0 \leq k < 2^d$, corresponds to a cell in $\mathcal{X}$ and splits into two non-empty cells $C_{d+1,2k}$ (left sibling) and $C_{d+1,2k+1} = C_{d,k} \setminus C_{d+1,2k}$ (right sibling). Any subtree $\mathcal{T} \subset \mathcal{T}_D$ can then be used to define a pre-order on $\mathcal{X}$: all elements in the same terminal cell will be considered as ties, and the ordering is induced by the left-to-right orientation of the subtree. The scoring rule is then defined as:

$$s_{\mathcal{T}}(x) = \sum_{C_{d,k}:\text{terminal cell}} 2^D(1-k/2^d)\cdot\mathbb{I}\{x \in C_{d,k}\}.$$

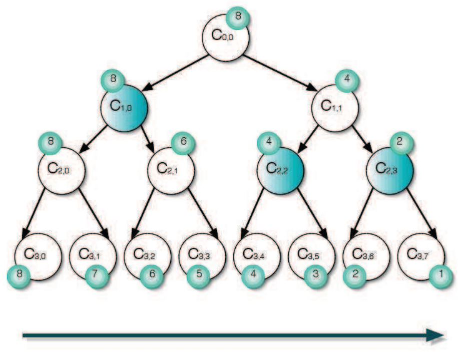


Fig. 2 Ranking tree.

Recursive partitioning for ranking. The TreeRank algorithm has been recently introduced in [1] and its theoretical properties have been discussed there in detail. We now recall the principle of this method. The algorithm follows the principle of CART proposed in [26], excepts that it builds an oriented binary tree and uses a splitting rule which implements AUC maximization at the node level (optimization step). Consider $(X_1, Y_1), \ldots, (X_n, Y_n)$, a sample of classification data. One of the findings of the analysis is that the optimization step corresponds to solving a cost-sensitive classification problem. At a given node, the cell C is split into two parts C_+ and $C_- = C \setminus C_+$ so that the weighted empirical error

$$\widehat{L}_{C,\omega}(\Gamma) = \frac{2(1-\omega)}{n} \sum_{i=1}^{n} \mathbb{I}\{X_i \in C \setminus \Gamma\}\mathbb{I}\{Y_i = 1\} + \frac{2\omega}{n} \sum_{i=1}^{n} \mathbb{I}\{X_i \in \Gamma \cap C\}\mathbb{I}\{Y_i = -1\} \; .$$

is minimized at $\Gamma = C_+$, for an adaptively chosen value of the weight ω. Then the TreeRank algorithm for the recursive optimization of the ROC curve can be formulated as follows. We point out that the class of candidate sets Γ is determined by the choice of a splitting rule.

```
THE TREERANK ALGORITHM

• Initialization. Start with $C_{0,0} = \mathcal{X}$.

• Itérations. For $d = 0, \ldots, D-1$ and $k = 0, \ldots, 2^d - 1$ ,
    1. Set $n_{d,k} = \sum_{i=1}^{n} \mathbb{I}\{X_i \in C_{d,k}\}$. Compute
       the rate of false positives in cell $C_{d,k}$:
       $\alpha(C_{d,k}) = \frac{1}{n_{d,k}} \cdot \sum_{i=1}^{n} \mathbb{I}\{X_i \in C_{d,k}\} \cdot \mathbb{I}\{Y_i = +1\}$.
    2. Minimize $\widehat{L}_{C_{d,k},\omega}(\Gamma)$ with
       $\omega = \omega_{d,k} = \alpha(\mathcal{C}_{d,k})$ .
    3. Set $C_{d+1,2k} = C_+$ the minimizer, and
       $C_{d+1,2k+1} = C_{d,k} \setminus C_{d+1,2k}$.

• Output. Oriented binary tree
    $\mathcal{T}_D = \{C_{d,k} : 0 \le d \le D, \; 0 \le k < 2^d\}$.
```

Splitting rules. The choice of a splitting rule relies on the the trade-off between efficiency and computability. In plain decision trees, simple rules are retained such as orthogonal splits. Here the task being more involved, more complex splitting rules may be considered. In principle, any classification algorithm with an easy tuning of asymmetric costs can do the job. In ([2]), a specific splitting rule is proposed, called LeafRank which uses the TreeRank algorithm at the node level recursively to build a partition and then splits the cells of this partition into two categories: one collection of cells going to the left sibling node, and the rest going to the right sibling node. The classification of cells relies on a result which states that, for a fixed partition, the optimal permutation in the sense of the AUC is the one corresponding to a monotone ordering of the ratio of true positive rate over false positive rate per cell. The advantage of LeafRank is that it uses orthogonal splits and hence preserves the interpretability of the scoring rule. It is possible indeed to recover measures of importance of the variables and of dependence between them (see [2]). An alternative to LeafRank is to use a cost-sensitive version of a SVM classifier (see [27]), which increases the complexity of the scoring rule but at the same time, interpretability of features is lost.

Pruning. Decision trees are known to suffer from overfitting as you can end up with one observation per cell. Beyond using arbitrary stopping rules, more can be done with pruning. Pruning consists in removing subtrees from the complete tree grown with a given splitting rule. This can be done in several ways and we consider here the simple option of taking a linear penalty with a constant calibrated through cross validation. The penalized criterion is the following:

$$\widehat{\mathrm{AUC}}(s_\tau) + \lambda \cdot \#\mathcal{P}(\tau) \; ,$$

where τ is a pruned tree, s_τ is the corresponding scoring function, $\widehat{\mathrm{AUC}}(s)$ is the empirical counterpart of the AUC, $\#\mathcal{P}(\tau)$ is the cardinal of the partition induced by τ on the input space $\mathcal{X}$, and λ is a constant to be estimated through cross validation.

Resampling and randomization. Aggregation procedures in machine learning are known to improve both performance and robustness of decision rules. This principle underlies numerous algorithms derived from bagging and boosting methods. Using the idea of resampling and aggregation, a collection of scoring rules can be built from many bootstrap samples in the spirit of bagging methods. By adding the ingredient of randomization an analogue to random forests called Ranking Forests can be derived (a presentation is given in [4]). For completeness, its description is provided in the box below.

The force of TreeRank. The strengths of the TreeRank algorithm are numerous. First, it possesses the main features of tree-based decision rules from the algorithmic point of view. It is also straightforward to read the contribution of latent variables when using decision trees as splitting rules at each node. One ranking tree by itself also suffers the same drawback as CART, and that is instability. Variants based on aggregation of many ranking trees, such as bagging and ranking forests, lead to more robust methods and improve performance significantly as shown in the experimental section. Another important feature of TreeRank lies in its theoretical soundness. Indeed, it can be shown (see [1]) that the scoring rule output by TreeRank produces near-optimal ROC curves in the sup-norm sense.

3.2 Other learning methods for ranking and scoring classification data

The optimization principles underlying several algorithms used in bipartite ranking are of two types: those which focus on the optimization of the target performance measure and those which aim at recovering the target regression function $\eta(x) = \mathbb{P}\{Y = +1 \mid X = x\}$, $\forall x \in \mathcal{X}$. In the first category belong methods with an optimization based on a pairwise criterion. This is indeed the case with AUC maximization methods such as RankBoost, RankSVM, RankLS. Other algorithms like RankNet optimize a nonconvex criterion but they also use pairwise cost functions. Among the methods which focus on the optimal function to recover, we consider classification methods which output a real-valued scoring function such as Logistic Regression, AdaBoost and LogitBoost, or Kernel Logistic Regression among others.

RankBoost ([5]). This method mimics AdaBoost in the pairwise setup. It boils down to the optimization of the following criterion

$$\frac{2}{n(n-1)}\sum_{i<j}\exp\{(Y_i - Y_j)(f(X_i) - f(X_j))\}\ ,$$

f being a linear combination of weak ranking rules. Weak ranking rules involved are decision stumps.
The functional above is a proxy to the empirical AUC.

RankSVM and RankLS (resp. [9] and [10]). These two methods are export the SVM methodology to set a pairwise criterion of the form

$$\frac{2}{n(n-1)}\sum_{i<j} d\big(Y_i - Y_j, f(X_i) - f(X_j)\big) + \lambda\|f\|_K,$$

where $f(x) = \sum_{i=1}^n \alpha_i K(x, X_i)$, λ is a smoothing parameter and $\|f\|_K$ is the RKHS norm of f related to the kernel K.
In the case of RankSVM the loss function d is the hinge loss $d(u,v) = (1 - uv)_+$, while for RankLS, it corresponds to the least-squares criterion $d(u,v) = (u - v)^2$.

RankNet ([11]). Here the cost function used is based on cross entropy between the target value of the probability of a pair and the estimated probability

$$\begin{aligned} d\big(Y_i - Y_j, f(X_i) - f(X_j)\big) &= -\Delta_{i,j}\log \pi_{i,j} \\ &\quad -(1-\Delta_{i,j})\log(1 - \pi_{i,j}), \end{aligned}$$

where

$$\Delta_{i,j} = \frac{1 + (Y_i - Y_j)/2}{2} \in \{0, 1/2, 1\}\ ,\ \text{and}$$

$$\pi_{i,j} = \pi(f(X_i) - f(X_j)) = \frac{e^{f(X_i) - f(X_j)}}{1 + e^{f(X_i) - f(X_j)}}\ .$$

and the candidate functions f have the same structure as neural nets.
Note that the cost function can also be written as:

$$\begin{aligned} d\big(Y_i - Y_j, f(X_i) - f(X_j)\big) &= \\ \log\left(1 + e^{f(X_i) - f(X_j)}\right) - \Delta_{i,j}\ &\big(f(X_i) - f(X_j)\big). \end{aligned}$$

LogitBoost ([8]) and Kernel Logistic Regression ([7]).
These two methods do not involve pairs of observations and therefore belong to classification methods, except that they output a real-valued

The Ranking Forest Algorithm

1. **Parameters.**
   - B number of bootstrap replicates,
   - n^* bootstrap sample size,
   - tuning parameters of TreeRank - depth D and presence/absence of pruning,
   - $\mathcal{FR}$ feature randomization strategy,
   - d pseudo-metric over orderings.

2. **Bootstrap profile makeup.**

   (a) (Resampling step.) Build B independent bootstrap samples $\mathcal{D}_1^*, \ldots, \mathcal{D}_B^*$, by drawing with replacement $n^* \cdot B$ pairs among the original training sample $\mathcal{D}_n$.

   (b) (Randomized TreeRank.) For $b = 1, \ldots, B$, run TreeRank combined with the feature randomization method $\mathcal{FR}$ based on the sample $\mathcal{D}_b^*$, yielding the ranking tree $\mathcal{T}_b^*$, related to the partition $\mathcal{P}_b^*$ of the space $\mathcal{X}$.

   The aggregation results from a median ranking rule over the largest subpartition $\mathcal{P}^*$ extracted from the collection of partitions implemented by the ranking trees. Indeed, every ranking tree $\mathcal{T}$ induces an ordering $\preceq_{\mathcal{T}}$ over the cells of $\mathcal{P}^*$ and the median ranking tree induces an ordering defined as the solution of a distance minimization problem.

$$\preceq^* = \underset{\preceq \in \mathbf{R}(\mathcal{P}^*)}{\arg\min} \sum_{\mathcal{T}} d(\preceq, \preceq_{\mathcal{T}}) \ ,$$

   where $\mathbf{R}(\mathcal{P}^*)$ denotes all possible orderings of cells in the subpartition $\mathcal{P}^*$, and d defines a distance between orderings, such as the Kendall tau (count of concordant pairs when comparing the two orderings), Spearman rule (L_1 distance between the orderings seen as vectors) or Spearman correlation coefficient (L_2 distance between the orderings), for instance.

3. **Aggregation.** Compute the largest subpartition partition $\mathcal{P}^* = \bigcap_{b=1}^B \mathcal{P}_b^*$. Let $\preceq_b^*$ be the ranking of $\mathcal{P}^*$'s cells induced by $\mathcal{T}_b^*$, $b = 1, \ldots, B$. Compute a median ranking $\preceq^*$ related to the bootstrap profile $\Pi^* = \{\preceq_b^* :\ 1 \leq b \leq B\}$ with respect to the metric d on $\mathbf{R}(\mathcal{P}^*)$:

$$\preceq^* = \underset{\preceq \in \mathbf{R}(\mathcal{P}^*)}{\arg\min} \sum_{b=1}^{B} d(\preceq, \preceq_b^*) \ ,$$

   as well as the scoring rule $s_{\preceq^*, \mathcal{P}^*}(\mathrm{x})$.

function which, in theory, is related to the target regression function η. The criterion used in both methods is the following:

$$\frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\{-Y_i \cdot f(X_i)\}\right) \ ,$$

or (if $Y_i \in \{0, 1\}$)

$$\frac{1}{n} \sum_{i=1}^{n} \left(-Y_i \cdot f(X_i) + \log\left(1 + \exp\{f(X_i)\}\right)\right) \ .$$

An iterative boosting-type procedure is used for the optimization in LogitBoost, while Kernel Logistic Regression relies on convex risk minimization with a penalty proportional to the RKHS norm of f.

P-norm Push ([15]). The criterion used in this method is very similar to RankBoost but the idea is to put more weight on the largest discrepancies. The algorithm implements the optimization of the following criterion:

$$\frac{2}{n(n-1)} \cdot \sum_{i=1}^{n} \left(\sum_{j=i+1}^{n} \exp\left\{(Y_i - Y_j)(f(X_i) - f(X_j))\right\} \right)^p$$

with $p \geq 1$, and f being a linear combination of weak ranking rules as in RankBoost. We point out that in the case where $p = 1$, the P-norm Push is the same algorithm as RankBoost, larger values of p put more emphasis on the largest scores.

In the present paper, we only consider those methods for which there is a publicly available implementation: KLR, RankBoost, RankLS, RankSVM, AdaBoost and also P-norm Push.

4 Numerical experiments

4.1 Data sets

The methods presented in the paper have been tested on several data sets, both real (10) and artificial (3).

– Artificial data sets are gaussian mixtures, in the space $\mathbb{R}^{20}$, called *GaussEasy20d*, *GaussMed20d* and *GaussHard20d*. They have an increasing complexity as the distance between the centers decreases. The covariance matrices for each class conditional distributions can be different but both are diagonal.
– Real data sets are ten benchmark data from the UCI repository and available at

  `<http://archive.ics.uci.edu/ml/>`

  with their characteristics given in Table 1.

4.2 Description of algorithms

In the series of experiments presented in this section, several variants of TreeRank have been tested and compared among each other and to their main competitors. More precisely, we first compare twelve variants of TreeRank on the three simulated toy examples. We then retain eight of them for an insightful comparison on the benchmark datasets from the UCI repository. Eventually, we compare two well-performing versions of the TreeRank algorithm with the six methods previously cited : RankBoost, RankLS, RankSVM, KLR, AdaBoost and P-norm Push.

Before specifying the TreeRank versions considered in these experiments, we first state some general observations and notations regarding the parametrization of these algorithms.

As explained in 3.1, any classification method can be used as *splitting rule* for growing a ranking tree, the choice of it mainly depending on the expectations of the user regarding the ranking rule (interpretability, computational cost, flexibility, etc.). In these experiments we have been using both the LeafRank splitting rule previously described and SVM classifiers. In the latter case, the kernel involved is either linear or gaussian, the one leading to the best performances being retained.

Besides, TreeRank variants based on resampling involve an aggregation procedure, leading to the *consensus* ranking (see 3.1). Here, we compute the median ranking rule using the pseudo-distance between orderings induced by the Spearman correlation coefficient.

Eventually, two additional parameters, d_1 and d_2, have to be considered when running the Ranking Forest algorithm. Both parameters are related to feature randomization, affecting respectively the master ranking tree output by TreeRank and the subtrees output by the LeafRank procedure. Precisely, d_1 denotes the number of predictors randomly chosen (among the total number of dimensions) to optimize the local AUC at each node of the master ranking tree. At each step of the TreeRank procedure, the best split will be chosen among those d_1 predictors. Similarly, d_2 denotes the number of predictors randomly chosen at each node of the classification subtrees when the LeafRank procedure is used as splitting rule. Again, at each step of the latter, the best split will be chosen among those d_2 predictors. If we denote by d the total number of explanatory variables (which is also the dimension of the observation vectors X_i), the following inequality stands true: $d_2 \leq d_1 \leq d$.

We now provide the specifications for each variant of the TreeRank algorithm used in the numerical experiments described below.

- TRK_{CART} - single ranking tree with a splitting rule based on the LeafRank procedure,
- TRK_{SVM} - single ranking tree with a splitting rule based on a SVM classifier,

Name	Sample de Size	Number of variables	Rate of positives	Categorical variables	Description
GaussEasy20d	2000	20	0.5	no	gaussian mixture
GaussMed20d	2000	20	0.5	no	gaussian mixture
GaussHard20d	2000	20	0.5	no	gaussian mixture
Australian Credit	690	14	0.44	yes	Credit risk data
German Credit	1000	20	0.7	yes	Credit risk data
Japanese Credit	690	15	0.45	yes	Credit risk data
Breast Cancer Diagnosis	569	30	0.37	yes	Detection de malignant tumors
Breast Cancer Original	683	9	0.35	yes	Detection of malignant tumors
Heart Disease	270	13	0.44	yes	Detection of heart illness
Hepatitis	112	18	0.17	yes	Detection of hepatitis
Autos MPG	392	7	0.53	yes	Levels of fuel consumption
Congressional Vote	232	16	0.53	yes	American Congress voting results
Ionosphere	351	34	0.64	yes	Detection of structures

Table 1 Description of synthetic and UCI benchmark data.

- Bagg$_{\text{CART}}$ - bagged version based on the aggregation of $B = 50$ *resampled* ranking trees, which growth is based on the LeafRank procedure,
- Bagg$_{\text{SVM}}$ - bagged version based on the aggregation of $B = 50$ *resampled* ranking trees, which growth is based on a SVM classifier,
- RF1$_{\text{CART}}$ - Ranking Forest version based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_1 = \lfloor d/4 \rfloor$), which growth is based on the LeafRank procedure,
- RF1$_{\text{SVM}}$ - Ranking Forest version based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_1 = \lfloor d/4 \rfloor$), which growth is based on a SVM classifier,
- RF2$_{\text{CART}}$ - Ranking Forest version based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_1 = \lfloor 1 \rfloor$), which growth is based on the LeafRank procedure,
- RF2$_{\text{SVM}}$ - Ranking Forestversion based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_1 = \lfloor 1 \rfloor$), which growth is based on a SVM classifier,
- RF3$_{\text{CART}}$ - Ranking Forest version based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_2 = \lfloor d/4 \rfloor$), which growth is based on the LeafRank procedure,
- RF4$_{\text{CART}}$ - Ranking Forest version based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_2 = \lfloor 1 \rfloor$), which growth is based on the LeafRank procedure,
- RF5$_{\text{CART}}$ - Ranking Forest version based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_1 = \lfloor d/2 \rfloor$ and $d_2 = \lfloor d/4 \rfloor$), which growth is based on the LeafRank procedure,
- RF6$_{\text{CART}}$ - Ranking Forest version based on the aggregation of $B = 50$ *resampled and randomized* ranking trees (with $d_1 = \lfloor d/4 \rfloor$

and $d_2 = \lfloor 1 \rfloor$), which growth is based on the LeafRank procedure.

4.3 Performance and stability metrics

Performance. In the following experiments we measured the algorithms performances mainly in terms of AUC and local AUC, but also with other criteria. When running the ranking algorithms on real datasets, these quantities are estimated using V-fold cross validation. We recall the principle of for a generic performance measure Π: from the training set $\mathcal{D}_n$, consider $V = 10$ blocks $\{\mathcal{D}^{(1)}, \cdots, \mathcal{D}^{(V)}\}$ of equal size which are randomly chosen and the expected performance is estimated as follows:

$$\overline{\Pi} = \frac{1}{V} \sum_{v=1}^{V} \widehat{\Pi}^{(v)},$$

where, for each $v \in \{1, \cdots, V\}$, the quantity $\widehat{\Pi}^{(v)}$ denotes the empirical estimate of Π for the scoring rule $\widehat{s}^{(-v)}$ trained on the data set $\mathcal{D}^{(-v)} = \mathcal{D}_n \setminus \mathcal{D}^{(v)}$ and computed over $\mathcal{D}^{(v)}$. We apply the cross validation to the following criteria:

– AUC - Area Under an ROC curve,
– AUC@u% - Local AUC with values $u = 5, 10, 20\%$ for the proportion u of best instances ([13]),
– AP - Average Precision,
– HR@x% - Hit Ratio with proportions $x = 10\%, 20\%$.

Additionally, in the case of the AUC and local AUC, we also provide the standard type error $\widehat{\sigma}^2$ of the collection of V-fold empirical estimates.

Stability. Stability is another interesting indicator to consider when comparing several algorithms. We propose two different and complementary ways to evaluate it. As V-fold cross validation is used on every dataset to provide *mean* values of the AUC and Local AUC, we provide for each algorithm an estimate of the standard deviation of the considered performance metrics. Moreover, on the simulated toy examples, we also consider an *instability* measure given by

$$\mathbf{Instab(S)} = \mathbb{E}\left[d(\preceq_b, \preceq_{b'})\right],$$

where d is a distance between orderings based on Spearman correlation coefficient and the orderings $\preceq_b$ and $\preceq_{b'}$ are respectively induced on the largest subpartition of the feature space $\mathcal{X}$ by two *resampled* ranking trees, meaning that the scoring rules $\preceq_b$ and $\preceq_{b'}$ are obtained by running the TreeRank heuristic on two *i.i.d.* learning datasets, resampled from the original one $\mathcal{D}_n$. Based on a resampling procedure, this indicator can be easily estimated through the following formula:

$$\widehat{\mathbf{Instab}} = \frac{2}{B(B-1)} \sum_{1 \le b < b' \le B} d(\preceq_b, \preceq_{b'}),$$

where B is the total number of bootstrap samples drawn from $\mathcal{D}_n$.

4.4 Results

Computational cost. Before going into the detail of the experimental results, we give a first insight into the computational cost of the TreeRank variants. It is well known that tree-based algorithms can yield high computational costs, although in most cases, optimal trees are grown through bottom-up greedy procedures, avoiding a costly exhaustive search among the collection of optimal trees.

In our experiments, the computational cost mainly comes from the cross validation based pruning stage within both TreeRank and LeafRank procedures (see 3.1). Indeed, considering the *GaussEasy20d* dataset (2000 observations for 20 relevant dimensions), growing a ranking tree through the TRK$_{\text{CART}}$ variant takes less than 2 seconds when the sizes are settled as input parameters. Yet, it takes about 20 seconds when the size of both the subtrees and the master ranking tree are automatically selected through 5-fold cross validation. Thus, cross validation based pruning will yield heavy computational costs when running the Ranking Forests procedure, which will grow and aggregate $B = 50$ ranking trees. Indeed, growing a single ranking tree through Bagg$_{\text{CART}}$ or any other Ranking Forest variant on the *GaussEasy20d* dataset takes about 15 minutes with a 5-fold cross validation based pruning. This computational time depends on the randomization level: the more randomized the algorithm, the less costly. Yet, it can be reduced to 2 minutes when the sizes of the grown trees are settled *a priori*.

Besides, making use of SVM based splitting rules yields heavier computational costs. Thus, the Bagg$_{\text{SVM}}$ procedure and its randomized counterparts need about 20 minutes to grow an automatically pruned (5-fold cross validation) aggregated ranking tree, and about 4 minutes when the sizes of the trees are settled in advance. Eventually, the computational costs observed on most of the benchmark datasets are lower to those detailed above (or comparable in some cases), due to the lower dimension and the sparsity of the problems.

Comparison between the variants of TreeRank. The main objective of this series of experiments is to compare the behavior of several versions of the TreeRank algorithm. We first run twelve heuristics on three toy examples based on gaussian mixtures: *GaussEasy20d*, *GaussMed20d* and *GaussHard20d*. In order to provide mean values for the performance indicators, we run all heuristics on $N = 30$ *i.i.d.* learning resamples of $\mathcal{D}_n$, with $n = 1000$ copies of the random pair $(X, Y) \in \mathcal{X} \times \{-1, +1\}$, and test the output scoring rules on a single test sample of $n = 1000$ observations. Given the computational costs described previously, for this series of experiments trees are not pruned automatically. For sake of computational time, the sizes of the grown trees (and subtrees) has been defined *a priori*: the maximum depth of the master ranking trees has been settled to $d_1 = 8$ and the maximum depth of the sub-ranking trees output by the LeafRank procedure to $d_2 = 6$. Figure 3 gives a global insight into the performances of TreeRank variants on these toy examples. The average rank of each variant over the three toy examples is displayed for the performance indicators described in previous Subsection. Moreover, visual displays for each datasets are provided on Figure 4. The histograms representing both performance (AUC) and standard type error ($\hat{\sigma}^2$). We also report the Local AUC measure corresponding to a proportion $u = 20\%$ of best instances. Eventually, the quantitative results can be found in Tables 2, 3 and 4.

The overview given by Figure 3 shows that RF1$_{\text{SVM}}$ clearly outperforms all other procedures, except in terms of instability. We also notice that resampling and randomization yield

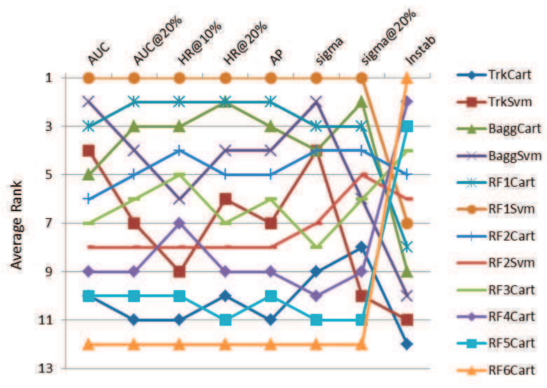
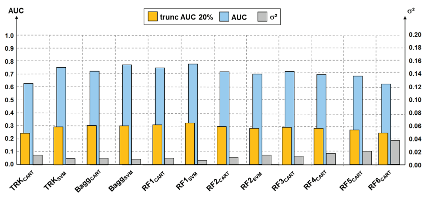


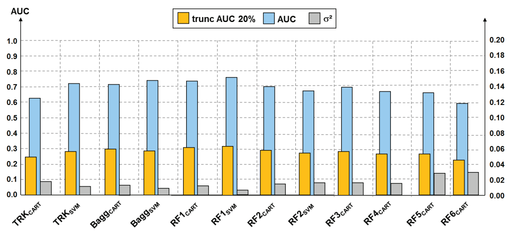
Fig. 3 Average ranks of TreeRank variants on the three toy example for several performance indicators.

better results, both in terms of learning performance and robustness, while the level of randomization remains *reasonable*. Indeed, highly randomized procedures RF4$_{\text{CART}}$, RF5$_{\text{CART}}$ and RF6$_{\text{CART}}$ yield the worst results.

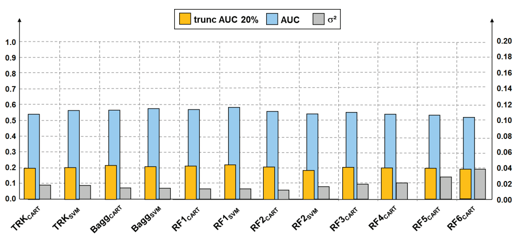
The graphs of Figure 4 allow to confirm our first observations and to go more into details. Thus, we notice that, when considering single ranking trees, the best performance is achieved by scoring rules using SVM classifiers as splitting rule, due to a superior representational capacity. As expected, performance is increased when aggregating many ranking trees. It is also interesting to notice that the choice of the splitting rule seems to have a weaker impact on the performances of those heuristics. Another interesting observation is related to the effect of feature randomization: it seems that randomization does not affect significantly the heuristics based on an SVM-type splitting rule. Moreover, as already mentioned, the more important the level of randomization, the worst the AUC performance (and the less robust the estimation), but we will see in the next experiment that this is not necessarily the case when considering real data sets. Actually, this phenomenon is not really surprising and can be explained in two different ways. As described in 3.1, Ranking Forest is a combination of *resampling*, *randomization* and *aggregation*. Although it is now well known in Machine Learning that *aggregating bootstrap* versions of a classifier improves both its performance and robustness, it is also quite clear that picking up *randomly* a subset of predictors to learn a model introduces some instability. The underlying principle of Ranking



a. Sample GaussEasy20D



b. Sample GaussMed20D



c. Sample GaussHard20D

Fig. 4 Comparison of TreeRank variants on simulated datasets.

Forests is precisely that randomization could improve the learning capacity of a classifier by focusing on subsets of most relevant predictors, the instability introduced in the process being compensated by the aggregation of several *re-sampled* classifiers. A first explanation to the results displayed on Figure 4 is that aggregating $B = 50$ ranking trees might not be sufficient to compensate the instability introduced by the different levels of randomization considered in the experiment. Moreover, it seems reasonable to state that randomization will perform better when the optimal model is sparse than when all predictors are of interest, which is the case in the simulated toy examples. Indeed, on such datasets randomization will lead to neglect relevant information, while in sparse cases it will improve the chances to select the most relevant variables.

On these toy examples, best performances in terms of total and local AUC and of standard type error are achieved by the RF1$_{\text{SVM}}$ heuristic and interestingly, the algorithms reaching the best performance in terms of low variance are the ones with high randomization (RF3$_{\text{CART}}$, RF4$_{\text{CART}}$, RF5$_{\text{CART}}$, RF6$_{\text{CART}}$). Yet, given the weaker ranking performance of these highly randomized versions (in comparison to the others), we discard the following heuristics RF2$_{\text{CART}}$, RF2$_{\text{SVM}}$, RF4$_{\text{CART}}$ and RF6$_{\text{CART}}$ from the subsequent comparisons.

In the following series of experiments, we focus on the eight remaining heuristics that we compare on real benchmark data sets, gathered from the UCI repository. This time, model selection is conducted via the pruning procedure introduced in 3.1. Therefore, the regularization coefficient λ has been calibrated through a nested stratified 5-fold cross validation procedure. As previously, a global overview of the performances overall the benchmark datasets is provided by Figure 5. Visual displays are also detailed for some of the datasets on Figure 6 below. Again, the histograms represent performance and standard type error $(\text{AUC}, \hat{\sigma}^2)$, as well as Local AUC measure AUC@u% for a proportion $u = 20\%$ of best instances.

The quantitative results can be found in Tables 5 and 6, gathering full and local AUC for

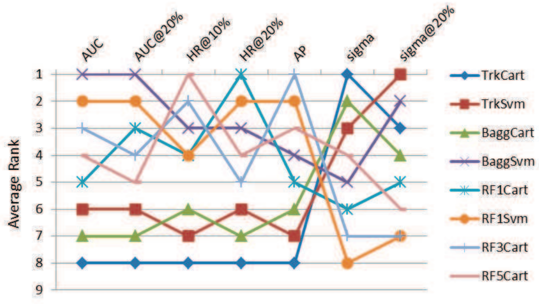
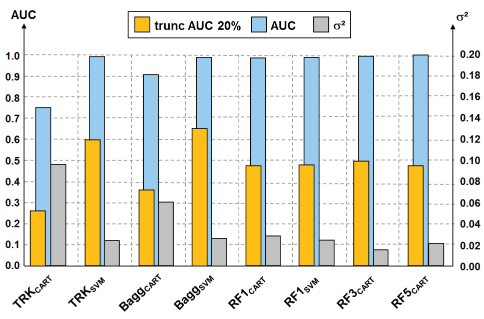
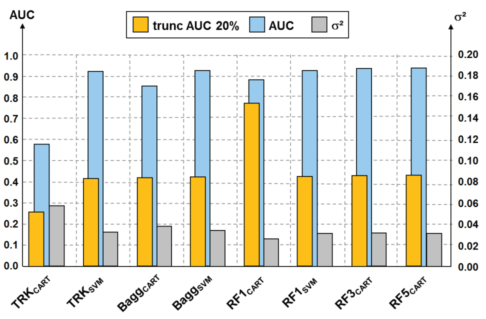


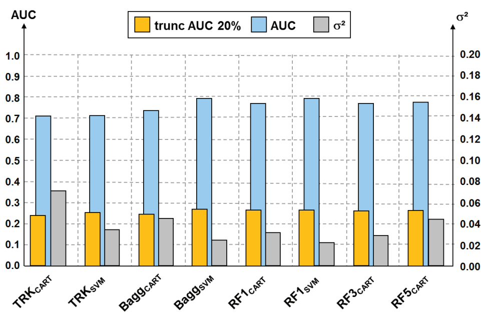
Fig. 5 Average ranks of TreeRank variants on the 10 benchmark datasets for several performance indicators.



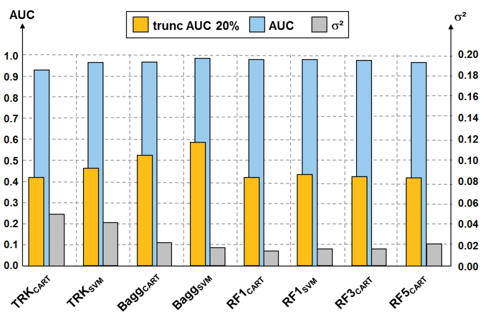
a. Dataset Congressional Vote



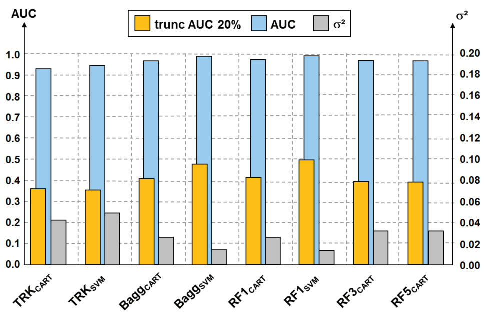
b. Dataset Australian Credit



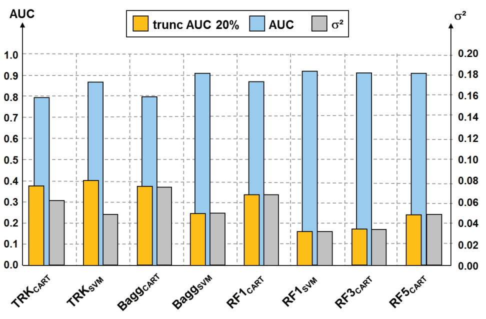
c. Dataset German Credit



d. Dataset Autos MPG



e. Dataset Ionosphere



f. Dataset Heart Disease

Fig. 6 Comparison of TreeRank variants on benchmark datasets.

proportions $u = 20, 10, 5\%$ of best instances indicated together with their associated standard type errors, given in parentheses, as well as hit ratios for proportions 10% and 20%, and the average precision. All these values are computed through a stratified 10-fold cross validation, as described in 4.3.

The global picture displayed on Figure 5 clearly shows two dynamics. When considering the indicators related to learning performances, we notice that randomized procedure are among the most efficient. Moreover, different behaviours appear when considering the robustness indicators. Indeed, the graph shows that the algorithms with highest AUC, average precision and hit ratios are also the ones having highest standard deviations, and conversely. This is due to the fact that these procedures are randomized. Indeed, the only procedure presenting both good learning performances and robustness is Bagg_{SVM}.

Besides, as noticed on the simulated toy examples, we can see on the graphs of Figure 6 that in most cases, making use of SVM classifiers as splitting rule leads to more efficient ranking rules, in terms of both full and Lo-

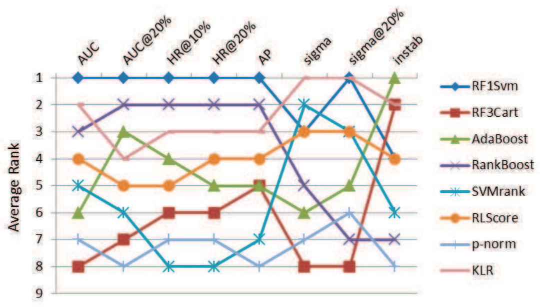


Fig. 7 Average ranks of TreeRank variants and some competitors on the toy examples for several performance indicators.

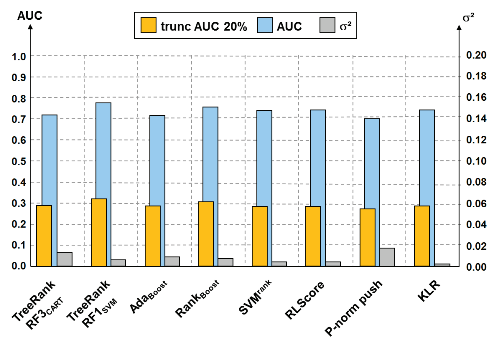
cal AUC, but also in terms of stability (lower standard deviation). As explained previously, SVM based classifier allow to capture the complex geometry of the regression level sets in a more flexible manner than CART based decision rules. Yet, when focusing on the *Heart Disease* dataset for instance, we can observe that Bagg_{SVM} and RF1_{SVM} perform worse on the 20% best instances than their CART-based counterparts $\text{Bagg}_{\text{CART}}$ and RF1_{CART}. This observation means that in this particular case, the topology of the first level sets (corresponding to the 20% of best instances) is better approximated by linear partitions than by nonlinear Gaussian partitions. Indeed, the geometry of the level sets can be different from one dataset to another, but also from one level set to another. This example emphasizes the advantage of using different splitting rules in the TreeRank procedure, its performance being intrinsically related to the capacity of the latter to recover the geometry of the level sets. Moreover, when comparing TRK_{CART} and TRK_{SVM} with their *resampled* versions, the positive impact of resampling and aggregation procedures on the performances and the variance of the output rules clearly appears.

Eventually, the main difference with the previous experiments lies in the impact of feature randomization. Even if Bagg_{SVM} achieves very good performances on several datasets, the best results are generally provided by resampled and randomized versions. Three heuristics perform particularly better than the others: Bagg_{SVM} and RF1_{SVM}, which often get similar results, and RF3_{CART}. For the following series of experiments we will retain only two (RF1_{SVM} and RF3_{CART}), which we will compare to several competitors.

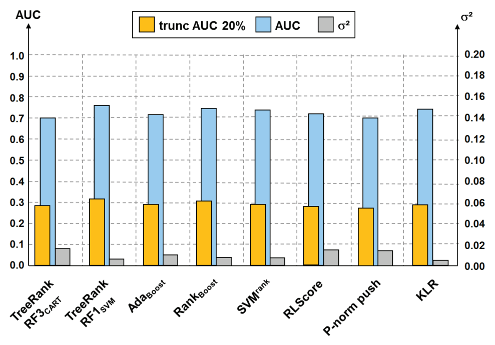
TreeRank *and some competitors.* In this series of experiments, both RF1_{SVM} and RF3_{CART} heuristics are compared to six scoring algorithms: RankBoost, RankLS, RankSVM, KLR, AdaBoost and P-norm Push. We first consider the performances of these heuristics on the three toy examples previously described, before considering the benchmark datasets from the UCI repository. As before, for the experiments based on simulated datasets the maximum depths of the (sub-) trees has been settled *a priori* respectively to $d_1 = 8$ and $d_2 = 6$, while on the UCI benchmark datasets model selection is based on a nested stratified 5-fold cross validation procedure. The quantitative results obtained on synthetic data are summarized in Tables 7, 8 and 9. They are also displayed on Figure 8 and a global overview of the performances of each algorithm is displayed on Figure 7.

As in the previous series of experiments, we notice that generally the algorithms with highest performances are also the less robust (see Figure 7). Moreover, we can see on Figure 8 that the performances of all heuristics in terms of global and local AUC are very similar on these data. Yet, we can observe that RankLS seems to perform a little better on the *GaussHard20d* data set. In the other cases the leadership goes to a version of Ranking Forest with an SVM classifier used as splitting rule. It is also interesting to notice that the instability results are much more variable than in previous experiments. Indeed, there is a big difference between the algorithms and from a data set to another. AdaBoost and RF3_{CART} heuristics appear to be more stable than the others. We can also underline that for several algorithms, variance increases with the complexity of the ranking problem (from *GaussEasy20d* data set to *GaussHard20d*). It is the case in particular for RankBoost, RankLS, RankSVM and KLR.

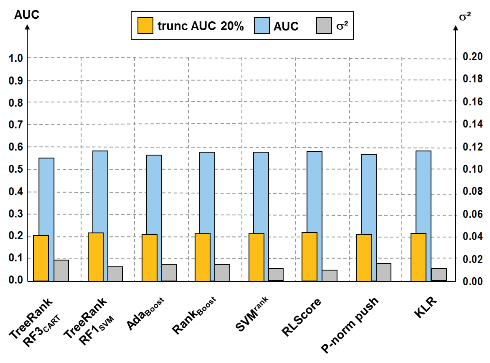
Yet, differences between the several algorithms appear more clearly on the real datasets. Performance and robustness results are summarized in Tables 10 and 11, detailed graphs per dataset are displayed on Figure 10 and a global comparison of the algorithms is given by Figure 9.



a. Dataset GaussEasy20D



b. Dataset GaussMed20D



c. Dataset GaussHard20D

Fig. 8 Comparison of TreeRank and some competitors on simulated datasets.

We clearly see on Figure 9 that both Ranking Forest algorithms, either based on the LeafRank procedure or on a SVM classifier, perform mostly better than other methods. This is particularly true regarding learning performances, while both methods present lower robustness. When going into more detail, we notice three exceptions: Australian, German and Japanese credit datsets, on which RankBoost achieves better performances than TreeRank (see Figure 10). However, TreeRank heuristics clearly provide better results on the best instances for all the considered datasets. This is related to the fact that TreeRank recursively optimizes the AUC and thus estimates the optimal ROC curve, while all other methods optimize the AUC criterion in a global manner.

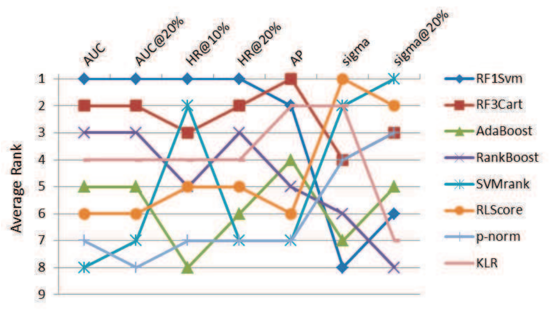
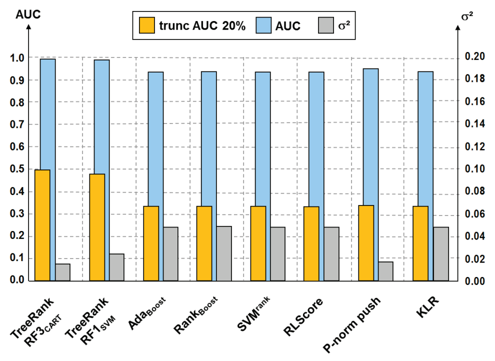


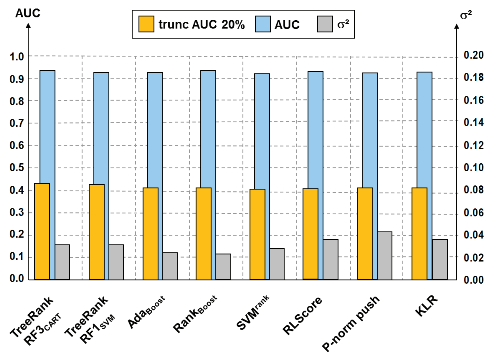
Fig. 9 Average ranks of TreeRank variants and some competitors on the 10 benchmark datasets for several performance indicators.

5 Discussion

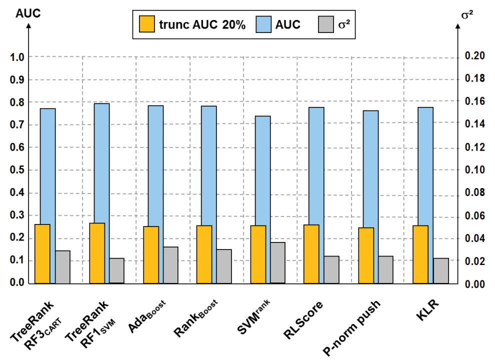
The numerical experiments performed in the present paper show that the three methodologies (Ranking trees, Pairwise SVM, Pairwise Boosting) are very competitive for AUC maximization. It is shown that the ranking trees heuristics take the lead when it comes to focusing on best instances. Theoretical evidence supports this experimental fact. Indeed, in their very construction, ranking trees derived from the TreeRank/LeafRank algorithm actually optimize the AUC locally. The present paper illustrates the theoretical result introduced in



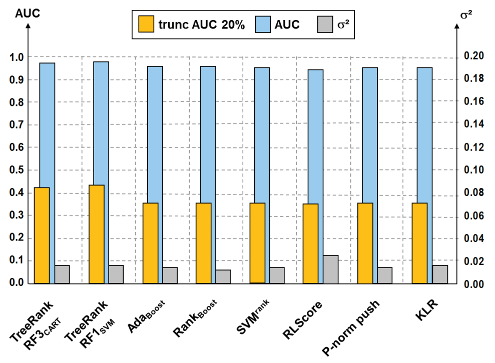
a. Dataset Congressional Vote



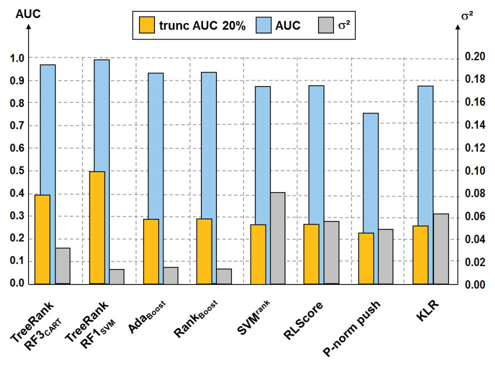
b. Dataset Australian Credit



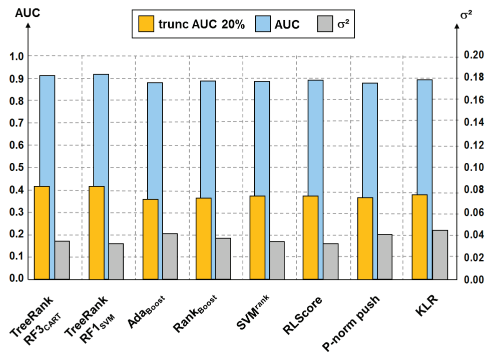
c. Dataset German Credit



d. Dataset Autos MPG



e. Dataset Ionosphere



f. Dataset Heart Disease

Fig. 10 Comparison of TreeRank and some competitors on benchmark datasets.

([1]) which states that, by using ranking trees, convergence to the optimal ROC curve is performed in a stronger sense than the AUC. Indeed, AUC optimality corresponds to convergence in L_1-norm of the corresponding ROC curve, while it can be shown that the recursive construction of ranking trees guarantees convergence in L_∞-norm. This property explains the improvement obtained with our method with respect to local ranking performance measures.

Acknowledgments. We warmly thank Cynthia Rudin who kindly provided the code for the P-norm Push algorithm.

AUC AUC@20% AUC@10% AUC@5%	*GaussEasy20d*	*GaussMed20d*	*GaussHard20d*
TRK$_{\mathrm{CART}}$	0.626 (±0.015)	0.625 (±0.017)	0.539 (±0.018)
	0.243 (±0.011)	0.244 (±0.013)	0.195 (±0.015)
	0.130 (±0.010)	0.132 (±0.012)	0.102 (±0.009)
	0.067 (±0.006)	0.068 (±0.006)	0.053 (±0.006)
TRK$_{\mathrm{SVM}}$	0.752 (±0.009)	0.721 (±0.011)	0.562 (±0.017)
	0.292 (±0.011)	0.282 (±0.015)	0.201 (±0.015)
	0.150 (±0.008)	0.145 (±0.009)	0.103 (±0.010)
	0.075 (±0.004)	0.073 (±0.005)	0.052 (±0.006)
Bagg$_{\mathrm{CART}}$	0.722 (±0.010)	0.714 (±0.013)	0.566 (±0.014)
	0.300 (±0.007)	0.297 (±0.009)	0.214 (±0.011)
	0.163 (±0.006)	0.163 (±0.006)	0.113 (±0.009)
	0.085 (±0.005)	0.086 (±0.004)	0.060 (±0.007)
Bagg$_{\mathrm{SVM}}$	0.772 (±0.008)	0.739 (±0.009)	0.576 (±0.014)
	0.298 (±0.010)	0.287 (±0.012)	0.207 (±0.013)
	0.156 (±0.007)	0.151 (±0.008)	0.107 (±0.010)
	0.080 (±0.005)	0.078 (±0.005)	0.055 (±0.007)
RF1$_{\mathrm{CART}}$	0.747 (±0.010)	0.737 (±0.012)	0.569 (±0.013)
	0.308 (±0.009)	0.308 (±0.009)	0.211 (±0.011)
	0.167 (±0.005)	0.165 (±0.006)	0.113 (±0.008)
	0.087 (±0.004)	0.087 (±0.004)	;0.060 (±0.006)
RF1$_{\mathrm{SVM}}$	**0.776 (±0.006)**	**0.760 (±0.006)**	**0.582 (±0.013)**
	0.319 (±0.006)	**0.316 (±0.005)**	**0.216 (±0.007)**
	0.172 (±0.004)	**0.171 (±0.003)**	**0.115 (±0.007)**
	0.089 (±0.003)	**0.089 (±0.003)**	**0.061 (±0.006)**
RF2$_{\mathrm{CART}}$	0.718 (±0.011)	0.701 (±0.014)	0.557 (±0.012)
	0.294 (±0.011)	0.289 (±0.011)	0.205 (±0.009)
	0.162 (±0.007)	0.157 (±0.008)	0.109 (±0.007)
	0.086 (±0.003)	0.083 (±0.004)	0.056 (±0.007)
RF2$_{\mathrm{SVM}}$	0.697 (±0.015)	0.675 (±0.016)	0.539 (±0.016)
	0.280 (±0.012)	0.271 (±0.011)	0.197 (±0.009)
	0.152 (±0.007)	0.145 (±0.008)	0.105 (±0.008)
	0.081 (±0.005)	0.077 (±0.005)	0.055 (±0.007)
RF3$_{\mathrm{CART}}$	0.718 (±0.013)	0.698 (±0.016)	0.551 (±0.019)
	0.289 (±0.010)	0.283 (±0.013)	0.203 (±0.012)
	0.156 (±0.008)	0.155 (±0.006)	0.108 (±0.009)
	0.081 (±0.005)	0.081 (±0.005)	0.057 (±0.007)
RF4$_{\mathrm{CART}}$	0.693 (±0.017)	0.670 (±0.015)	0.540 (±0.021)
	0.278 (±0.013)	0.267 (±0.013)	0.200 (±0.013)
	0.151 (±0.009)	0.144 (±0.009)	0.106 (±0.008)
	0.080 (±0.006)	0.078 (±0.005)	0.055 (±0.005)
RF5$_{\mathrm{CART}}$	0.684 (±0.021)	0.662 (±0.028)	0.535 (±0.028)
	0.269 (±0.014)	0.265 (±0.017)	0.195 (±0.016)
	0.148 (±0.008)	0.144 (±0.010)	0.106 (±0.011)
	0.078 (±0.005)	0.076 (±0.006)	0.054 (±0.008)
RF6$_{\mathrm{CART}}$	0.622 (±0.037)	0.592 (±0.029)	0.519 (±0.028)
	0.244 (±0.019)	0.225 (±0.018)	0.189 (±0.017)
	0.131 (±0.012)	0.123 (±0.014)	0.099 (±0.013)
	0.068 (±0.007)	0.064 (±0.010)	0.053 (±0.008)

Table 2 Comparison of 12 variants of the TreeRank algorithm on simulated data. Performance is measured through the AUC and Local AUC for three different values $u = 20, 10, 5\%$, given in that order. Standard type error $\hat{\sigma}^2$ for these values is indicated in parentheses.

HR@10% ; HR@20% AP	*GaussEasy20d*	*GaussMed20d*	*GaussHard20d*
TRK_{CART}	68%; 66% 0.59	68%; 65% 0.58	54%; 55% 0.53
TRK_{SVM}	79%; 78% 0.67	75%; 74% 0.64	54%; 56% 0.54
$\text{Bagg}_{\text{CART}}$	84%; 79% 0.68	82%; 77% 0.66	60%; 59% 0.56
Bagg_{SVM}	81%; 79% 0.69	77%; 75% 0.66	57%; 57% 0.55
RF1_{CART}	86%; 80% 0.69	84%; 79% 0.68	60%; 58% 0.56
RF1_{SVM}	**88%; 83%** **0.71**	**86%; 81%** **0.69**	**61%; 59%** **0.56**
RF2_{CART}	84%; 77% 0.67	79%; 75% 0.65	58%; 57% 0.55
RF2_{SVM}	79%; 74% 0.66	74%; 71% 0.63	56%; 55% 0.53
RF3_{CART}	81%; 76% 0.67	79%; 73% 0.65	57%; 56% 0.54
RF4_{CART}	78%; 74% 0.65	74%; 70% 0.62	57%; 55% 0.53
RF5_{CART}	77%; 72% 0.64	74%; 69% 0.62	56%; 54% 0.53
RF6_{CART}	69%; 66% 0.60	64%; 60% 0.57	53%; 53% 0.52

Table 3 Comparison of 12 variants of the TreeRank algorithm on simulated data. Performance is measured through the hit ratios for proportions 10% and 20%, respectively given in that order, and the average precision indicated below.

$\widehat{\textbf{Instab}}$	*GaussEasy20d*	*GaussMed20d*	*GaussHard20d*
TRK_{CART}	*137.5*	*141.7*	*149.5*
TRK_{SVM}	*56.5*	*64.5*	*108.2*
$\text{Bagg}_{\text{CART}}$	*15.3*	*16.3*	*16.3*
Bagg_{SVM}	*24.4*	*27.8*	*41.3*
RF1_{CART}	*6.5*	*7.2*	*7.9*
RF1_{SVM}	*5.5*	*6.4*	*9.6*
RF2_{CART}	*4.5*	*4.8*	*5.7*
RF2_{SVM}	*5.6*	*6.1*	*7.2*
RF3_{CART}	*3.5*	*3.5*	*3.6*
RF4_{CART}	*3.1*	*3.3*	*3.2*
RF5_{CART}	*3.4*	*3.3*	*3.3*
RF6_{CART}	**3.0**	**3.2**	**3.1**

Table 4 Comparison of 12 variants of the TreeRank algorithm on simulated data in terms of instability, computed with $B = 20$ bootstrap samples.

AUC AUC@20% AUC@10% AUC@5%	TRK$_{\text{CART}}$	TRK$_{\text{SVM}}$	Bagg$_{\text{CART}}$	Bagg$_{\text{SVM}}$	RF1$_{\text{CART}}$	RF1$_{\text{SVM}}$	RF3$_{\text{CART}}$	RF5$_{\text{CART}}$
Congressional Vote	0.749 (±0.096)	0.991 (±0.024)	0.905 (±0.06)	0.991 (±0.026)	0.985 (±0.028)	0.987 (±0.024)	**0.992 (±0.015)**	0.988 (±0.021)
	0.261 (±0.060)	0.595 (±0.105)	0.359 (±0.049)	0.650 (±0.112)	0.473 (±0.061)	0.477 (±0.054)	**0.495 (±0.059)**	0.474 (±0.053)
	0.128 (±0.052)	0.186 (±0.004)	0.232 (±0.075)	0.186 (±0.004)	0.217 (±0.064)	0.201 (±0.042)	0.201 (±0.042)	0.204 (±0.054)
	0.069 (±0.046)	0.093 (±0.002)	0.11 (±0.031)	0.093 (±0.002)	0.094 (±0)	0.094 (±0)	0.094 (±0)	0.094 (±0)
Australian Credit	0.577 (±0.057)	0.921 (±0.032)	0.852 (±0.038)	0.925 (±0.034)	0.882 (±0.026)	0.926 (±0.031)	0.934 (±0.031)	**0.939 (±0.031)**
	0.257 (±0.050)	0.414 (±0.024)	0.419 (±0.014)	0.423 (±0.016)	0.771 (±0.059)	0.425 (±0.012)	0.429 (±0.014)	**0.429 (±0.014)**
	0.134 (±0.023)	0.256 (±0.082)	0.235 (±0.031)	0.252 (±0.040)	0.663 (±0.075)	0.248 (±0.039)	0.273 (±0.059)	0.278 (±0.061)
	0.068 (±0.011)	0.115 (±0.028)	0.141 (±0.041)	0.115 (±0.012)	0.543 (±0.081)	0.111 (±0.002)	0.122 (±0.031)	0.129 (±0.038)
German Credit	0.711 (±0.071)	0.771 (±0.034)	0.737 (±0.045)	**0.794 (±0.024)**	0.771 (±0.031)	0.793 (±0.022)	0.769 (±0.029)	0.775 (±0.044)
	0.237 (±0.031)	0.254 (±0.017)	0.245 (±0.022)	**0.269 (±0.009)**	0.264 (±0.016)	0.266 (±0.012)	0.260 (±0.020)	0.263 (±0.017)
	0.123 (±0.019)	0.133 (±0.018)	0.128 (±0.020)	0.146 (±0.016)	0.146 (±0.021)	0.148 (±0.020)	0.144 (±0.022)	0.147 (±0.027)
	0.061 (±0.014)	0.065 (±0.004)	0.063 (±0.008)	0.098 (±0.033)	0.075 (±0.014)	0.081 (±0.021)	0.083 (±0.029)	0.077 (±0.015)
Japanese Credit	0.789 (±0.053)	0.924 (±0.049)	0.839 (±0.048)	0.926 (±0.044)	0.887 (±0.038)	0.923 (±0.038)	**0.931 (±0.042)**	**0.930 (±0.043)**
	0.373 (±0.043)	0.423 (±0.050)	0.406 (±0.024)	0.416 (±0.026)	0.416 (±0.015)	0.414 (±0.020)	**0.417 (±0.023)**	**0.418 (±0.027)**
	0.220 (±0.059)	0.232 (±0.053)	0.250 (±0.047)	0.245 (±0.052)	0.256 (±0.063)	0.256 (±0.068)	0.260 (±0.073)	0.274 (±0.081)
	0.106 (±0.024)	0.105 (±0.005)	0.131 (±0.035)	0.127 (±0.030)	0.125 (±0.033)	0.125 (±0.032)	0.141 (±0.047)	0.123 (±0.035)
Autos MPG	0.927 (±0.049)	0.961 (±0.041)	0.967 (±0.022)	**0.982 (±0.017)**	0.978 (±0.014)	0.978 (±0.016)	0.972 (±0.016)	0.964 (±0.021)
	0.420 (±0.011)	0.462 (±0.144)	0.523 (±0.112)	**0.584 (±0.109)**	0.418 (±0.039)	0.433 (±0.058)	0.421 (±0.016)	0.418 (±0.046)
	0.229 (±0.081)	0.184 (±0.010)	0.190 (±0.002)	0.190 (±0)	0.244 (±0.087)	0.238 (±0.079)	0.240 (±0.08)	0.236 (±0.076)
	0.092 (±0.007)	0.092 (±0.005)	0.094 (±0.004)	0.095 (±0)	0.095 (±0)	0.095 (±0)	0.095 (±0)	0.095 (±0)
Ionosphere	0.926 (±0.042)	0.943 (±0.049)	0.966 (±0.026)	0.985 (±0.014)	0.971 (±0.026)	**0.990 (±0.013)**	0.968 (±0.032)	0.966 (±0.032)
	0.360 (±0.127)	0.352 (±0.129)	0.408 (±0.116)	0.477 (±0.072)	0.411 (±0.099)	**0.494 (±0.062)**	0.392 (±0.086)	0.391 (±0.032)
	0.146 (±0.007)	0.149 (±0.008)	0.181 (±0.052)	0.156 (±0)	0.179 (±0.048)	0.156 (±0)	0.178 (±0.044)	0.179 (±0.038)
	0.074 (±0.003)	0.074 (±0.004)	0.078 (±0)	0.078 (±0)	0.078 (±0)	0.078 (±0)	0.078 (±0)	0.081 (±0.008)
Breast Cancer Diagnosis	0.958 (±0.02)	0.989 (±0.008)	0.986 (±0.016)	0.990 (±0.015)	0.990 (±0.009)	**0.994 (±0.008)**	0.988 (±0.007)	0.990 (±0.007)
	0.503 (±0.015)	0.565 (±0.063)	0.578 (±0.036)	**0.680 (±0.071)**	0.548 (±0.025)	0.595 (±0.047)	0.528 (±0.005)	0.527 (±0.006)
	0.314 (±0.059)	0.341 (±0.102)	0.319 (±0.106)	0.290 (±0.070)	0.403 (±0.121)	0.317 (±0.103)	0.453 (±0.036)	0.444 (±0.037)
	0.131 (±0.004)	0.133 (±0.003)	0.134 (±0)	0.134 (±0)	0.134 (±0)	0.134 (±0)	0.135 (±0)	0.134 (±0)
Breast Cancer Original	0.984 (±0.017)	0.995 (±0.005)	0.990 (±0.009)	0.994 (±0.007)	**0.995 (±0.005)**	0.995 (±0.006)	0.993 (±0.006)	0.993 (±0.006)
	0.568 (±0.024)	0.566 (±0.027)	0.565 (±0.014)	**0.570 (±0.027)**	0.558 (±0.009)	0.559 (±0.010)	0.556 (±0.012)	0.555 (±0.010)
	0.375 (±0.101)	0.364 (±0.107)	0.441 (±0.106)	0.399 (±0.133)	0.430 (±0.085)	0.442 (±0.076)	0.407 (±0.078)	0.409 (±0.087)
	0.154 (±0.039)	0.139 (±0.006)	0.157 (±0.044)	0.157 (±0.043)	0.148 (±0.016)	0.146 (±0.010)	0.142 (±0.002)	0.156 (±0.042)
Heart Disease	0.794 (±0.061)	0.868 (±0.048)	0.798 (±0.074)	0.908 (±0.049)	0.869 (±0.067)	**0.917 (±0.032)**	0.911 (±0.034)	0.907 (±0.048)
	0.376 (±0.054)	0.401 (±0.057)	0.372 (±0.047)	0.407 (±0.050)	0.397 (±0.042)	**0.416 (±0.027)**	0.415 (±0.026)	0.408 (±0.033)
	0.237 (±0.092)	0.238 (±0.090)	0.221 (±0.056)	0.278 (±0.099)	0.261 (±0.064)	0.273 (±0.070)	0.272 (±0.070)	0.289 (±0.076)
	0.109 (±0.037)	0.101 (±0.011)	0.139 (±0.058)	0.114 (±0.030)	0.118 (±0.033)	0.118 (±0.017)	0.121 (±0.031)	0.120 (±0.032)
Hepatitis	0.771 (±0.171)	0.813 (±0.154)	0.813 (±0.132)	**0.872 (±0.14)**	0.850 (±0.141)	0.864 (±0.139)	0.868 (±0.162)	0.842 (±0.17)
	0.405 (±0.146)	0.547 (±0.202)	0.470 (±0.132)	**0.629 (±0.213)**	0.551 (±0.24)	0.572 (±0.240)	0.625 (±0.230)	0.562 (±0.233)
	0.25 (±0.139)	0.306 (±0.129)	0.279 (±0.172)	0.412 (±0.131)	0.327 (±0.178)	0.413 (±0.138)	0.346 (±0.162)	0.277 (±0.187)
	0.174 (±0.142)	0.188 (±0.137)	0.202 (±0.177)	0.339 (±0.157)	0.251 (±0.196)	0.269 (±0.190)	0.342 (±0.154)	0.277 (±0.187)

Table 5 Comparison of eight variants of the TreeRank algorithm on benchmark data. Performance is measured through AUC and Local AUC for three different values $u = 20, 10, 5\%$, given in that order. Standard type error $\hat{\sigma}^2$ for these values is indicated in parentheses.

HR@10% ; HR@20% AP	$\mathrm{TRK}_{\mathrm{CART}}$	$\mathrm{TRK}_{\mathrm{SVM}}$	$\mathrm{Bagg}_{\mathrm{CART}}$	$\mathrm{Bagg}_{\mathrm{SVM}}$	$\mathrm{RF1}_{\mathrm{CART}}$	$\mathrm{RF1}_{\mathrm{SVM}}$	$\mathrm{RF3}_{\mathrm{CART}}$	$\mathrm{RF5}_{\mathrm{CART}}$
Congressional Vote	92%; 92% 0.75	100%; 98% 0.80	100%; 96% 0.80	100%; 98% 0.82	100%; 100% 0.84	100%; 100% 0.84	**100%**; **100%** **0.85**	100%; 100% 0.84
Australian Credit	68%; 59% 0.48	100%; 100% 0.72	100%; 100% 0.71	100%; 100% 0.75	100%; 100% 0.73	100%; 100% 0.75	100%; 100% 0.76	**100%**; **100%** **0.76**
German Credit	87%; 84% 0.80	94%; 91% 0.84	90%; 88% 0.82	**99%**; **96%** **0.87**	96%; 93% 0.85	98%; 94% 0.86	99%; 92% 0.86	99%; 93% 0.86
Japanese Credit	78%; 83% 0.64	97%; 95% 0.68	98%; 92% 0.70	99%; 95% 0.71	98%; 97% 0.70	95%; 95% 0.71	**97%**; **95%** **0.71**	**97%**; **95%** **0.72**
Autos MPG	89%; 92% 0.74	87%; 92% 0.72	98%; 99% 0.78	100%; 100% 0.77	100%; 100% 0.79	**100%**; **100%** **0.80**	100%; 100% 0.79	**100%**; **100%** **0.80**
Ionosphere	100%; 100% 0.85	100%; 97% 0.87	100%; 99% 0.91	100%; 100% 0.93	100%; 100% 0.92	**100%**; **100%** **0.94**	100%; 99% 0.92	100%; 99% 0.92
Breast Cancer Diagnosis	98%; 97% 0.57	98%; 99% 0.60	100%; 100% 0.69	100%; 100% 0.67	100%; 100% 0.69	**100%**; **100%** **0.70**	100%; 100% 0.70	100%; 100% 0.70
Breast Cancer Original	100%; 99% 0.52	99%; 99% 0.50	100%; 99% 0.60	99%; 99% 0.55	**100%**; **99%** **0.64**	100%; 99% 0.65	99%; 99% 0.64	100%; 100% 0.64
Heart Disease	71%; 72% 0.59	90%; 89% 0.67	93%; 86% 0.66	90%; 90% 0.74	93%; 94% 0.69	**97%**; **94%** **0.73**	93%; 96% 0.75	93%; 94% 0.72
Hepatitis	75%; 40% 0.25	65%; 47% 0.31	45%; 35% 0.29	**75%**; **47%** **0.41**	45%; 43% 0.31	65%; 40% 0.33	**85%**; **42%** **0.43**	65%; 42% 0.31

Table 6 Comparison of eight variants of the TreeRank algorithm on benchmark data. Performance is measured through the hit ratios for proportions 10% and 20%, given in that order respectively, and the average precision given below.

AUC AUC@20% AUC@10% AUC@5%	*GaussEasy20d*	*GaussMed20d*	*GaussHard20d*
RF1$_{\text{SVM}}$	**0.776 (±0.006)** **0.319 (±0.006)** 0.172 (±0.004) 0.089 (±0.003)	**0.760 (±0.006)** **0.316 (±0.005)** 0.171 (±0.003) 0.089 (±0.003)	**0.582 (±0.013)** **0.216 (±0.007)** 0.115 (±0.007) 0.061 (±0.006)
RF3$_{\text{CART}}$	0.718 (±0.013) 0.289 (±0.010) 0.156 (±0.008) 0.081 (±0.005)	0.698 (±0.016) 0.283 (±0.013) 0.155 (±0.006) 0.081 (±0.005)	0.551 (±0.019) 0.203 (±0.012) 0.108 (±0.009) 0.057 (±0.007)
AdaBoost	0.716 (±0.009) 0.286 (±0.007) 0.152 (±0.005) 0.080 (±0.005)	0.714 (±0.010) 0.289 (±0.008) 0.158 (±0.006) 0.083 (±0.004)	0.565 (±0.015) 0.209 (±0.010) 0.111 (±0.008) 0.057 (±0.005)
RankBoost	**0.754 (±0.007)** **0.306** (±0.009) 0.163 (±0.006) 0.087 (±0.006)	**0.746 (±0.008)** **0.307 (±0.009)** 0.166 (±0.004) 0.086 (±0.004)	**0.577 (±0.015)** **0.577 (±0.015)** 0.111 (±0.007) 0.058 (±0.005)
RankSVM	0.740 (±0.004) 0.282 (±0.005) 0.147 (±0.003) 0.075 (±0.003)	0.737 (±0.007) 0.288 (±0.007) 0.153 (±0.005) 0.081 (±0.004)	**0.578 (±0.011)** **0.213 (±0.010)** 0.112 (±0.008) 0.056 (±0.005)
RankLS	0.742 (±0.004) 0.283 (±0.005) 0.147 (±0.003) 0.075 (±0.003)	0.719 (±0.015) 0.278 (±0.008) 0.151 (±0.006) 0.080 (±0.005)	**0.581 (±0.010)** **0.216 (±0.008)** 0.112 (±0.007) 0.057 (±0.005)
P − norm push	0.700 (±0.017) 0.271 (±0.010) 0.142 (±0.006) 0.071 (±0.004)	0.701 (±0.014) 0.272 (±0.009) 0.146 (±0.006) 0.076 (±0.005)	0.566 (±0.016) 0.208 (±0.011) 0.109 (±0.008) 0.056 (±0.005)
KLR	0.742 (±0.002) 0.284 (±0.004) 0.147 (±0.004) 0.075 (±0.003)	0.740 (±0.005) 0.286 (±0.006) 0.155 (±0.004) 0.082 (±0.005)	**0.582 (±0.011)** **0.215 (±0.010)** 0.112 (±0.008) 0.057 (±0.005)

Table 7 Comparison of TreeRank and some competitors on simulated data. Total AUC and Local AUC for three different values $u = 20, 10, 5\%$ are given in that order. Standard type error $\hat{\sigma}^2$ for these values is indicated in parentheses.

HR@10% ; HR@20% AP	*GaussEasy20d*	*GaussMed20d*	*GaussHard20d*
RF1$_{\text{SVM}}$	**88%**; **83%** **0.71**	**86%**; **81%** **0.69**	**61%**; **59%** **0.56**
RF3$_{\text{CART}}$	81%; 76% 0.67	79%; 73% 0.65	57%; 56% 0.54
AdaBoost	79%; 76% 0.66	80%; 75% 0.66	59%; 58% 0.55
RankBoost	84%; 80% 0.69	84%; 79% 0.68	59%; 59% 0.56
RankSVM	72%; 62% 0.53	70%; 55% 0.51	55%; 49% ; 0.56
RankLS	77%; 76% 0.67	77%; 73% 0.65	60%; 60% 0.56
P − norm push	74%; 72% 0.64	75%; 72% 0.64	60%; 57% 0.55
KLR	77%; 76% 0.67	79%; 75% 0.67	60%; 59% 0.56

Table 8 Comparison of TreeRank and some competitors on simulated data. Performance is measured through the hit ratios for proportions 10% and 20%, respectively given in that order, the average precision, given below.

$\widehat{\textbf{Instab}}$	*GaussEasy20d*	*GaussMed20d*	*GaussHard20d*
$\mathrm{RF1}_{\mathrm{SVM}}$	*5.5*	*6.4*	*9.6*
$\mathrm{RF3}_{\mathrm{CART}}$	*3.5*	*3.5*	*3.6*
AdaBoost	**2.6**	**2.9**	**3.8**
RankBoost	*9.2*	*11.4*	*20.8*
RankSVM	*4.2*	*5.0*	*12.5*
RankLS	*3.1*	*11.2*	*10.9*
$\mathrm{P-norm\ push}$	*19.3*	*16.2*	*21.9*
KLR	**2.8**	*3.4*	*10.3*

Table 9 Comparison of TreeRank and some competitors on simulated data in terms of instability, computed with $B = 20$ bootstrap samples.

AUC AUC@20% AUC@10% AUC@5%	$RF1_{SVM}$	$RF3_{CART}$	AdaBoost	RankBoost	RankSVM	RankLS	$P - \text{norm push}$	KLR
Congressional Vote	**0.987 (±0.024)**	**0.992 (±0.015)**	0.935 (±0.048)	0.937 (±0.049)	0.935 (±0.048)	0.934 (±0.048)	0.947 (±0.017)	0.935 (±0.048)
	0.477 (±0.054)	**0.495 (±0.059)**	0.332 (±0.025)	0.333 (±0.025)	0.333 (±0.025)	0.332 (±0.025)	0.337 (±0.011)	0.332 (±0.025)
	0.201 (±0.042)	0.201 (±0.042)	0.168 (±0.012)	0.169 (±0.012)	0.17 (±0.013)	0.168 (±0.012)	0.171 (±0.006)	0.169 (±0.012)
	0.094 (±0)	0.094 (±0)	0.084 (±0.006)	0.084 (±0.006)	0.084 (±0.006)	0.084 (±0.007)	0.085 (±0.004)	0.084 (±0.006)
Australian Credit	0.926 (±0.031)	0.934 (±0.031)	0.928 (±0.024)	**0.937 (±0.023)**	0.92 (±0.028)	0.929 (±0.036)	0.924 (±0.043)	0.929 (±0.036)
	0.425 (±0.012)	**0.429 (±0.014)**	0.411 (±0.017)	0.412 (±0.014)	0.404 (±0.024)	0.405 (±0.024)	0.409 (±0.015)	0.411 (±0.022)
	0.248 (±0.039)	0.273 (±0.059)	0.201 (±0.013)	0.206 (±0.013)	0.204 (±0.013)	0.199 (±0.014)	0.206 (±0.012)	0.204 (±0.015)
	0.111 (±0.002)	0.122 (±0.031)	0.101 (±0.006)	0.103 (±0.011)	0.103 (±0.010)	0.096 (±0.013)	0.102 (±0.008)	0.102 (±0.012)
German Credit	**0.793 (±0.022)**	0.769 (±0.029)	0.784 (±0.032)	0.781 (±0.030)	0.737 (±0.036)	0.775 (±0.024)	0.761 (±0.024)	0.775 (±0.022)
	0.266 (±0.012)	0.260 (±0.020)	0.250 (±0.021)	0.254 (±0.016)	0.254 (±0.012)	0.257 (±0.013)	0.245 (±0.015)	0.254 (±0.016)
	0.148 (±0.020)	0.144 (±0.022)	0.127 (±0.013)	0.130 (±0.010)	0.130 (±0.004)	0.13 (±0.008)	0.125 (±0.010)	0.130 (±0.011)
	0.081 (±0.021)	0.083 (±0.029)	0.063 (±0.009)	0.067 (±0.005)	0.065 (±0.005)	0.065 (±0.003)	0.062 (±0.006)	0.066 (±0.005)
Japanese Credit	0.923 (±0.038)	**0.931 (±0.042)**	0.922 (±0.038)	**0.930 (±0.040)**	0.904 (±0.046)	0.910 (±0.047)	0.897 (±0.040)	0.916 (±0.039)
	0.414 (±0.020)	**0.417 (±0.023)**	0.395 (±0.021)	0.395 (±0.025)	0.38 (±0.043)	0.383 (±0.053)	0.383 (±0.035)	0.393 (±0.032)
	0.256 (±0.068)	0.260 (±0.073)	0.198 (±0.015)	0.199 (±0.017)	0.188 (±0.023)	0.188 (±0.026)	0.200 (±0.016)	0.197 (±0.020)
	0.125 (±0.032)	0.141 (±0.047)	0.098 (±0.014)	0.095 (±0.010)	0.094 (±0.017)	0.091 (±0.018)	0.101 (±0.009)	0.099 (±0.015)
Autos MPG	**0.978 (±0.016)**	0.972 (±0.016)	0.955 (±0.014)	0.955 (±0.012)	0.95 (±0.014)	0.942 (±0.025)	0.951 (±0.014)	0.951 (±0.016)
	0.433 (±0.058)	0.421 (±0.016)	0.354 (±0.005)	0.354 (±0.005)	0.353 (±0.008)	0.350 (±0.011)	0.355 (±0.008)	0.354 (±0.008)
	0.238 (±0.079)	0.240 (±0.08)	0.180 (±0.006)	0.178 (±0.006)	0.177 (±0.005)	0.177 (±0.006)	0.175 (±0.004)	0.175 (±0.005)
	0.095 (±0)	0.095 (±0)	0.09 (±0.005)	0.09 (±0.006)	0.09 (±0.003)	0.089 (±0.006)	0.089 (±0.003)	0.091 (±0.004)
Ionosphere	**0.990 (±0.013)**	0.968 (±0.032)	0.930 (±0.015)	0.933 (±0.013)	0.872 (±0.081)	0.875 (±0.055)	0.754 (±0.048)	0.872 (±0.062)
	0.494 (±0.062)	0.392 (±0.086)	0.285 (±0.008)	0.288 (±0.005)	0.263 (±0.044)	0.263 (±0.029)	0.224 (±0.039)	0.256 (±0.040)
	0.156 (±0)	0.178 (±0.044)	0.143 (±0.004)	0.144 (±0.003)	0.131 (±0.024)	0.126 (±0.023)	0.105 (±0.026)	0.123 (±0.032)
	0.078 (±0)	0.078 (±0)	0.073 (±0.003)	0.072 (±0.003)	0.065 (±0.014)	0.062 (±0.016)	0.050 (±0.015)	0.060 (±0.018)
Breast Cancer Diagnosis	**0.994 (±0.008)**	**0.988 (±0.007)**	0.983 (±0.003)	0.982 (±0.004)	0.98 (±0.005)	0.982 (±0.008)	0.969 (±0.016)	0.983 (±0.005)
	0.595 (±0.047)	**0.528 (±0.005)**	0.505 (±0.013)	0.507 (±0.012)	0.502 (±0.013)	0.506 (±0.013)	0.501 (±0.013)	0.50 (±0.009)
	0.317 (±0.103)	0.453 (±0.036)	0.255 (±0.010)	0.253 (±0.007)	0.253 (±0.006)	0.253 (±0.012)	0.252 (±0.011)	0.259 (±0.010)
	0.134 (±0)	0.0135 (±0)	0.130 (±0.009)	0.125 (±0.008)	0.126 (±0.010)	0.130 (±0.010)	0.128 (±0.008)	0.131 (±0.011)
Breast Cancer Original	**0.995 (±0.006)**	0.993 (±0.006)	0.981 (±0.007)	0.983 (±0.006)	0.983 (±0.003)	0.984 (±0.004)	0.984 (±0.005)	0.984 (±0.004)
	0.559 (±0.010)	0.556 (±0.012)	0.536 (±0.022)	0.534 (±0.018)	0.537 (±0.017)	0.537 (±0.010)	0.531 (±0.011)	0.539 (±0.011)
	0.442 (±0.076)	0.407 (±0.078)	0.259 (±0.013)	0.265 (±0.012)	0.271 (±0.009)	0.265 (±0.008)	0.268 (±0.010)	0.264 (±0.008)
	0.146 (±0.010)	0.142 (±0.002)	0.131 (±0.011)	0.132 (±0.014)	0.137 (±0.012)	0.134 (±0.011)	0.136 (±0.012)	0.136 (±0.014)
Heart Disease	**0.917 (±0.032)**	0.911 (±0.034)	0.879 (±0.041)	0.885 (±0.037)	0.883 (±0.034)	0.891 (±0.032)	0.876 (±0.040)	0.892 (±0.044)
	0.416 (±0.027)	0.415 (±0.026)	0.358 (±0.037)	0.361 (±0.041)	0.371 (±0.035)	0.371 (±0.033)	0.366 (±0.039)	0.377 (±0.025)
	0.273 (±0.070)	0.272 (±0.070)	0.180 (±0.024)	0.176 (±0.027)	0.188 (±0.022)	0.187 (±0.020)	0.183 (±0.024)	0.189 (±0.017)
	0.118 (±0.017)	0.121 (±0.031)	0.090 (±0.015)	0.089 (±0.017)	0.094 (±0.011)	0.094 (±0.01)	0.092 (±0.012)	0.096 (±0.010)
Hepatitis	0.864 (±0.139)	**0.868 (±0.162)**	0.831 (±0.138)	0.798 (±0.168)	0.812 (±0.158)	0.803 (±0.166)	0.828 (±0.153)	0.826 (±0.152)
	0.572 (±0.240)	**0.625 (±0.230)**	0.544 (±0.205)	0.504 (±0.225)	0.526 (±0.248)	0.543 (±0.231)	0.561 (±0.234)	0.543 (±0.230)
	0.413 (±0.138)	0.346 (±0.162)	0.284 (±0.103)	0.263 (±0.115)	0.272 (±0.125)	0.280 (±0.116)	0.289 (±0.118)	0.281 (±0.116)
	0.269 (±0.190)	0.342 (±0.154)	0.143 (±0.051)	0.133 (±0.057)	0.137 (±0.062)	0.141 (±0.057)	0.145 (±0.058)	0.141 (±0.057)

Table 10 Comparison of TreeRank and several competitors on benchmark data. Total AUC and Local AUC for three different values $u = 20, 10, 5\%$ are given in that order. Standard type error $\hat{\sigma}^2$ for these values is indicated in parentheses.

HR@10% ; HR@20% AP	RF1$_{\text{SVM}}$	RF3$_{\text{CART}}$	AdaBoost	RankBoost	RankSVM	RankLS	P − norm push	KLR
Congressional Vote	100%; 100% 0.84	**100**%; **100**% **0.85**	**100**%; **100**% **0.85**	100%; 100% 0.84	100%; 100% 0.84	100%; 100% 0.84	100%; 100% 0.84	**100**%; ; 100% **0.85**
Australian Credit	100%; 100% 0.75	100%; 100% 0.76	100%; 100% 0.75	**100**%; **100**% **0.78**	100%; 100% 0.76	100%; 100% 0.76	100%; 100% 0.76	100%; 100% 0.77
German Credit	98%; 94% 0.86	**99**%; **92**% 0.86	95%; 92% 0.86	96%; 93% **0.87**	97%; 94% 0.84	**98**%; **94**% 0.86	93%; 92% 0.85	96%; 93% 0.84
Japanese Credit	95%; 95% **0.71**	97%; 95% **0.71**	95%; 95% 0.71	**97**%; **96**% **0.71**	97%; 91% 0.69	94%; 92% 0.70	95%; 93% 0.70	95%; 95% **0.71**
Autos MPG	**100**%; **100**% **0.80**	100%; 100% 0.79	100%; 100% 0.77	100%; 100% 0.77	100%; 100% 0.77	100%; 99% 0.76	100%; 100% 0.77	100%; 100% 0.78
Ionosphere	**100**%; **100**% **0.94**	100%; 99% 0.92	100%; 99% 0.92	100%; 100% 0.92	94%; 93% 0.87	89%; 94% 0.87	79%; 84% 0.78	94%; 92% 0.86
Breast Cancer Diagnosis	100%; 100% 0.70	100%; 100% 0.70	100%; 100% 0.69	100%; 100% 0.69	100%; 99% 0.70	100%; 100% 0.69	100%; 100% 0.68	**100**%; **100**% **0.71**
Breast Cancer Original	**100**%; **99**% **0.65**	99%; 99% ;0.64	97%; 98% 0.64	100%; 99% 0.63	100%; 99% 0.65	100%; 99% 0.65	100%; 98% 0.65	**100**%; **99**% **0.65**
Heart Disease	**97**%; **94**% 0.73	93%; 94% **0.75**	93%; 92% 0.73	93%; 90% 0.71	97%; 94% 0.73	97%; 92% 0.74	93%; 92% 0.85	97%; 92% **0.75**
Hepatitis	65%; 40% 0.33	85%; 42% 0.43	45%; 38% **0.48**	45%; 42% 0.33	65%; 37% 0.33	**85**%; **47**% 0.42	65%; 42% 0.37	65%; 42% 0.35

Table 11 Comparison of TreeRank and several competitors on benchmark data.Performance is provided here in terms of hit ratios for proportions 10% and 20% respectively (figures on top) and also in terms of average precision (figure on bottom).

References

1. S. Clémençon, N. Vayatis, (2009), Tree-based Ranking Methods, IEEE Transactions on Information Theory, 9:4316-4336.
2. S. Clémençon, M. Depecker, N. Vayatis, (2011), Adaptive Partitioning Schemes for Bipartite Ranking, Machine Learning Journal, 43(1):3169.
3. S. Clémençon, M. Depecker, N. Vayatis, (2009), Bagging Ranking Trees, In Proceedings of ICMLA.
4. S. Clémençon, N. Vayatis, (2010), Ranking Forests, To be published.
5. Y. Freund, R.Iyer, R.E. Schapire, Y. Singer, (2003), An Efficient Boosting Algorithm for Combining Preferences, Journal of Machine Learning Research, 4:933-969.
6. T. Hastie, R. Tibshirani, (1990), Generalized Additive Models, Chapman & Hall/CRC.
7. J. Zhu, T. Hastie, (2005), Kernel Logistic Regression and the Import Vector Machine, Journal of Computational and Graphical Statistics, 14(1):185-205.
8. J. Friedman, T. Hastie, R. Tibshirani, (2000), Additive Logistic Regression: a Statistical View of Boosting, Annals of Statistics, 28(2):337-407.
9. T. Joachims, (2002), Optimizing Search Engines using Clickthrough Data, Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 133-142.
10. T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, T. Salakoski, (2007), Learning to Rank with Pairwise Regularized Least-Squares, Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval, 27-33.
11. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, G. Hullender, (2005), Learning to Rank using Gradient Descent, Proceedings of the 22nd International Conference on Machine Learning, 89-96.
12. L. Dodd, M. Pepe, (2003), Partial AUC Estimation and Regression, Biometrics, 59(3).
13. S. Clémençon and N. Vayatis, (2007), Ranking the Best Instances, Journal of Machine Learning Research, 8:2671-2699.
14. S. Clémençon, N. Vayatis, (2008), Empirical performance maximization for linear rank statistics, Proceedings of NIPS'08, 305-312.
15. C. Rudin, (2009), The P-Norm Push: A Simple Convex Ranking Algorithm that Concentrates at the Top of the List, Journal of Machine Learning Research, 10:2233-2271.
16. S. Robertson, H. Zaragoza, (2007), On rank-based effectiveness measures and optimization, Information Retrieval, 10(3):321-339.
17. P. Bartlett, M. Jordan, J. McAuliffe, (2006), Convexity, classification and risk bounds, Journal of the American Statistical Association, 101(473):138-156.
18. P. Bartlett, A. Tewari, (2007), Sparseness vs Estimating Conditional Probabilities: Some Asymptotic Results, Journal of Machine Learning Research, 8:775-790.
19. D. Mease, A. Wyner, (2008), Evidence Contrary to the Statistical View of Boosting, Journal of Machine Learning Research, 9:131-156.
20. , Devroye, L., L. Györfi, G. Lugosi, (1996), A Probabilistic Theory of Pattern Recognition, Springer-Verlag.
21. S. Clémençon, N. Vayatis, (2010), Overlaying classifiers: a practical approach for optimal scoring, Constructive Approximation, 32(3):619-648.
22. S. Boucheron, O. Bousquet, G. Lugosi, (2005), Theory of classification: a survey of recent advances, ESAIM: Probability and Statistics, 9:323-375.
23. J. Hanley, J. McNeil, (1982), The Meaning and Use of the Area Under a ROC curve, Radiology, 143:29-36.
24. S. Clémençon, G. Lugosi, N. Vayatis, (2008), Ranking and Empirical Risk Minimization of U-statistics, The Annals of Statistics, 36:844-874.
25. N. Ailon and M. Mohri, (2010), Preference-based learning to rank, Machine Learning Journal, 80, 2:pp. 189–211.
26. L. Breiman, J. Friedman, R. Olshen, C. Stone, (1984), Classification, Regression Trees, Wadsworth and Brooks.
27. F.R. Bach, D.Heckerman, Eric Horvitz, (2006), Considering cost asymmetry in learning classifiers, Journal of Machine Learning Research, 7:1713-1741.