SortNet: Learning to Rank by a Neural Preference Function

Leonardo Rigutini, Tiziano Papini, Marco Maggini, *Member, IEEE*, and Franco Scarselli

***Abstract*— Relevance ranking consists in sorting a set of objects with respect to a given criterion. However, in personalized retrieval systems, the relevance criteria may usually vary among different users and may not be predefined. In this case, ranking algorithms that adapt their behavior from users' feedbacks must be devised. Two main approaches are proposed in the literature for learning to rank: the use of a scoring function, learned by examples, that evaluates a feature-based representation of each object yielding an absolute relevance score, a pairwise approach, where a preference function is learned to determine the object that has to be ranked first in a given pair. In this paper, we present a preference learning method for learning to rank. A neural network, the comparative neural network (CmpNN), is trained from examples to approximate the comparison function for a pair of objects. The CmpNN adopts a particular architecture designed to implement the symmetries naturally present in a preference function. The learned preference function can be embedded as the comparator into a classical sorting algorithm to provide a global ranking of a set of objects. To improve the ranking performances, an active-learning procedure is devised, that aims at selecting the most informative patterns in the training set. The proposed algorithm is evaluated on the LETOR dataset showing promising performances in comparison with other state-of-the-art algorithms.**

***Index Terms*— Learning to rank, neural networks, preference learning, selective learning.**

I. Introduction

THE STANDARD classification or regression tasks do not include all the possible supervised learning problems. Some applications require to compute other properties of the input objects, rather than real values or classes. For instance, in ranking tasks, the goal is to output a sorting of a set of items, and hence, the result depends on properties of the whole dataset instead of on the single patterns. In other cases, the main goal is to retrieve the top k objects without considering the ordering for the remaining ones. The differences among these classes of problems affect the properties of the required prediction, the representation of the data, and the type of the available supervision. For example, when an user indicates that an object is to be preferred with respect to another or that two objects should be in the same class, he/she does not assign a specific value to each individual item. In these cases, the given examples are in the form of relationships on pairs of objects and the supervision values are the result of a preference or similarity function applied to a pair of items. Two of these peculiar supervised learning tasks are *preference learning*, whose goal is to build a preference function, and *learning to rank*, where the goal is to produce an adapted sorting of a set of items [1]. These two research areas have been shown to have many interactions. In fact, preference learning techniques can be applied to ranking problems.

Actually, learning to rank algorithms can be used in a large set of application fields, including the implementation of user customized information retrieval systems, focused search engines and information filtering. However, despite this potentiality, some issues may limit their use in practical applications. In fact, supervised learning is based on the availability of examples, which usually are difficult to collect in real life environments. In particular, the application of learning to rank algorithms to real applications would need to define a feasible scheme to obtain the required feedback from users without being too invasive and increasing the interaction burden. Learning preferences from pairwise examples would require the collection of users' feedbacks as choices on pairs of items rather than complete rank lists or absolute score values. Pairwise comparisons are more natural for humans than providing absolute judgments about a set of items. For instance, the user of an information retrieval system can easily state whether a document, returned in response to a query, is less or more related to the query than another document. On the contrary, he/she would be hardly able to provide a precise ranking of the whole set of documents returned by the information retrieval engine. The use of pairwise supervisions is, thus, one of the main motivations of our proposal.

In this paper, we propose a preference learning approach to learning to rank, called *SortNet*. A neural network, the *comparative neural Network (CmpNN)*, is trained by examples to approximate a comparison function that specifies for an input pair of objects which is the preferred one. The CmpNN adopts a particular architecture specially designed to implement the symmetries naturally present in a preference function. The neural network can be embedded as a comparator into a sorting algorithm to rank a set of objects. To improve the training procedure, we propose an "active learning" procedure [2], [3], that aims at selecting the most informative patterns in the training set. This family of active learning techniques is known as "windowing" [4] or "incremental learning" [5], [6]. More precisely, at each iteration of the learning algorithm, the neural

Manuscript received November 9, 2009; revised March 31, 2011; accepted June 6, 2011. Date of publication July 18, 2011; date of current version August 31, 2011.

The authors are with the Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Siena, Siena 53100, Italy (e-mail: rigutini@dii.unisi.it; ppntizi@gmail.com; maggini@dii.unisi.it; franco@dii.unisi.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2011.2160875

comparator is trained only on a subset of the original training set. This subset is extended at each step by including the misclassified patterns. At the end of the iterative procedure, the comparator that obtains the best performance on the validation set is selected.

The neural network architecture we propose is shown to be able to deal also with nontransitive preference criteria making it suitable to be applied in cases where the provided feedbacks can be inspired by more general criteria. We also provide a theoretical result showing that the proposed architecture is able to approximate any preference function featuring the basic properties of symmetry, thus allowing the implementation of a very wide class of ranking functions. As far we know, these issues are not considered in related works. Finally, we propose a fast ranking algorithm trying to reduce the intrinsic quadratic complexity of the pairwise approach. The resulting algorithm compares favorably with the other top scoring algorithms proposed in the literature for this task.

The SortNet algorithm was evaluated on the Learning TO Rank (LETOR) dataset [7]. A comparison with other ranking algorithms (RankSupport Vector Machines (SVM) [8], RankBoost [9], FRank [10], ListNet [11], and AdaRank [12]), carried out using several performance measures, reports promising results. Actually, the evaluation shows that SortNet yields state-of-the-art performances and, in some cases, outperforms all the other methods.

This paper is organized as follows. In the following section, some related works are reviewed, while in Section III, we introduce the CmpNN architecture along with a brief mathematical description of its approximation properties. In Section IV, the SortNet algorithm is described, showing how the CmpNN is used in the sorting procedure. In Section V, we present the experimental results. Finally, in Section VI, some conclusions are drawn.

II. Related Works

Recently, the topic of *preference learning* received considerable attention in artificial intelligence and machine learning. For instance, in many tasks related to "Intelligent Agents" and "Planning," the best action to be taken at each step may not be unique, and could be appropriately chosen by exploiting an underlying "preference model." The approaches proposed in the literature vary from approximating the utility function of a single agent on the basis of a question-answer process, often referred to as "preference elicitation" [13], [14], to "collaborative filtering" [15]–[17], where the preferences of a given user are estimated from the preferences of other users. In this scenario, it is not surprising that, in recent years, automatic methods for learning and predicting preferences have been widely investigated in disciplines such as machine learning, knowledge discovery, and recommender systems.

In the following, the symbols $a \succ b$ and $a \prec b$ are used to indicate the preference relationship between objects. In particular, $a \succ b$ indicates that a is to be preferred to b while $a \prec b$ that b is to be preferred to a (it's the same as $b \succ a$).

As reviewed in [18], two different formulations of the preference learning problem have been proposed in the literature: *learning labels preference (LLP)* and *learning objects preference (LOP)*. The first one faces the problem of predicting a preference function between pairs of labels, given an input object. In more details, being L the label set, the goal of LLP is to learn a function $P : X \times L \times L \to \{\succ, \prec\}$ that estimates for each $L_1, L_2 \in L$ the values $L_1 \succ L_2$ and $L_1 \prec L_2$ given an instance $x \in X$. In other words, the prediction function decides if it is preferable to attach the label L_1 or the label L_2 to x.

In the LOP scenario, the goal is to learn a function that estimates the preference between two objects. Given a pair of objects $x, y \in X$, the goal is to learn a function $P : X \times X \to \{\succ, \prec\}$ which approximates the preference expressed by the user on the given pair, i.e., $x \succ y$ if x is to be preferred to y and $x \prec y$ vice versa. This last approach is also known as *pairwise preference learning* since it is based on pairs of objects.

The task of relevance ranking consists of sorting a set of objects with respect to a given criterion. In "Learning To Rank (L2R)" the criterion is not predefined but it has to be inferred from the users' feedbacks. Interestingly, even if in L2R the training set naturally consists of pairs of ordered objects, both LOP and LLP methods have been used for LETOR. The former case is the most straightforward, where both the preference model and the loss function, which is used for learning, are maps that depend on a pair of objects. On the other hand, in LLP approaches to ranking, the loss function is pairwise, but the model processes a single object to predict its score. As a matter of fact, most of the proposed approaches belong to the LLP class, probably because they are simpler to design. In particular, the approach of Herbrich *et al.* [19], which is based on a binary classifier, is considered the first work on preference learning and LETOR. Recently, an increasing number of algorithms have been proposed to learn a scoring function for ranking objects. Among the others, Freund *et al.* [9] proposed RankBoost, an algorithm based on a collaborative filtering approach.

In an earlier work [20], Tesauro proposed an approach in which a neural network is employed to model the underlying preference function. Similarly to the approach proposed in this paper, the neural network is trained on pairs of examples using the back-propagation algorithm. The model, however, significantly differs from the method proposed in this paper for the weight-sharing scheme and for the training set construction procedure. A more detailed comparison between the two architectures is reported in Section III. Recently, this approach has been used by Burges *et al.* [21] to develop RankNet. In [8], the authors propose a pairwise learning approach to train a SVM model (SVMRank), while AdaRank [12] uses an AdaBoost scheme to learn the preference function. For the first time, the ListNet algorithm [11] extends the pairwise approach to a listwise approach, in which lists of objects are used as instances for learning. Most of the LETOR algorithms have been tested on applications in information retrieval and search engines for which there is an increasing interest [7], [22].

In the following, a connectionist approach based on LOP will be presented. It will be shown that the method has the theoretical advantage of being able to implement a larger

class of rankings both with respect to LLP methods and the neural model proposed in [20]. Moreover, when compared with SVMs and boosting, it retains the common advantages and disadvantages of artificial neural networks. Here, it is only worth mentioning that real life ranking applications usually require fast sorting algorithms and layered neural networks tend to be faster in the testing phase both with respect to boosting and SVMs. In fact, the cost of the test phase in SVMs depends on the number of support vectors, which usually grows along with the training set dimension, whereas for layered neural networks it only depends on the chosen number of hidden neurons. Similarly, neural networks require a smaller number of units when compared with the biases used by boosting [23].

III. Preference Learning with the CmpNN

The method presented in this paper is a pairwise preference learning approach, where the preference function is implemented by a multilayered feed-forward neural network, called *CmpNN*. A CmpNN has three-layers, corresponding to an input layer with $2d$ units (d is the dimension of the object feature vectors), one hidden layer, and an output layer with two units. In fact, a CmpNN processes a representation of two input objects x, y and produces an estimate of the "evidence" of the relationships $x \succ y$ and $y \succ x$ (i.e., $x \prec y$). Formally, we assume that the input object can be described by a d-dimensional feature vector, i.e., $x, y \in R^d$. Thus, the input to the neural network is the concatenation of the two representations $[\mathbf{x}, \mathbf{y}] = [x_1, \ldots, x_d, y_1, \ldots, y_d]$. The outputs will be denoted by $N_{\succ}([\mathbf{x}, \mathbf{y}])$ and $N_{\prec}([\mathbf{x}, \mathbf{y}])$, respectively, where $N_{\succ}([\mathbf{x}, \mathbf{y}])$ estimates the evidence of $x \succ y$ and $N_{\prec}([\mathbf{x}, \mathbf{y}])$ the evidence of $x \prec y$. The neural network can be trained with the standard back-propagation algorithm once appropriate targets are provided for the two outputs. For each pair of inputs $[\mathbf{x}, \mathbf{y}]$, the assigned target is

$$\mathbf{t} = [t_1 \ t_2]' = \begin{cases} [1 \ 0]' & \text{if } x \succ y \\ [0 \ 1]' & \text{if } x \prec y \end{cases} \quad (1)$$

and the error is measured by the squared error function

$$E([\mathbf{x}, \mathbf{y}], \mathbf{t}) = (t_1 - N_{\succ}([\mathbf{x}, \mathbf{y}]))^2 + (t_2 - N_{\prec}([\mathbf{x}, \mathbf{y}]))^2. \quad (2)$$

After training, the model can be used to predict the preference relationship for an input pair of objects x, y as

$$\begin{cases} x \succ y, & \text{if } N_{\succ}([\mathbf{x}, \mathbf{y}]) \geq N_{\prec}([\mathbf{x}, \mathbf{y}]) \\ x \prec y, & \text{if } N_{\prec}([\mathbf{x}, \mathbf{y}]) \geq N_{\succ}([\mathbf{x}, \mathbf{y}]). \end{cases}$$

It is worth noticing that the operators implemented by the preference function realize a correct total ordering of the objects only if the following properties hold.

1) Reflexivity: both $x \succ x$ and $x \prec x$ hold.
2) Equivalence between $\prec$ and $\succ$: if $x \succ y$ then $y \prec x$ and vice versa, if $y \succ x$ then $x \prec y$.
3) Anti-symmetry: if $x \succ y$ and $x \prec y$ then $x = y$.
4) Transitivity: if $x \succ y$ and $y \succ z$, then $x \succ z$ (similarly for the $\prec$ relation).

In our approach, the reflexivity and the equivalence between $\prec$ and $\succ$ are ensured by the particular architecture adopted for the network. The anti-symmetry property fails only if the two outputs are equal, i.e., $N_{\succ}([\mathbf{x}, \mathbf{y}]) = N_{\prec}([\mathbf{x}, \mathbf{y}])$, and $x \neq y$ holds, but, since the outputs are real numbers, such an event is very unlikely. Finally, the transitivity property is generally hard to be guaranteed by a pairwise preference learning approach and our method does not overcome this limitation. However, the experimental results will show that the transitivity relation can be learned and the performance is only marginally affected by the initial allocation of the patterns (see Section V-B).

Interestingly, non-transitive preference functions can still be adopted to sort a set of objects. Actually, most of the sorting algorithms work also with non-transitive comparators provided that we accept the limitation that the result may depend on the initial permutation of the objects. In fact, classical sorting algorithms compare only a small subset of all the possible pairs and return an ordering consistent with those comparisons. If an algorithm knows, from the comparator response, that $x \succ y$ and $y \succ z$ hold, then $x \succ z$ is usually assumed without a further comparison. Thus, even if theoretically an inconsistent comparison operator does not define any total ordering, sorting algorithms automatically chose a consistent subset of the available comparisons in order to define the final ordering. However, as an alternative approach, the preference function could be also used (for instance in an one-against-all championship) to obtain a score for each object that, in turns, provides an unique sorting of the objects (see Section V-B). Interestingly, a given sorting of a set of objects always corresponds to a transitive comparator (an object is preferable over another one if it appears first in the sorting). This may lead to the erroneous conclusion that transitive and non-transitive criteria are somehow equivalent. However, when a non-transitive comparator is used, the mutual position of two objects may depend also on the other objects of the set: for instance, using an one-against-all championship an object position depends on the comparisons with all the other objects. Such a behavior cannot be realized by any transitive comparator.

Example 1: Non-transitive criteria are not only a theoretical matter of research, but they can be naturally used in practical applications. For example, let us consider the following preference function:

$$x \succ y \iff \#_{(|x_i|>|y_i|)_{i=1}^d} > \#_{(|x_i|<|y_i|)_{i=1}^d}$$

where x_i (y_i) indicates the i^{th} feature of the pattern $x(y)$ and the value $\#_{(|x_i|>|y_i|)_{i=1}^d}$ stands for the number of features of x which have a value greater than y. This preference criterion is non-transitive: for example, with $x = [1, 1, 2]$, $y = [2, 2, 0]$, and $z = [0, 3, 1]$, we have $y \succ x$, $z \succ y$, a $x \succ z$. Moreover, this criterion is realistic, since it corresponds to the case in which a set of votes is available for each item and the preference is given to the one having the larger number of greater scores. It is worth mentioning that this criterion can be embedded into an LOP method, but not into an LLP approach. A straightforward proof of this claim is obtained observing that LLP approaches can only provide sortings that

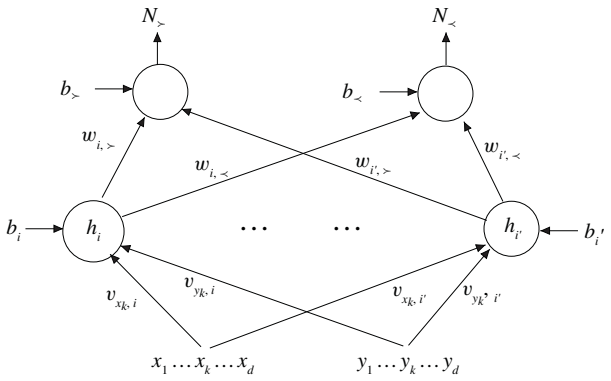

Fig. 1. CmpNN architecture.

correspond to transitive comparators, because they assign a numerical score to each single object.

A. CmpNN Architecture

The CmpNN architecture adopts a weight-sharing technique in order to ensure that the reflexivity and the equivalence between $\prec$ and $\succ$ hold (see Fig. 1). Let us assume that $v_{x_k,i}$ ($v_{y_k,i}$) denotes the weight of the connection from the input node x_k (y_k)[1] to the i-th hidden node, $w_{i,\succ}$, $w_{i,\prec}$ represent the weights of the connections from the i-th hidden to the output nodes, b_i is the bias of i-th hidden and $b_\succ, b_\prec$ are the output biases. For each hidden neuron i, a dual neuron i' exists whose weights are shared with i according to the following schema:

1) $v_{x_k,i'} = v_{y_k,i}$ and $v_{y_k,i'} = v_{x_k,i}$ hold, i.e., the weights from x_k, y_k to i are swapped in the connections to i';
2) $w_{i',\succ} = w_{i,\prec}$ and $w_{i',\prec} = w_{i,\succ}$ hold, i.e., the weights of the connections from the hidden i to the outputs $N_\succ$, $N_\prec$ are swapped in the connections leaving from i';
3) $b_i = b_{i'}$ and $b_\succ = b_\prec$ hold, i.e., the biases are shared between the dual hiddens i and i' and between the outputs $N_\succ$ and $N_\prec$.

In order to study the properties of the above described architecture, let us denote by $h_i([\mathbf{x},\mathbf{y}])$ the output of the i-th hidden neuron when the network is fed on the pair x, y. Then, using the weight-sharing rule in point 1, we have

$$\begin{aligned} h_i([\mathbf{x},\mathbf{y}]) &= \sigma\left(\sum_k (v_{x_k,i}\, x_k + v_{y_k,i}\, y_k) + b_i\right) \\ &= \sigma\left(\sum_k (v_{x_k,i'}\, y_k + v_{y_k,i'}\, x_k) + b_i\right) \\ &= h_{i'}([\mathbf{y},\mathbf{x}]) \end{aligned}$$

where σ is the activation function used for the hidden units. Moreover, from the sharing rules in points 2 and 3 and the previous equality $h_i([\mathbf{x},\mathbf{y}]) = h_{i'}([\mathbf{y},\mathbf{x}])$, it follows:

$$\begin{aligned} &N_\succ([\mathbf{x},\mathbf{y}]) \\ &\quad= \sigma\left(\sum_{i,i'} (w_{i,\succ}\, h_i([\mathbf{x},\mathbf{y}]) + w_{i',\succ}\, h_{i'}([\mathbf{x},\mathbf{y}]) + b_\succ\right) \\ &\quad= \sigma\left(\sum_{i,i'} (w_{i',\prec}\, h_{i'}([\mathbf{y},\mathbf{x}]) + w_{i,\prec}\, h_i([\mathbf{y},\mathbf{x}]) + b_\prec\right) \\ &\quad= N_\prec([\mathbf{y},\mathbf{x}]). \end{aligned}$$

Thus, we proved

$$N_\succ([\mathbf{x},\mathbf{y}]) = N_\prec([\mathbf{y},\mathbf{x}]) \tag{3}$$

which implies that the equivalence between $\prec$ and $\succ$ is satisfied. Moreover, it immediately follows $N_\succ([\mathbf{x},\mathbf{x}]) = N_\prec([\mathbf{x},\mathbf{x}])$, which implies that also the reflexivity property is fulfilled.

Other neural architectures could be exploited to implement the preference function as, for example, a model where one output value is produced by the network and the other one is derived from the assumption $N_\succ([\mathbf{x},\mathbf{y}]) = 1 - N_\succ([\mathbf{y},\mathbf{x}])$ [i.e., $N_\succ([\mathbf{x},\mathbf{y}]) = 1 - N_\prec([\mathbf{x},\mathbf{y}])$]. To motivate the choice of using two distinct outputs, let us consider the case of a neural network with one output. In this case, the neural network is trained to approximate the $x \succ y$ preference score using the pair $[\mathbf{x},\mathbf{y}]$ as input, i.e., $N_\succ([\mathbf{x},\mathbf{y}])$. However, in order to decide which relationship holds for an input pair, we would need to evaluate also the evidence of the reverse relationship $x \prec y$ [i.e., $N_\prec([\mathbf{x},\mathbf{y}]) = N_\succ([\mathbf{y},\mathbf{x}])$] since there is no a-priori constraint between the values $N_\succ([\mathbf{x},\mathbf{y}])$ and $N_\prec([\mathbf{x},\mathbf{y}])$. The final decision is then taken by comparing the two values $N_\succ([\mathbf{x},\mathbf{y}])$ and $N_\succ([\mathbf{y},\mathbf{x}])$. In other words, given a pair of objects x and y to be compared, both the $[\mathbf{x},\mathbf{y}]$ and $[\mathbf{y},\mathbf{x}]$ pairs must be processed by the network to yield the predicted relationship ($x \succ y$ or $x \prec y$). When the one-output network adopts the weight-sharing scheme of Fig. 1 in the hidden layer, then it is equivalent to a CmpNN in which the two values $N_\succ([\mathbf{x},\mathbf{y}])$ and $N_\prec([\mathbf{x},\mathbf{y}])$ are evaluated in two distinct forward phases. Thus, the proposed two-output network with the proposed weight-sharing scheme can learn and compute the preference function more directly and enforcing the required function symmetries.

Notice that the CmpNN architecture is an extension of the model proposed in [20] and employed in [21] for developing the RankNet method. In particular, the neural network in [20] consists of two equal subnets (i.e., sharing their weights) coupled only through the output layer. Given a pair of objects $[\mathbf{x},\mathbf{y}]$, a subnet processes $\mathbf{x}$ while the other subnet processes $\mathbf{y}$. Each subnetwork produces an independent vectorial representation of the input element and subsequently the two representations are processed by the output layer to yield the overall decision. Actually, the network proposed in [20] is equivalent to a CmpNN, where the weights connecting the left input/output to the right hiddens and the weights connecting the right input/output to the left hiddens are set to 0, i.e., $v_{x_k,i'} = v_{y_k,i} = 0$ and $w_{i',\succ} = w_{i,\prec} = 0$. Hence, the approach proposed in [20], [21], mixes both the LOP and LLP schemes.

[1]Here, with an abuse of notation, x_k represents the node that is fed with the k-th feature of x, and y_k that fed on the k-th feature of the second object y.

Even if the models are trained using a single target for each pair of objects, however, the whole architecture is comprised of subnetworks that accept in input only one object at a time. On the other hand, the proposed CmpNN is based on a pure LOP approach, since it actually processes pairs of objects. Moreover, the architectures in [20], [21], are less general even in terms of their approximation capability. Actually, they cannot implement non-transitive preference functions, since, so as LLP approaches, they assign a numerical score to each single object. Thus, for example, the criterion based on voting described in Example 1 cannot be implemented by RankNet.

B. Approximation Capability

Another interesting question that arises about the CmpNN architecture regards whether it can implement all the preference functions that may be found in applications or whether it has some limitations due to the weight-sharing mechanism. In the following, it will be proved that a CmpNN can approximate most of the practically useful functions that satisfy the constraint (3) up to any degree of precision. This result is a direct consequence of the universal approximation capability of three-layer neural networks [24], [25].

Formally, let $\mathcal{F}$ be a set of functions $f : R^n \to R^m$, $\|\cdot\|$ be a norm on $\mathcal{F}$ and σ be an activation function. The universal approximation property for three-layer networks, with activation function σ in the hidden layer and linear activation function in the output layer,[2] states that for any function $f \in \mathcal{F}$ and any real $\varepsilon > 0$, there exists a network that implements a function $N : R^n \to R^m$ such that $\|f - N\| \leq \varepsilon$ holds. Different versions of this property have been proved, according to the adopted function set $\mathcal{F}$ (e.g., the set of the continuous or the measurable functions), the norm $\|\cdot\|$ (e.g., the infinity norm or a probability norm) and the activation function σ (e.g., a sigmoidal or a threshold activation function) [26]. The following theorem demonstrates that the network with the proposed weight sharing maintains the universal approximation property provided that we restrict the attention to the functions that satisfy the constraint (3).

Theorem 1: Let $\mathcal{F}$ be a set of functions $f : R^{2d} \to R^2$, $\|\cdot\|$ be a norm on $\mathcal{F}$ and σ be an activation function such that the corresponding three-layer neural network class has the universal approximation property on $\mathcal{F}$. Let us denote by $\overline{\mathcal{F}}$ the set of the functions k that belong to $\mathcal{F}$ and for any input $[\mathbf{x}, \mathbf{y}]$, fulfill

$$k_{\succ}([\mathbf{x}, \mathbf{y}]) = k_{\prec}([\mathbf{y}, \mathbf{x}]) \tag{4}$$

where $k_{\succ}, k_{\prec}$ denote the two components of the outputs of k. Then, for any function $k \in \overline{\mathcal{F}}$ and any real $\varepsilon > 0$, there exists a three–layer neural network satisfying the weight-sharing schema defined in points 1–4 such that

$$\|k - h\| \leq \varepsilon$$

holds, where $h : R^{2d} \to R^2$ is the function implemented by the neural network.

[2]For the sake of simplicity, we consider only networks with linear activation function in the output layer. However, the results can be easily extended to a wider class of output activation functions.

Proof: By the universal approximation hypothesis, there exists a three-layer neural network $\mathcal{A}$ that implements a function $r : R^{2d} \to R^2$ such $\|(k/2) - r\| \leq (\varepsilon/2)$, where k and ε are the function and the real of the hypothesis, respectively. Then, we can construct another network $\mathcal{B}$ that has twice the number of the hidden nodes of $\mathcal{A}$. The indexes of hidden nodes of $\mathcal{B}$ are partitioned into pairs i, i'. The neurons with index i are connected to the input and to the outputs with the same weights as in $\mathcal{A}$, i.e., $\mathcal{B}$ contains $\mathcal{A}$ as a subnetwork. On the other hand, the weights of the hidden neurons with index i' are defined following the weight-sharing rules in points 1, 2, and 3. The neurons in this latter set are contained in a subnetwork $\mathcal{A}'$, that due to the weight-sharing scheme, implements a function $r'([\mathbf{x}, \mathbf{y}])$ such that $r_{\prec}([\mathbf{x}, \mathbf{y}]) = r'_{\succ}([\mathbf{y}, \mathbf{x}])$ and $r_{\succ}([\mathbf{x}, \mathbf{y}]) = r'_{\prec}([\mathbf{y}, \mathbf{x}])$. Finally, the function computed by the neural network $\mathcal{B}$ is $h([\mathbf{x}, \mathbf{y}]) = r([\mathbf{x}, \mathbf{y}]) + r'([\mathbf{x}, \mathbf{y}])$. Notice that, by construction, the network $\mathcal{B}$ satisfies all the rules of points 1–3 and it is a good candidate to be the network of the thesis.

Since, the output function h implemented by $\mathcal{B}$ is given by the sum of the two components, then

$$\|k - h\| = \|k - (r + r')\| \leq \left\|\frac{k}{2} - r\right\| + \left\|\frac{k}{2} - r'\right\| \leq \varepsilon$$

where we exploited the property of (4) for the function k and the definition of r' to bound the last term. ■

IV. LETOR with SortNet

In the proposed method, called SortNet, the CmpNN is embedded, as a comparison function, into a classical sorting algorithm, that in our experiments is based on a binary search tree. In the test phase, the algorithm sorts the objects by calling the CmpNN every time a comparison is needed. Thus, SortNet can order a set of objects in $O(n \log n)$ operations, which is the lowest computational complexity for a ranking algorithm.

A. Learning Algorithm

In the learning phase, the CmpNN is trained on a dataset composed of pairs of objects for which the value of the preference function is given. The learning set must be chosen with a particular attention due to the peculiarities of the given problem. In fact, the quality of a ranked list is a property that can be measured only considering all the involved patterns, i.e., counting the pairs of the objects that have been correctly ordered. On the other hand, the comparative neural network is trained using the square error function in (2) that forces the network outputs to be close to the desired targets on each single pair of objects. When the network produces a perfect classification of any input pair, also the ranking algorithm yields a perfect sorting of the objects, but, in general, the optimization of the square error does not necessarily correspond to a good ranking. Actually, a good training procedure should use, in some way, a measure of the quality of the global ordering. Moreover, if the number n of the objects is large, then the inclusion of all the $[n(n-1)/2]$ pairs into the learning set can make the training very slow. On the other hand, an indiscriminate use of all the available

patterns may not be useful and it may even decrease the learning performance of a neural network. Such a situation arises, for example, when the learning set is unbalanced and there are a large number of patterns of the same class in a small region of the input domain. In this case, the learning procedure is focused on the most common patterns and tends to disregard all the other patterns. In fact, during the training phase, a pattern gives a contribution to the square error of (2) even when it is already correctly classified by the neural network, f.i., when the desired order is $x \succ y$ and the network outputs satisfy $N_{\succ}([\mathbf{x}, \mathbf{y}]) > N_{\prec}([\mathbf{x}, \mathbf{y}])$. As a consequence, a set of incorrectly classified patterns can be hidden by a large set of correctly predicted patterns. Such a problem is particularly important in the considered domain, since the most frequent features x are copied in all the pairs $[\mathbf{x}, \mathbf{y}]$ where the corresponding objects are included. Thus, it is necessary to design a criterion to select the most informative pairs to be included into the training set.

These ideas have inspired the SortNet training algorithm. It consists of an iterative procedure, where, at each iteration, the learning set is extended by the addition of new examples. The goal is to select, from the labeled data, a subset of the most informative examples to be used in the training process. Each learning set is used to train a different CmpNN and the result of the iterative procedure is the CmpNN that achieves the best performance according to a measure of the ranking quality. More precisely, let us assume that a function $RankQuality$ is available to measure the quality of a given ranking. Some possible choices for this measure will be described later in this section. Moreover, let T and V be two sets of objects with associated rankings R_T and R_V, respectively. At each iteration i, a new CmpNN C^i is trained using two sets $P_T^i \subset T \times T \times \{\prec, \succ\}$ and $P_V^i \subset V \times V \times \{\prec, \succ\}$, that contain the training and validation pairs,[3] respectively. In particular, P_T^i contains the pairs that are used to train the neural network, while P_V^i is used as validation set for the training procedure. At the beginning of Algorithm 1, P_T^0 and P_V^0 are empty (lines 3 and 4) and the initial CmpNN is a network with randomly initialized weights (line 5). At the first iteration ($i = 0$) the training procedure is not executed and hence, the training and validation sets for step 1 (P_T^1 and P_V^1) are selected using the randomly initialized CmpNN. Then, at each iteration $i \geq 1$, a new CmpNN C^i is randomly initialized and trained using P_T^i and P_V^i (line 8) for a predefined number of epochs. The selected neural network is the one that achieves the best result, over all the epochs, on the validation set P_V^i. Once the new CmpNN C^i has been trained, it is employed to sort the training set objects T (line 10), yielding either the sorting R_T^i of the training examples and the set E_T^i of the object pairs that are mis-classified in the comparisons performed by the used sorting algorithm. More precisely, the calls of the sorting algorithm to the comparator C^i are monitored registering the considered pairs and the corresponding outputs of C^i. Those pairs for which the comparator makes a wrong prediction are included into E_T^i. Similarly, C^i is used to rank the validation set (line 11), producing a sorting R_V^i and a set of mis-classified pairs E_V^i. Then, the set of mis-classified pairs E_T^i is added to the current learning set P_T^i, removing the eventual duplicates (line 17), while the pairs in the set E_V^i are inserted into P_V^i (line 18). These sets are the training set and the validation set, respectively, of the next iteration. The procedure is repeated until a maximum number of iterations are reached (max_iter) or until there is no difference in the sets P_T^i and P_V^i between two consecutive iterations, i.e., $P_T^{i+1} = P_T^i$ and $P_V^{i+1} = P_V^i$ (lines 19–21). Finally, the output of the SortNet training algorithm is the CmpNN C^* that yields the best performance on the validation set during the iterations, where the performance is measured using the function $RankQuality$ (line 12–16).

Algorithm 1 SortNet learning algorithm

```
1: T ← Set of training objects
2: V ← Set of validation objects
3: P_T^0 ← ∅;
4: P_V^0 ← ∅;
5: C^0 ← RandomWeightNetwork();
6: for i = 0 to max_iter do
7:   if i ≥ 1 then
8:     C^i ← TrainAndValidate(P_T^i, P_V^i);
9:   end if
10:  [E_T^i , R_T^i] ← Sort(C^i, T);
11:  [E_V^i , R_V^i] ← Sort(C^i, V);
12:  score ← RankQuality(R_V^i);
13:  if score > best_score then
14:    best_score ← score;
15:    C* ← C^i;
16:  end if
17:  P_T^{i+1} ← P_T^i ∪ E_T^i;
18:  P_V^{i+1} ← P_V^i ∪ E_V^i;
19:  if P_T^{i+1} = P_T^i and P_V^{i+1} = P_V^i then
20:    return C*;
21:  end if
22: end for
23: return C*;
```

It is worth noticing that the idea of building the learning set by choosing the most informative samples from a set of available patterns is basically an *active learning* technique and, in particular, it can be seen to implement a kind of *incremental learning* approach [5], [6]. These methods are known also as *Windowing* [4] and they are usually applied for tasks where all the patterns in the learning set are given with their associated targets before starting the training procedure. Moreover, the inclusion of the mis-classified pairs at each iteration is a mechanism by which the learning is focused on the areas of the input domain that are more prone to yield errors. On the other hand, the mis-classified patterns are expected to be the closest patterns to the separation curve defined by the classifier. Thus, the whole algorithm can be interpreted as a rough attempt to increase the margin between the mistaken patterns and the separation curve. The proposed method is intuitively related to some boosting methods [27], [28], where the weights of

[3]Notice that for the particular network architecture all the supervisions can be provided in the form $x \succ y$ (or $x \prec y$).

the learning set patterns are modified according to the errors of the classifier in the previous iteration and whose relations with the margin have been proved [29]. Similarly to boosting methods, the procedure could increase the noise level in the training data due to the selection criterion that favors those examples on which the learner shows worse generalization [30]. However, the experimental results did not evidence any evident drawback in using this scheme for this particular task, whereas the proposed selection criterion was proved to be quite effective in making the training successful. The proposed selection method is also somehow related to the use of the hinge loss function[4] in kernel-based models, that have been demonstrated to provide good results in classification and preference learning [19], [31].

Finally, the validation set P_V is in some sense biased because it only contains a subset of the preference pairs. However, preliminary experimental sessions showed that the incremental choice of P_V does not affect the performances significantly.

B. Measures of the Ranking Quality

Several ranking measures can be used to implement the function *RankQuality* in the SortNet algorithm. In this paper, we will consider the three measures proposed for the LETOR benchmark [7].

1) **Precision at position n (P@n)**: This value measures the relevance of the top n results of the ranking list with respect to a given query

$$P@n = \frac{\text{relevant docs in top } n \text{ results}}{n}.$$

2) **Mean average precision (MAP)**: Given a query q, the average precision is

$$AP_q = \frac{\sum_{n=1}^{N_q} P@n \cdot rel(n)}{N_q}$$

where N_q is the number of documents in the result set of q and $rel(n)$ is 1 if the n-th document in the ordering is relevant and 0 otherwise. Thus, AP_q averages the values of $P@n$ over the positions n of the relevant documents. Finally, the MAP value is computed as the mean of AP_q over the set of all queries.

3) **Normalized discount cumulative gain (NDCG@n)**: This measure exploits an explicit rating of the documents in the list. The NDCG value of a ranking list at position n is calculated as

$$NDCG@n \equiv Z_n \left((2^{r_1} - 1) + \sum_{j=2}^{n} \frac{2^{r_j} - 1}{log(j)} \right)$$

where $r_j \geq 0$ is the rating of the j-th document (smaller values indicate less relevance) and Z_n is a normalization factor chosen such that the ideal ordering (the DGC-maximizing one) gets a NDCG@n score of 1.

[4]A (squared) hinge loss function H is defined as $H([\mathbf{x}, \mathbf{y}]) = 0$, if $[\mathbf{x}, \mathbf{y}]$ is correctly classified, and $H([\mathbf{x}, \mathbf{y}]) = E([\mathbf{x}, \mathbf{y}])$, otherwise (see 2).

The measure can be chosen according to the user's requirements. For example, if an information retrieval system displays only the best k documents, then $P@k$ might be the preferred measure. If, in the other hand, the user is interested in the overall precision of the results, then MAP might be the best choice. Obviously, the selection of the measure to use in the *RankQuality* test affects the performance evaluation of the model. For instance, by selecting the $P@k$ measure, we will obtain a model yielding good values of $P@k$ but not necessarily optimal values for MAP.

V. Experimental Results

Two different sets of experiments have been carried out to evaluate both the CmpNN and the SortNet algorithm. First, an artificial dataset was used to compare the learning capabilities of the CmpNN model with respect to the architecture proposed in [20] for approximating a non–transitive preference function. Then, the performances of the proposed technique were assessed on a benchmark widely used in the learning to rank literature.

A. Non-Transitive Preference Function

In order to verify whether the CmpNN is a more general architecture than that described in [20], the two models were compared on the task of learning a non–transitive preference function. More precisely, an artificial dataset was generated following the voting criterion described in Example 1. In particular, each pattern is represented as a 7-D vote vector whose component values have been randomly drawn in [0, 1]. To assign targets, given a pair of patterns, the preference is given to the object having the majority of higher votes. Table I reports and compares the classification accuracy achieved by the two models. The results have been averaged using 5 fold cross validation and for each trial we randomly selected 10 000 pairs for training, 4000 pairs for validation, and 6000 pairs for testing.

The t-student test (with p-value less than 0.05) proves that the model proposed in this paper is able to learn a non-transitive preference function with a significant improvement with respect to the model proposed in [20].

Such a result confirms the different approximation capabilities of the two models. It can also be noticed that the variance and the over-fitting (the difference between the performance on training and test sets) is significantly smaller for the CmpNN. This behavior can be explained observing that the CmpNN can implement the unique function that perfectly solves the problem, whereas all the functions that can be obtained by the model in [20] are far from the target. Thus, it is likely that the local minima encountered during the training of the CmpNN are closer to the goal and less widespread.

B. Experiments on LETOR Benchmark

The SortNet algorithm has been experimentally evaluated on the LETOR dataset [7], a package of benchmarks for LETOR, released by Microsoft Research Asia and available

TABLE I
COMPARISON OF THE CLASSIFICATION ACCURACY (IN PERCENTAGE) BETWEEN THE CmpNN AND THE MODEL PROPOSED IN [20]

Hidden neurons		CmpNN	Neural Network of [20]
10	Test	**87.46 ± 1.67**	50.75 ± 29.11
	Train	**90.27 ± 1.41**	50.29 ± 29.53
	Validation	**87.09 ± 1.42**	49.56 ± 29.00
20	Test	**88.25 ± 0.56**	60.71 ± 12.15
	Train	**90.73 ± 0.39**	61.43 ± 12.06
	Validation	**88.02 ± 0.38**	60.02 ± 12.22
30	Test	**88.21 ± 0.95**	61.36 ± 19.27
	Train	**90.65 ± 1.78**	61.56 ± 20.24
	Validation	**87.63 ± 1.27**	62.02 ± 18.58
40	Test	**87.95 ± 0.74**	68.14 ± 7.15
	Train	**90.97 ± 1.85**	69.55 ± 7.68
	Validation	**87.93 ± 0.80**	69.14 ± 6.60
50	Test	**88.42 ± 1.08**	68.36 ± 12.01
	Train	**91.40 ± 2.06**	69.64 ± 11.95
	Validation	**88.07 ± 1.36**	68.88 ± 11.90

TABLE II
ASSIGNMENT OF THE LABELS $\succ$ AND $\prec$ FROM THE RELEVANCE DEGREE

		y		
		R	PR	NR
x	R	–	$\succ$	$\succ$
	PR	$\prec$	–	$\succ$
	NR	$\prec$	$\prec$	–

TABLE III
ACCURACY, PRECISION AND RECALL (IN PERCENTAGE) OF THE PREFERENCE PREDICTION BY THE CmpNN ON THE TEST SET

Hiddens		**TD2003**	**TD2004**	**OHSUMED**
10	Accuracy	**92.93 ± 3.11**	97.22 ± 2.25	**94.38 ± 2.73**
	Precision	**92.95 ± 2.40**	97.15 ± 1.50	**94.93 ± 1.82**
	Recall	**92.95 ± 2.41**	97.15 ± 1.50	**94.93 ± 1.82**
20	Accuracy	90.66 ± 3.80	**97.97 ± 1.98**	94.28 ± 1.74
	Precision	90.11 ± 2.20	**97.64 ± 1.32**	94.11 ± 1.17
	Recall	90.10 ± 2.20	**97.65 ± 1.33**	94.19 ± 1.17
30	Accuracy	88.65 ± 3.94	96.63 ± 2.12	93.92 ± 1.45
	Precision	89.43 ± 2.29	97.08 ± 1.41	93.95 ± 0.96
	Recall	89.43 ± 2.29	97.08 ± 1.41	93.95 ± 0.97

TABLE IV
SELECTED NUMBER OF HIDDEN NEURONS FOR THE DIFFERENT DATASETS AND RANKING QUALITY MEASURES

Dataset		Hidden neurons
TD2003	SortNet MAP	20
	SortNet P@10	30
	SortNet NDCG@10	20
TD2004	SortNet MAP	20
	SortNet P@10	10
	SortNet NDCG@10	10
OHSUMED	SortNet MAP	20
	SortNet P@10	10
	SortNet NDCG@10	10

on the Web.[5] The task considered in LETOR is that of LETOR, according to their relevance, the documents returned by information retrieval systems in response to a query. In this paper, we consider the version 2.0 of LETOR, which contains three benchmarks: the OHSUMED dataset, the 2003 and the 2004 TREC collections, denoted by TD2003 and TD2004, respectively. The TD2003 dataset consists of 50 sets of documents, each one containing 1000 documents returned in response to a query (i.e., 50 different queries). Similarly, TD2004 contains 75 sets of 1000 documents, each corresponding to a different queries. Each query-document pair is represented by 44 values that include several features commonly used in information retrieval. A label is assigned to each document to specify whether it is relevant (R) or not (NR) with respect to the given query. For all queries, the relevant documents are roughly 1% of the whole set of the available documents. The OHSUMED collection [32] is a subset of the medical publication repository MEDLINE consists of 106 sets of documents, with associated relevance degrees, that have been returned in response to queries. The relevance degree is provided by humans and consists in one of the following categories: relevant R, possibly relevant PR, and non-relevant NR. The documents are represented using a set of 25 features.

A preprocessing procedure has been adopted to constrain the features into the range $[-1, 1]$. More precisely, let N_i be the number of documents returned in response to the i-th query and let us denote the normalized r-th feature of the j-th document of the i-th query by $\hat{x}^i_{j,r}$, then

$$\hat{x}^i_{j,r} = \frac{x^i_{j,r} - \mu^i_r}{max_{s=1,\ldots,N_i} |x^i_{s,r}|} \tag{5}$$

holds, where $x^i_{j,r}$ is the original document feature and $\mu^i_r = (\sum_{s=1}^{N_i} x^i_{s,r}/N_i)$.

In the reported experiments, the training set was created by sampling pairs of documents from a query and assigning the target according to the relevance degree provided in the datasets. More precisely, given a pair of documents x, y, the target is assigned according to (1) and Table II. The PR relevance degree is not considered when using the TD2003 and the TD2004 datasets. The pairs containing two documents with the same relevance degree were not used for learning.

For a more fair comparison, the LETOR benchmark comes along with a splitting into 5 folds, where each fold is further divided into a test, a validation, and a training set. In all the experiments, a 5 fold cross validation procedure was adopted based on the provided splitting.

The experimentation has been carried out considering different settings, each useful to evaluate and analyze a particular aspect of the proposed scheme. First, the classification capabilities of the CmpNN were evaluated on LETOR dataset, measuring the accuracy in the prediction of the preference. Then, the complete SortNet algorithm was tested comparing its

[5]Available at http://research.microsoft.com/users/LETOR.

TABLE V
RESULTS ON TD2003 MEASURED BY $NDCG@n$ AND $P@n$

	NDCG@n										P@n									
	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
RankBoost	0.26	0.28	0.27	0.27	0.28	0.28	0.29	0.28	0.28	0.29	0.26	0.27	0.24	0.23	0.22	0.21	0.21	0.19	0.18	0.18
RankSVM	0.42	0.37	0.38	0.36	0.35	0.34	0.34	0.34	0.34	0.34	0.42	0.35	0.34	0.3	0.26	0.24	0.23	0.23	0.22	0.21
Frank-c19.0	0.44	0.39	0.37	0.34	0.33	0.33	0.33	0.33	0.34	0.34	0.44	0.37	0.32	0.26	0.23	0.22	0.21	0.21	0.2	0.19
ListNet	0.46	0.43	0.41	0.39	0.38	0.39	0.38	0.37	0.38	0.37	0.46	0.42	0.36	0.31	0.29	0.28	0.26	0.24	0.23	0.22
AdaRank MAP	0.42	0.32	0.29	0.27	0.24	0.23	0.22	0.21	0.2	0.19	0.42	0.31	0.27	0.23	0.19	0.16	0.14	0.13	0.11	0.1
AdaRank NDCG	0.52	0.41	0.37	0.35	0.33	0.31	0.3	0.29	0.28	0.27	0.52	0.4	0.35	0.31	0.27	0.24	0.21	0.19	0.17	0.16
SortNet MAP	**0.58**	**0.46**	**0.43**	**0.4**	**0.4**	**0.39**	**0.4**	**0.39**	**0.39**	**0.39**	**0.58**	**0.45**	**0.39**	**0.33**	**0.31**	**0.29**	**0.28**	**0.27**	**0.26**	**0.24**
SortNet P@10	**0.44**	**0.42**	**0.38**	**0.37**	**0.37**	**0.37**	**0.37**	**0.37**	**0.36**	**0.36**	**0.44**	**0.4**	**0.33**	**0.3**	**0.29**	**0.27**	**0.26**	**0.25**	**0.23**	**0.22**
SortNet NDCG@10	**0.5**	**0.42**	**0.41**	**0.39**	**0.39**	**0.39**	**0.39**	**0.39**	**0.38**	**0.38**	**0.5**	**0.41**	**0.38**	**0.34**	**0.31**	**0.3**	**0.29**	**0.27**	**0.26**	**0.24**

TABLE VI
RESULTS ON TD2004 MEASURED BY $NDCG@n$ AND $P@n$

	NDCG@n										P@n									
	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
RankBoost	0.48	0.47	0.46	0.44	0.44	0.45	0.46	0.46	0.46	0.47	0.48	0.45	0.4	0.35	0.32	0.3	0.29	0.28	0.26	0.25
RankSVM	0.44	0.43	0.41	0.41	0.39	0.4	0.41	0.41	0.41	0.42	0.44	0.41	0.35	0.33	0.29	0.27	0.26	0.25	0.24	0.23
FRank	0.44	0.47	0.45	0.43	0.44	0.45	0.46	0.45	0.46	0.47	0.44	0.43	0.39	0.34	0.32	0.31	0.3	0.27	0.26	0.26
ListNet	0.44	0.43	0.44	0.42	0.42	0.42	0.43	0.45	0.46	0.46	0.44	0.41	0.4	0.36	0.33	0.31	0.3	0.29	0.28	0.26
AdaRank MAP	0.41	0.39	0.4	0.39	0.39	0.4	0.4	0.4	0.4	0.41	0.41	0.35	0.34	0.3	0.29	0.28	0.26	0.24	0.23	0.22
AdaRank NDCG	0.36	0.36	0.38	0.38	0.38	0.38	0.38	0.38	0.39	0.39	0.36	0.32	0.33	0.3	0.28	0.26	0.24	0.23	0.22	0.21
SortNet MAP	**0.47**	**0.47**	**0.46**	**0.46**	**0.45**	**0.45**	**0.45**	**0.46**	**0.47**	**0.48**	**0.47**	**0.43**	**0.38**	**0.36**	**0.33**	**0.3**	**0.28**	**0.28**	**0.27**	**0.26**
SortNet P@10	**0.39**	**0.43**	**0.42**	**0.44**	**0.44**	**0.45**	**0.46**	**0.46**	**0.46**	**0.47**	**0.39**	**0.4**	**0.36**	**0.36**	**0.33**	**0.32**	**0.31**	**0.28**	**0.27**	**0.26**
SortNet NDCG@10	**0.43**	**0.47**	**0.46**	**0.47**	**0.46**	**0.47**	**0.47**	**0.48**	**0.48**	**0.49**	**0.43**	**0.43**	**0.39**	**0.37**	**0.34**	**0.32**	**0.31**	**0.29**	**0.28**	**0.27**

performance with other state-of-the-art models proposed in the literature. The role of the incremental learning procedure and of the transitivity was also investigated. Finally, some ideas about the use of different sorting algorithms are sketched.

1) Preference Function Approximation: The accuracy of the classification obtained by the CmpNN after training was measured by considering the preferences for a pair of documents that were correctly and not correctly predicted on the examples in the test set. The results on the three benchmarks, with different number of hidden neurons, are reported in Table III. The high accuracy proves that the network was able to learn a good approximation for the preference function. This preliminary result was important to assess the viability of the proposed approach.

2) Ranking Results: A good ranking is characterized by the presence of the documents belonging to the relevant class (R) in the top positions, followed by the possibly–relevant documents (PR) and by the non-relevant ones (NR). The SortNet algorithm has been compared on ranking tasks with other methods, i.e., RankSVM [8], RankBoost [9], FRank [10], ListNet [11], AdaRank MAP and AdaRank NDCG [12], using the measures proposed for the LETOR benchmark [7] (see Section IV-B): P@n, MAP, NDCG@n.

In the following experiments, the network was trained by Algorithm 1, with a sorting algorithm based on a binary search tree. The variable max_iter was set to 30, a value determined heuristically as the iteration after which the learning does not show significant improvements in average. In three different sets of experiments, MAP, $NDCG@10$, and $P@10$ were used by the $RankQuality$ function;, we refer to the three different configurations as *SortNet MAP*, *SortNet NDCG@10*, and *SortNet P@10*, respectively.

In order to select the best configuration, three CmpNN architectures having 10, 20, and 30 hidden neurons, respectively, were evaluated and the one achieving the best result on the validation set was chosen. Table IV reports the selected number of hidden neurons for each dataset and rank quality function.

Tables V–VII show the performances on TD2003, TD2004, and OHSUMED, respectively. Each table displays the results obtained measuring the ranking quality both with $NDCG@n$ and $P@n$, varying n in the range $[1, 10]$ (see the columns). The performances measured by MAP are presented in Table VIII. It can be noticed that, on the TD2003 and TD2004 datasets, the SortNet algorithm outperforms all the other methods using the MAP measure and for several values of n, also using the $NDCG@n$ and $P@n$ measures. On the OHSUMED dataset the results are, in most of the cases, comparable with the best algorithms.

Moreover, a further interesting analysis regards the effect of the measure adopted as $RankQuality$. In fact, even if the performances could be expected to be higher when $RankQuality$ is the same measure as the one used to evaluate the results, this effect is only evident for TD2003 in the case of MAP (see Table VIII). Probably, in several cases, the difference between the considered measures is not significant enough to evidence a clear separation in the results. On the other hand, observing Tables V–VII, we can notice that SortNet MAP is often better than SortNet P@10, when n is small ($n << 10$), whereas there

TABLE VII
RESULTS ON OHSUMED MEASURED BY $NDCG@n$ AND $P@n$

	NDCG@n										P@n									
	n=1	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
RankBoost	0.5	0.48	0.47	0.46	0.45	0.44	0.44	0.44	0.43	0.44	0.6	0.6	0.59	0.56	0.54	0.52	0.52	0.5	0.49	0.5
RankSVM	0.5	0.48	0.46	0.46	0.46	0.45	0.45	0.44	0.44	0.44	0.63	0.62	0.59	0.58	0.58	0.56	0.54	0.52	0.52	0.51
Frank-c4.2	0.54	0.51	0.5	0.48	0.47	0.46	0.45	0.45	0.44	0.44	0.67	0.62	0.62	0.58	0.56	0.53	0.51	0.5	0.5	0.49
ListNet	0.52	0.5	0.48	0.47	0.47	0.45	0.45	0.45	0.45	0.45	0.64	0.63	0.6	0.58	0.57	0.54	0.53	0.52	0.51	0.51
AdaRank.MAP	0.54	0.5	0.48	0.47	0.46	0.45	0.44	0.44	0.44	0.44	0.66	0.6	0.58	0.57	0.54	0.53	0.51	0.5	0.5	0.49
AdaRank.NDCG	0.51	0.47	0.46	0.46	0.44	0.44	0.44	0.44	0.44	0.44	0.63	0.6	0.57	0.56	0.53	0.53	0.52	0.51	0.5	0.49
MHR-BC	0.55	0.49	0.49	0.48	0.47	0.46	0.45	0.44	0.44	0.44	0.65	0.61	0.61	0.59	0.57	0.55	0.53	0.51	0.5	0.5
SortNet MAP	**0.52**	**0.48**	**0.47**	**0.46**	**0.45**	**0.45**	**0.44**	**0.44**	**0.44**	**0.44**	**0.63**	**0.58**	**0.57**	**0.56**	**0.55**	**0.55**	**0.53**	**0.52**	**0.51**	**0.5**
SortNet P@10	**0.49**	**0.43**	**0.44**	**0.43**	**0.42**	**0.42**	**0.42**	**0.42**	**0.42**	**0.42**	**0.62**	**0.57**	**0.57**	**0.55**	**0.53**	**0.51**	**0.51**	**0.51**	**0.5**	**0.49**
SortNet NDCG@10	**0.52**	**0.48**	**0.45**	**0.45**	**0.44**	**0.44**	**0.44**	**0.44**	**0.43**	**0.43**	**0.64**	**0.61**	**0.57**	**0.57**	**0.55**	**0.54**	**0.53**	**0.52**	**0.5**	**0.49**

TABLE VIII
MAP RESULTS ON TD2003, TD2004, AND OHSUMED

	TD2003	TD2004	OSHUMED
RankBoost	0.212	0.384	0.44
RankSVM	0.256	0.35	0.447
Frank-c4.2	0.245	0.381	0.446
ListNet	0.273	0.372	0.45
AdaRank.MAP	0.137	0.331	0.442
AdaRank.NDCG	0.185	0.299	0.442
MHR-BC	NA	NA	0.44
SortNet MAP	**0.307**	**0.391**	**0.442**
SortNet P@10	**0.256**	**0.381**	**0.438**
SortNet NDCG@10	**0.297**	**0.402**	**0.44**

is not a large gap with respect to NDCG@10. This result can be explained observing that the MAP measure gives a larger weight to the errors in the initial positions of the ranking, whereas $P@10$ gives the same importance to all the errors in the first 10 positions.

3) Evaluation of the Incremental Learning Procedure: In order to evaluate the role played by the active learning algorithm, which increases the training set step by step, the CmpNN was also trained using the complete training and validation sets. Then, the CmpNN was used to sort the documents so as in SortNet. Fig. 2 shows the results when the performance is measured by MAP, the performances measured by $P@n$ and $NCDG@n$ are similar. The non-incremental version of SortNet is denoted as *CmpNN*. The results confirm that the active learning procedure consistently improves the quality of the ranking and is crucial in the design of SortNet in most cases.

Fig. 3(a)–(c) shows the history of the scores (the values in line 12 of Algorithm 1, that measure the ranking quality), obtained at each iteration of a run of the learning algorithm for SortNet MAP. The behavior for the SortNet P@10 and $NCDG@n$ versions is similar. The plot clearly suggests that the largest score is obtained before the maximum number of iterations is reached. Fig. 3(d) depicts the evolution of the training and validation set sizes during learning. Interestingly, after 30 iterations the training set contains on average only 17 829 pairs out of the 87 514 pairs that compose the complete validation set (20%) and the training set contains 47 602 pairs out of 26 2543 (19%). The behavior of the algorithm in other runs is similar, proving that the training technique effectively takes advantage of the incremental learning procedure.

4) Evaluation of the Transitivity: As previously mentioned, we cannot ensure that the operator implemented by the CmpNN satisfies the transitivity property. However, non-transitive preference criteria can arise from data in real applications and the case reported in Section V-A is an example. When dealing with a non-transitive comparison function, the results produced by the sorting algorithm may depend on the initial permutation of the objects. This effect is clear, for example, in a tree-based sorting algorithm where the final result depends on the actual comparisons that are made while building the tree. However, it is important to notice that, also when employing a non-transitive preference function, a sorting algorithm ends up yielding a final ordering. The only problem is that the resulting ordering is not unique and generally inconsistent with the provided comparison function, since it may happen that in the ordering $[a, b, c]$ it holds $a \succ b$, $b \succ c$ but $a \prec c$ with the given non-transitive preference criterion. However, we were interested to evaluate if, in the case of potentially transitive comparison functions, this property could be inferred from examples by the CmpNN. We considered a large amount of triples of objects randomly sampled from the learning and test sets. For each triple x, y, z, the CmpNN is fed on the three pairs $[\mathbf{x}, \mathbf{y}]$, $[\mathbf{y}, \mathbf{z}]$, and $[\mathbf{z}, \mathbf{x}]$ and the coherence of the outputs is evaluated. For example, if the CmpNN outputs $x \succ y$, $y \succ z$, and $z \succ x$, then the prediction is incoherent, whereas if the CmpNN outputs $x \succ y$, $y \succ z$, and $x \succ z$, then the prediction is consistent. The percentages of the consistent triples are reported in Table X. Interestingly, the accuracy of a random predictor can be easily computed as 75% and it can be used as a baseline for a comparison. The results prove that the preference function satisfies the transitivity property in most of the cases, showing that the learning procedure is able to capture this property that cannot be easily embedded into the network architecture.

Moreover, the effect of the presence of mis-compared patterns on the final ranking was also studied. As already mentioned, most of the sorting algorithms can still operate

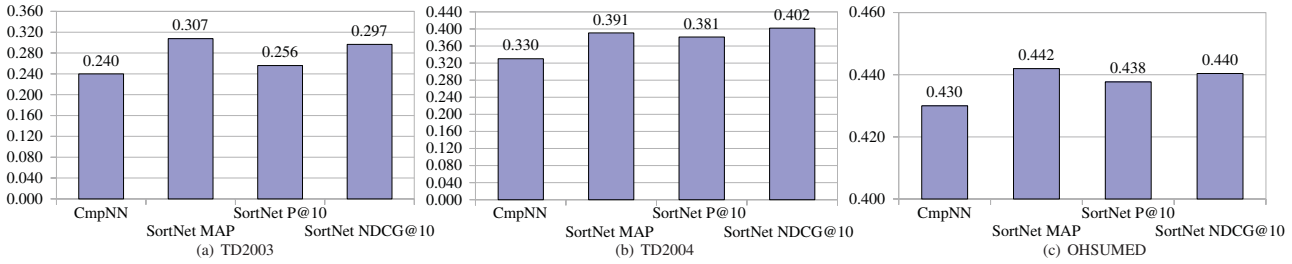

Fig. 2. Results, measured by MAP (a) achieved on TREC2003, (b) TD2004, and (c) OSHUMED. Obtained with the CmpNN trained without the active-learning technique and with the different versions of SortNet.

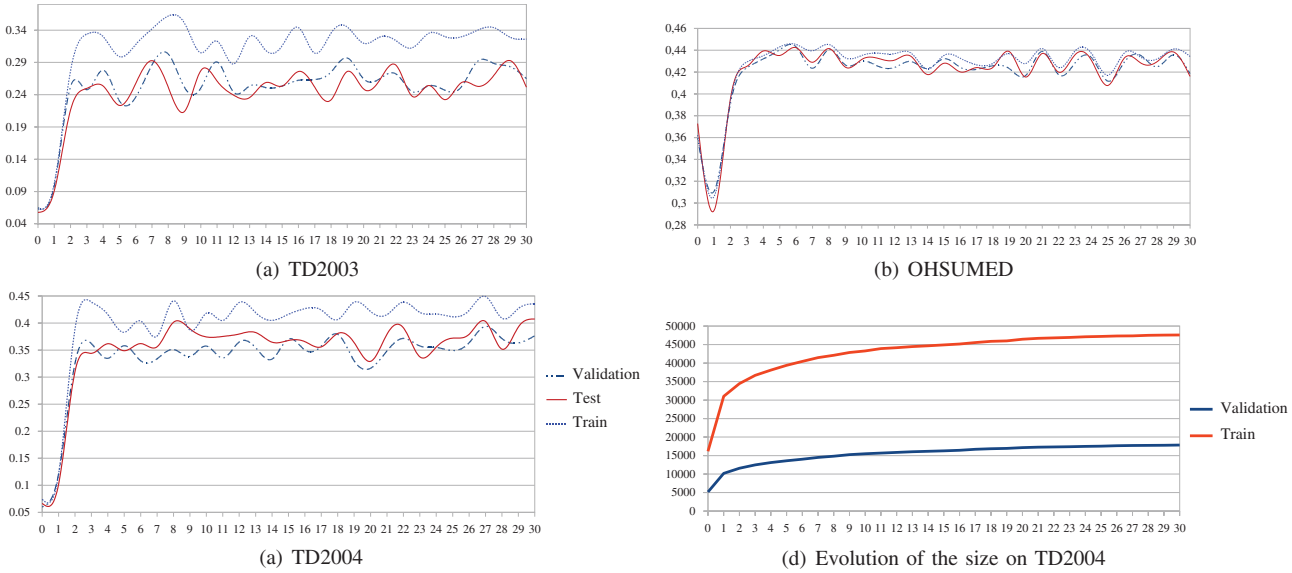

Fig. 3. Evolution of the MAP measure during the training process on (a) TD2003, (b) OHSUMED, (c) TD2004, and (d) evolution of the size of the training and validation sets in the SortNet learning procedure when processing the TD2004 dataset.

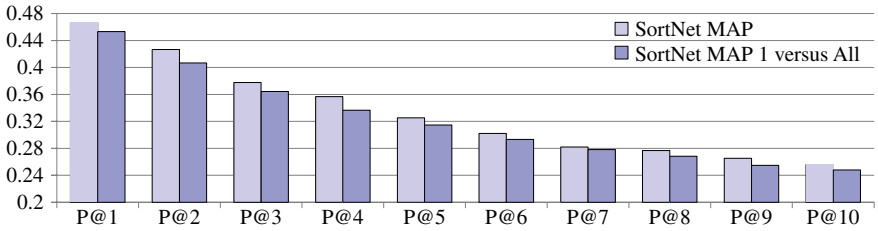

Fig. 4. Original Sortnet versus Sortnet 1vsAll scheme on TD2004.

TABLE IX
STANDARD DEVIATION OF THE *MAP*, *P@n*, AND *NDCG@n* SCORES WHEN TESTING THE SORTNET ALGORITHM USING 20 DIFFERENT INITIAL SHUFFLINGS OF THE TEST SET

MAP	0.0062									
	n = 1	**n = 2**	**n = 3**	**n = 4**	**n = 5**	**n = 6**	**n = 7**	**n = 8**	**n = 9**	**n = 10**
NDCG@n	0.0170	0.0144	0.0138	0.0104	0.0103	0.0116	0.0106	0.0096	0.0091	0.0101
P@n	0.0170	0.0166	0.0161	0.0103	0.0105	0.0115	0.0102	0.0094	0.0088	0.0085

with a comparison operator that gives inconsistent comparisons (such as the violation of the transitivity property), but the results depend on the initial permutation of the objects and on the order with which the pairs are compared by the algorithm (i.e., on the specific sorting algorithm that is used). Thus, in a further set of trials, the trained CmpNN was used to rank 20 different initial shufflings of each set of documents.

Table X shows that standard deviations for the resulting values for the P@n, NDCG@n, and MAP measures. These values are quite low, which proves that the transitivity has a small effect on the final ranking. In fact, two factors

TABLE X
EVALUATION OF THE TRANSITIVITY PROPERTY: THE PERCENTAGES OF THE TRIPLES CONSISTENTLY CLASSIFIED

	Hiddens	TD2003	TD2004	OHSUMED
Training set	10	97.93 ± 1.51	98.98 ± 0.75	99.77 ± 0.24
	20	$\mathbf{97.98 \pm 1.27}$	$\mathbf{99.02 \pm 0.21}$	$\mathbf{99.88 \pm 0.09}$
	30	97.95 ± 1.34	99.00 ± 0.37	99.79 ± 0.17
Test set	10	98.79 ± 0.96	99.02 ± 0.83	99.69 ± 0.19
	20	98.87 ± 0.96	99.08 ± 0.85	99.79 ± 0.21
	30	$\mathbf{98.94 \pm 0.99}$	$\mathbf{99.13 \pm 0.88}$	$\mathbf{99.85 \pm 0.23}$

may contribute to this behavior. Formerly, the number of inconsistently ordered pairs is small, as previously shown, and not all of them are considered by the sorting algorithm which compares only $O(n \log n)$ object pairs out of $n(n-1)/2$. Thus, the effect of inconsistent pairs may be hidden by the other errors. Moreover, only the first objects in the ranked list have a big impact on $P@n$, $NDCG@n$, and MAP scores, thus even when different sortings are produced as a consequence of the initial permutations of the objects, the resulting score may be similar.

5) Other Sorting Algorithms: The embedding of the preference function into a sorting algorithm is not the only method available to sort the objects. Actually, the preference function could be also used to obtain, for each object a, a global score s_a that measures the relevance of a. For example, exploiting an one-against-all scheme, s_a can be computed as the number of the pairs (a, b) for which $a \prec b$ holds. The scores provide a consistent ordering of the objects, but their computation has a large impact on the computational complexity of the algorithm, because $(n(n-1)/2)$ pairs have to be considered. Although $O(n^2)$ is not a prohibitive cost for normal computing systems, in real–time environments with a large number of results per query and a high rate of queries per minute, it can significantly degrade the performance. Moreover, the results in Fig. 4 prove that such an approach does not provide a significant advantage over the proposed method. In [33], a wide set of other methods to compute s_a were evaluated and similar conclusion were drawn.

VI. CONCLUSION

In this paper, a new neural architecture for learning a preference function (CmpNN) and an adaptive ranking algorithm (SortNet) have been proposed. More precisely, the proposed CmpNN has a particular architecture designed to implement the symmetries naturally present in a preference function. The properties of this architecture have been studied and links and comparisons with similar models, previously proposed in the literature, have been described.

Furthermore, a new adaptive sorting algorithm which employs the CmpNN was proposed. The SortNet algorithm exploits an iterative procedure aimed at selecting the most informative patterns from the training set. The algorithm has been evaluated using the LETOR datasets, a suite of benchmarks widely used in the LETOR research area. The results show that the proposed method compares favorably with other state-of-the-art techniques. We discussed the influence of the errors of the CmpNN in the final ranking, reporting a detailed analysis on different factors that may vary the final ranking. We also showed the effectiveness of the proposed incremental learning procedure employed for training the CmpNN in the SortNet algorithm.

Matters of future research include a wider experimentation and the application of the method to different kinds of preference learning problems.

REFERENCES

[1] T.-Y. Liu, "Learning to rank for information retrieval," *Found. Trends Inform. Retrieval*, vol. 3, no. 3, pp. 225–331, Mar. 2009.

[2] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 129–145, Mar. 1996.

[3] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *J. Mach. Learn. Res.*, vol. 2, no. 1, pp. 45–66, Mar. 2002.

[4] J. R. Quinlan, "Learning efficient classification procedures and their application to chess end-games," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds. San Mateo, CA: Morgan Kaufmann, 1983, pp. 463–482.

[5] D. A. Cohn, "Neural network exploration using optimal experiment design," *Neural Netw.*, vol. 9, no. 6, pp. 1071–1083, Aug. 1996.

[6] A. Engelbrecht and I. Cloete, "Incremental learning using sensitivity analysis," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 2. Washington D.C., Jul. 1999, pp. 1350–1355.

[7] T. Qin, T.-Y. Liu, J. Xu, and H. Li, "LETOR: A benchmark collection for research on learning to rank for information retrieval," *Inform. Retrieval*, vol. 13, no. 4, pp. 346–374, Aug. 2010.

[8] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon, "Adapting ranking SVM to document retrieval," in *Proc. 29th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, Seattle, WA, Aug. 2006, pp. 186–193.

[9] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *J. Mach. Learn. Res.*, vol. 4, no. 6, pp. 933–969, Dec. 2003.

[10] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma, "FRank: A ranking method with fidelity loss," in *Proc. 30th Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval*, Amsterdam, The Netherlands, Jul. 2007, pp. 383–390.

[11] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *Proc. 24th Int. Conf. Mach. Learn.*, Corvallis, OR, Jun. 2007, pp. 129–136.

[12] J. Xu and H. Li, "AdaRank: A boosting algorithm for information retrieval," in *Proc. 30th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, Amsterdam, The Netherlands, Jul. 2007, pp. 391–398.

[13] C. Boutilier, "A POMDP formulation of preference elicitation problems," in *Proc. 18th Nat. Conf. Artif. Intell.*, Edmonton, AB, Canada, Jul. 2002, pp. 239–246.

[14] A. Blum, J. Jackson, T. Sandholm, and M. Zinkevich, "Preference elicitation and query learning," *J. Mach. Learn. Res.*, vol. 5, pp. 649–667, Dec. 2004.

[15] M. Morita and Y. Shinoda, "Information filtering based on user behavior analysis and best match text retrieval," in *Proc. 17th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, Dublin, Ireland, Jul. 1994, pp. 272–281.

[16] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proc. 14th Conf. Uncertain. Artif. Intell.*, Madison, WI, Jul. 1998, pp. 43–52.

[17] B. M. Sarwar, J. A. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl, "Using filtering agents to improve prediction quality in the GroupLens research collaborative filtering system," in *Proc. ACM Conf. Comp. Supp. Cooperat. Work*, Seattle, WA, Nov. 1998, pp. 345–354.

[18] J. Fürnkranz and E. Hüllermeier, *Preference Learning*, 1st ed. Berlin, Germany: Springer-Verlag, 2011.

[19] R. Herbrich, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer, "Learning a preference relation for information retrieval," in *Proc. 15th AAAI Conf. Workshop Learn. Text Categ.*, Madison, WI, Jul. 1998, pp. 83–86.

[20] G. Tesauro, "Connectionist learning of expert preferences by comparison training," in *Proc. Adv. Neural Inf. Process. Syst.*, Denver, CO, Dec. 1988, pp. 99–106.

[21] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proc. 22nd Int. Conf. Mach. Learn.*, Bonn, Germany, Aug. 2005, pp. 89–96.
[22] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview," in *Proc. JMLR Workshop*, vol. 14. Haifa, Israel, Jun. 2011, pp. 1–24.
[23] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 930–945, May 1993.
[24] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Netw.*, vol. 2, no. 3, pp. 183–192, May 1989.
[25] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control. Sign. Syst.*, vol. 2, no. 4, pp. 303–314, Dec. 1989.
[26] F. Scarselli and A. C. Tsoi, "Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results," *Neural Netw.*, vol. 11, no. 1, pp. 15–37, Jan. 1998.
[27] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
[28] R. E. Schapire, "The boosting approach to machine learning: An overview," in *Nonlinear Estimation and Classification* (Lecture Notes in Statistics), vol. 171, D. Denison, M. Hansen, C. Holmes, B. Mallick, and B. Yu, Eds. New York: Springer-Verlag, 2003, pp. 149–172.
[29] S. Rosset, J. Zhu, and T. Hastie, "Boosting as a regularized path to a maximum margin classifier," *J. Mach. Learn. Res.*, vol. 5, no. 5, pp. 941–973, Dec. 2004.
[30] J. Fürnkranz, "Integrative windowing," *J. Artif. Intell. Res.*, vol. 8, pp. 129–164, May 1998.
[31] R. Herbrich, T. Graepel, and K. Obermayer, "Support vector learning for ordinal regression," in *Proc. 9th Int. Conf. Artif. Neural Netw.*, vol. 1. Edinburgh, U.K., Sep. 1999, pp. 97–102.
[32] W. Hersh, C. Buckley, T. J. Leone, and D. Hickam, "OHSUMED: An interactive retrieval evaluation and new large test collection for research," in *Proc. 17th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, Dublin, Ireland, Jul. 1994, pp. 192–201.
[33] L. Rigutini, T. Papini, M. Maggini, and M. Bianchini, "A neural network approach for learning object ranking," in *Proc. 18th Int. Conf. Artif. Neural Netw.*, Prague, Czech Republic, Sep. 2008, pp. 899–908.

Leonardo Rigutini received the Laurea degree in computer engineering and the Ph.D. degree in information engineering from the University of Siena, Siena, Italy, in 2001 and 2005, respectively.

He was a Visiting Scholar at the University of Illinois at Chicago, Chicago, working on a Cross-Lingual Text Classification Project, in 2004. He was an Associate Researcher at the Department of Information Engineering in the University of Siena from 2005 to 2010, and was a Co-Founder of QuestIt S.R.L, Siena, a spin-off company that develops technologies for question answering, natural language processing, text mining, and information extraction. He participated in several research projects funded by companies and public administrations and he has been teaching several academic courses. His current research interests include the automatic analysis of electronic text documents, and in particular text classification and clustering, text representation and feature selection techniques. Further research topics are semantic regularization, learning to rank, natural language processing, and web mining.

Tiziano Papini received the Laurea degree in computer engineering from the University of Siena, Siena, Italy, in February 2007. His Masters thesis was on the design and experimentation of a novel biometric system for password checking. Since 2007, he is a Ph.D. degree student with the Doctorate School on Information Engineering, University of Siena.

His current research interests include the study and the development of machine learning models for document ranking and document retrieval, mainly based on neural networks approaches.

Marco Maggini (M'99) received the Laurea degree (*cum laude*) in electronic engineering and the Ph.D. degree in computer science and control systems from the University of Florence, Florence, Italy, in 1991 and 1995, respectively.

He became an Assistant Professor with the Department of Information Engineering in the University of Siena, Siena, Italy, in February 1996, where, since March 2001, he has been an Associate Professor. His current research interests include machine learning, neural networks and kernel machines, integration of symbolic and sub-symbolic knowledge, web mining, search engine technology, and pattern recognition.

Dr. Maggini served as an Associate Editor of the Association for Computing Machinery's Transactions on Internet Technology (ACM TOIT) and he has been a member of the program committee of several international conferences and workshops. He has a been Guest Editor of a special issue of the ACM TOIT on machine learning and the Internet. He is a member of the International Association for Pattern Recognition-International Conference, of the Italian Artificial Intelligence Society, and of the IEEE Computer and Computational Intelligence Societies.

Franco Scarselli received the Laurea degree with honors in computer science from the University of Pisa, Pisa, Italy, in 1989, and the Ph.D. degree in computer science and automation engineering from the University of Florence, Florence, Italy, in 1994.

He has been supported by foundations of private and public companies and by a post-doctoral grant of the University of Florence. In 1999, he moved to the University of Siena, Siena, Italy, where he was initially a Research Associate and is currently an Associate Professor at the Department of Information Engineering. He is the author of more than 50 journal and conference papers and has been involved in several research projects, founded by public institutions and private companies, focused on software engineering, machine learning, and information retrieval. His current research interests include the field of machine learning with a particular focus on adaptive processing of data structures, neural networks, approximation theory, image understanding, information retrieval, and web applications.