SortNet: Learning To Rank by a Neural Preference Function

Tiziano Papini
papinit@dii.unisi.it

Dipartimento di Ingegneria dell'Informazione
via Roma 56, Siena, Italy

Abstract

The problem of relevance ranking consists in sorting a set of objects with respect to a given criterion. Since users may prefer different relevance criteria, the ranking algorithms should be adaptable to the users' needs. Two main approaches are proposed in the literature for learning to rank: the use of a score function, learned by examples, that evaluates the properties of each object yielding an absolute relevance score which can be used to order the objects; a pairwise approach, where a preference function is learned using pairs of objects to define which one has to be ranked first. In this paper, we present a preference learning method for learning to rank. A neural network, the "Comparative Neural Network (CmpNN)", is trained by examples to approximate a comparison function that specifies for each pair of objects which is the preferred one. The CmpNN adopts a particular architecture designed to implement the symmetries naturally present in a preference function. The trained preference function is embedded as the comparator into a classical sorting algorithm to provide the ranking of a set of objects. To improve the ranking performances, an active-learning based procedure is adopted, which 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.

Introduction

The standard classification or regression tasks do not include all the possible supervised learning problems. Some applications require to focus on other computed properties of the patterns, rather than on real values or classes. For instance, in ranking tasks, the goal is to produce a sorting of a set of objects, so that the result depends on a property 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 patterns 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 object. 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 on the 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 a sorting of a set of patterns. These two research areas have shown many interactions. In fact, preference learning techniques can be applied to solve ranking problems.

Preference Learning with the Comparative Neural Network

CmpNN processes a representation of two input objects x, y and produces an estimate of the "evidence" of the relationships $x \succ y$ and $x \prec y$. The input to the neural network is the concatenation of the two representations $[\mathbf{x}, \mathbf{y}] = [x_1, \cdots, x_d, y_1, \cdots, y_d]$. 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

$$t = \begin{cases} [1\ 0] & \text{if } x \succ y \\ [0\ 1] & \text{if } x \prec y \end{cases}, \tag{1}$$

and the error is measured by the squared error function

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

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

$$\begin{cases} x \succ y & \text{if } \ N_\succ([\mathbf{x}, \mathbf{y}]) = max(N_\succ([\mathbf{x}, \mathbf{y}]), N_\prec([\mathbf{x}, \mathbf{y}])) \\ x \prec y & \text{if } \ N_\prec([\mathbf{x}, \mathbf{y}]) = max(N_\succ([\mathbf{x}, \mathbf{y}]), N_\prec([\mathbf{x}, \mathbf{y}])) \end{cases} . \tag{3}$$

CmpNN is more expressive with respect to another model where one output is produced by the network and the other one is deduced, f.i., by the assumption $N_\succ([\mathbf{x}, \mathbf{y}]) = 1 - N_\prec([\mathbf{x}, \mathbf{y}])$.
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.

- **Reflexivity:** $x \succ x$ and $x \prec x$ hold.
- **Equivalence between $\prec$ and $\succ$:** if $x \succ y$ then $y \prec x$ and, vice versa, if $y \succ x$ then $x \prec y$.
- **Anti–symmetry:** if $x \succ y$ and $x \prec y$ then $x = y$.
- **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 limit [RPMB08, RPMS08].

The Comparative Neural Network Architecture (CmpNN)

The CmpNN architecture adopts a weight sharing technique in order to ensure that the reflexivity and the equivalence between $\prec$ and $\succ$ hold.

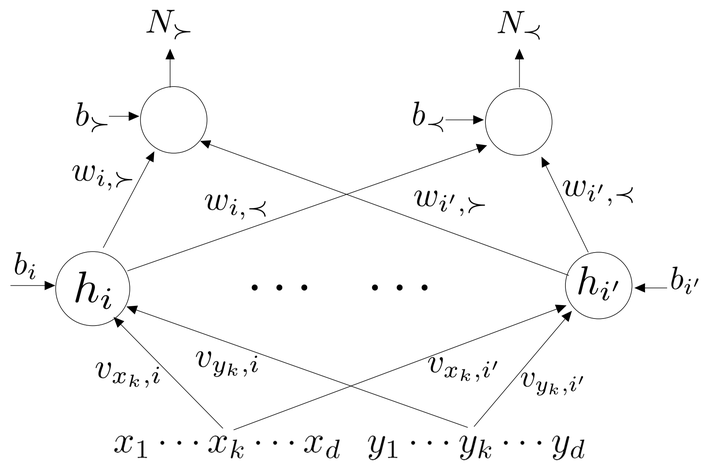

FIGURE 1: The comparative network architecture.

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 $N_\prec$.

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. It can, it will be proved that a CmpNN can approximate up to any degree of precision most of the practically useful functions that satisfy the constraint $N_\succ([\mathbf{x}, \mathbf{y}]) = N_\prec([\mathbf{y}, \mathbf{x}])$. Such a result is counterpart of the universal approximation capability of common three layered neural networks [HSW89, Fun89, Cyb89].

The learning algorithm

CmpNN is trained on a dataset composed of pairs of objects for which the value of the preference function is given. The comparative neural network is trained using the square error function 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.

1: $T \leftarrow Set\ of\ training\ objects$
2: $V \leftarrow Set\ of\ validation\ objects$
3: $P_T^0 \leftarrow A\ small\ random\ subset\ of\ T \times T$;
4: $P_V^0 \leftarrow A\ small\ random\ subset\ of\ V \times V$;
5: **for** $i = 0$ to max_iter **do**
6: $\quad C^i \leftarrow TrainAndValidate(P_T^i, P_V^i)$;
7: $\quad [E_T^i\ ,\ R_T^i] \leftarrow Sort(C^i, T)$;
8: $\quad [E_V^i\ ,\ R_V^i] \leftarrow Sort(C^i, V)$;
9: $\quad score \leftarrow RankQuality(R_V^i)$;
10: $\quad$ **if** $score > best\_score$ **then**
11: $\quad\quad best\_score \leftarrow score$;
12: $\quad\quad C^* \leftarrow C^i$;
13: $\quad$ **end if**
14: $\quad P_T^{i+1} \leftarrow P_T^i \cup E_T^i$;
15: $\quad P_V^{i+1} \leftarrow P_V^i \cup E_V^i$;
16: $\quad$ **if** $P_T^{i+1} = P_T^i$ and $P_V^{i+1} = P_V^i$ **then**
17: $\quad\quad$ return C^*;
18: $\quad$ **end if**
19: **end for**
20: return C^*;

ALGORITHM 1: The SortNet learning algorithm

Once the neural network is trained, SortNet can order a set of objects by $O(n \log n)$ operations, which is the lowest computational complexity reachable for a ranking algorithm.

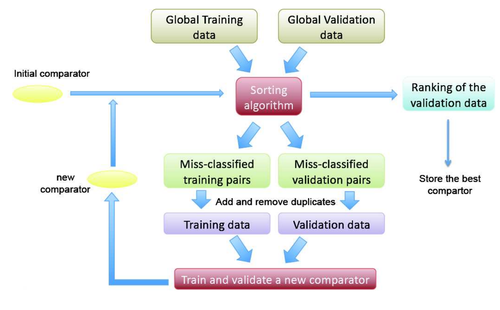

FIGURE 3: The SortNet algorithm scheme.

Experimental results

The SortNet algorithm has been experimentally evaluated on the LETOR dataset [LXQ+07], a package of benchmarks for LEarning TO Rank, released by Microsoft Research Asia and available on the Web. The task considered in LETOR is that of learning to rank, according to their relevance, the documents returned by information retrieval systems in response to a query. 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.

	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	**0,38**	**0,3**	**0,28**	**0,29**	**0,29**	**0,3**	**0,29**	**0,29**	**0,28**		**0,38**	**0,29**	**0,25**	**0,25**	**0,24**	**0,24**	**0,21**	**0,2**	**0,18**	**0,17**
SortNet P@10	**0,32**	**0,32**	**0,31**	**0,31**	**0,3**	**0,3**	**0,29**	**0,3**	**0,31**	**0,31**	**0,32**	**0,31**	**0,29**	**0,27**	**0,25**	**0,23**	**0,21**	**0,21**	**0,2**	**0,2**

TABLE 1: 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	0,48	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	0,41	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	**0,43**	**0,47**	**0,46**	**0,46**	**0,46**	**0,47**	**0,47**	**0,48**	**0,48**	**0,49**	**0,43**	**0,43**	**0,4**	**0,37**	**0,35**	**0,33**	**0,31**	**0,3**	**0,29**	**0,27**
SortNet P@10	**0,45**	**0,48**	**0,46**	**0,46**	**0,47**	**0,47**	**0,48**	**0,48**	**0,49**	**0,49**	**0,45**	**0,45**	**0,39**	**0,36**	**0,35**	**0,33**	**0,32**	**0,3**	**0,29**	**0,27**

TABLE 2: 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	0,5	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	0,47	0,45	0,44	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	**0,53**	**0,49**	**0,47**	**0,46**	**0,46**	**0,45**	**0,45**	**0,44**	**0,44**	**0,44**	**0,65**	**0,61**	**0,58**	**0,56**	**0,55**	**0,54**	**0,53**	**0,52**	**0,51**	**0,5**
SortNet P@10	**0,46**	**0,47**	**0,45**	**0,46**	**0,45**	**0,44**	**0,44**	**0,44**	**0,44**	**0,44**	**0,59**	**0,59**	**0,56**	**0,57**	**0,54**	**0,53**	**0,53**	**0,51**	**0,51**	**0,5**

TABLE 3: Results on OHSUMED measured by $NDCG@n$ and $P@n$.

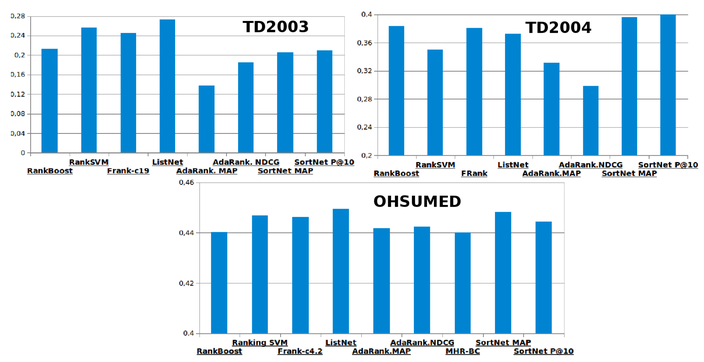

FIGURE 2: Result on TD2003, TD2004 and OHSUMED measured by MAP.

Conclusions

SortNet is a new adaptive ranking algorithm, it employs a feedforward neural network with a special architecture that allows to implement a preference function preserving some of its symmetries. The properties of such an architecture have been studied proving that it is an universal approximator. The SortNet algorithm has been evaluated on the LETOR benchmarks. The results show that the proposed method is often close the current state of the art and, in some it cases, it outperforms all the other approaches. Finally, some considerations about the errors of the CmpNN and about the cost function used to train the neural network have been draw. In particular, an error of the comparator in predicting the right order of a pair impacts on the final ranking scheme as a position error of one of the two objects in the pair. A further consideration regards the iterative learning scheme of the SortNet algorithm. The selective learning approach can be seen as an implementation of the hinge loss function which is largely used in the kernel-based models and which has been demostrated to have good results in classification and preference learning.
Matters of future research includes a wider experimentation and the application of the method to different kinds of preference learning problems. Moreover, it should be theoretically deepened the motivations underlining the proposed selective sampling procedure and its relationship to related algorithms as boosting methods and other active and selective learning approaches. Finally, we would investigate the performances of the CmpNN using different error functions.

References

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 3:303–314, 1989.

[Fun89] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2:183–192, 1989.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2:359–366, 1989.

[LXQ+07] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. LETOR: Benchmarking learning to rank for information retrieval. *In Proceedings of the 30th SIGIR Conference - Workshop on Learning to Rank for Information Retrieval*, 2007.

[RPMB08] L. Rigutini, T. Papini, M. Maggini, and M. Bianchini. A Neural Network Approach for Learning Object Ranking. *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN)*, pages 899–908, 2008.

[RPMS08] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli. SortNet: learning to rank by a neural-based sorting algorithm. *SIGIR 2008 workshop: Learning to Rank for Information Retrieval*, 2008.