A neural network approach for learning object ranking

Leonardo Rigutini[1], Tiziano Papini[2], Marco Maggini[3], and Monica Bianchini[4]

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Siena
Via Roma 56, Siena, Italy
{rigutini[1], papinit[2], maggini[3], monica[4]}@dii.unisi.it

Abstract. In this paper, we present a connectionist approach to preference learning. In particular, a neural network is trained to realize a comparison function, expressing the preference between two objects. Such a "comparator" can be subsequently integrated into a general ranking algorithm to provide a total ordering on some collection of objects. We evaluate the accuracy of the proposed approach using the LETOR benchmark, with promising preliminary results.

1 Introduction

Recently, the topic of *Preference Learning* received considerable attention in Artificial Intelligence (AI). For instance, in many tasks related to "Intelligent Agents" and "Planning", the best action to be taken at each step can not be unique, and could be appropriately chosen by exploiting an underlying "preference model". Similarly, the goal of "Object Ranking" can be cast in a framework where the criterion used to order a set of given objects is not predefined but inferred from users' preferences. In all these cases, the system behaviour should be adapted based on a set of feedbacks, provided by an user or by sensors. 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") to "collaborative filtering", where the preferences of a given user are estimated from the preferences of the 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 AI literature, preference learning problems have been formalized in various different settings, depending on the underlying preference model or the type of information provided to the learning system. However, two formalisms mainly appear to be central: the "Learning Labels Preference (LLP)" and the "Learning Objects Preference (LOP)".

This paper presents a neural model for Learning Objects Preferences based on a pairwise approach. In particular, a comparison function, referred to as a "comparator", is learned using a set of examples. Given an ordered input pair

of examples, the comparator decides which of the two examples is preferable. The performances of the proposed approach are firstly tested in a classification framework by evaluating the accuracy of the preference assignment for each input pair, and then the comparator is applied to a ranking task by using three different schemes. These tests are performed using the LETOR (LEarning TO Rank) dataset [1], which is a standard benchmark for the "Learning to Rank" task. The comparison considers several state-of-the-art ranking algorithms for this task, like RankSVM [2], RankBoost [3], FRank [4], ListNet [5], and AdaRank [6].

The paper is organized as follows. In the next section, we introduce the model and provide a brief mathematical description, whereas, in Section 3 we describe the three ranking algorithms used to test the comparator. Subsequently, in Section 4, we present the experimental setup, the LETOR dataset, the related evaluation measures, and some comparative experimental results. Finally, in Section 5, same conclusions are drawn.

2 The comparator model

The connectionist method proposed in this paper is based on a pairwise preference learning approach. In particular, let S be a set of objects described by a vector of features, and let $< x, y > \in S \times S$ be the pair of objects to be compared. The model is trained to decide if x has to be preferred to y or vice-versa. Formally, a pairwise preference function realizes a function f such that:

$$f(x,y) = \begin{cases} > 0 \; if \; x \succ y \\ < 0 \; if \; x \prec y \\ = 0 \; if \; x \sim y \end{cases} \tag{1}$$

where $x \succ y$ means that x is preferred to y, $x \prec y$ if y is preferred to x, and $x \sim y$ if there is no preference between x and y (i.e. x and y are considered equivalent). In many cases, the equivalence relationship can be ignored. To induce a correct ordering, the function f has to meet the following constraints:

- anti-simmetry: if $x \succ y$ then $y \prec x$. It means that: $f(x,y) = -f(y,x)$.
- self-equivalence: if $x = y$, then $f(x,x) = 0$.
- transitivity: if $x \succ y$ and $y \succ q$, then $x \succ q$. It means that if $f(x,y) > 0$ and $f(y,q) > 0$, then $f(x,q) > 0$.

The transitivity property is not guaranteed by the proposed model, even if it can be learnt from data.

2.1 The comparator neural network

The comparator neural network is a feed-forward neural network with a single output neuron such that:

- all neurons (hiddens and output) have an anti-symmetric transfer function;

– all neurons have no biases;
– the input pattern is the difference between the two exampels in the pair: $z = (x - y)$.

This particular architecture realizes an anti-symmetric function $f(z) : f(\overrightarrow{z}) = -f(\overleftarrow{z})$, where $\overrightarrow{z}$ indicates the ordered pair $< x, y >$, while $\overleftarrow{z}$ indicates $< y, x >$. This can be easily shown as follows. The output, for the inputs $\overrightarrow{z}$ and $\overleftarrow{z}$ can be expressed as

$$f(\overrightarrow{z}) = s(\sum_i w_i \overrightarrow{y_i} + b)$$
$$f(\overleftarrow{z}) = s(\sum_i w_i \overleftarrow{y_i} + b)$$

where s is the transfer function of the output neuron, and y_i and w_i are the hidden neurons' outputs and the hidden-output weights, respectively . By imposing the anti-symmetry of f, we can write:

$$s\left(\sum_i w_i \overrightarrow{y_i} + b\right) = -s\left(\sum_i w_i \overleftarrow{y_i} + b\right) \ .$$

If s is an anti-symmetric function, it follows that

$$\sum_i w_i \overrightarrow{y_i} + b = -\sum_i w_i \overleftarrow{y_i} - b \Longrightarrow \begin{cases} \overrightarrow{y_i} = -\overleftarrow{y_i} \\ b = -b = 0 \end{cases}$$

where $\overrightarrow{y_i}$ and $\overleftarrow{y_i}$ represent the output of the i^{th} hidden neuron for the inputs $\overrightarrow{z}$ and $\overleftarrow{z}$, respectively. Similarly, $\overrightarrow{y_i}$ and $\overleftarrow{y_i}$ can be expressed as:

$$\overrightarrow{y_i} = g(\sum_j v_{x_j,i} x_j + \sum_j v_{y_j,i} y_j + b_i)$$
$$\overleftarrow{y_i} = g(\sum_j v_{x_j,i} y_j + \sum_j v_{y_j,i} x_j + b_i)$$

where $v_{x_j,i}$ and $v_{y_j,i}$ are the weights between the j^{th} feature of x and y, and the hidden neuron i. By imposing $\overrightarrow{y_i} = -\overleftarrow{y_i}$, we can write:

$$g(\sum_j v_{x_j,i} x_j + \sum_j v_{y_j,i} y_j + b_i) = -g(\sum_j v_{x_j,i} y_j + \sum_j v_{y_j,i} x_j + b_i) \ .$$

If the function g is anti-symmetric, it follows that

$$\sum_j v_{x_j,i} x_j + \sum_j v_{y_j,i} y_j + b_i = -\sum_j v_{x_j,i} y_j - \sum_j v_{y_j,i} x_j - b_i$$
$$\Longrightarrow \sum_j v_{x_j,i}(x_j + y_j) + b_i = \sum_j -v_{y_j,i}(x_j + y_j) - b_i \Longrightarrow \begin{cases} v_{x_j,i} = -v_{y_j,i} \\ b_i = -b_i = 0 \end{cases} \ .$$

Therefore, the weight between the i^{th} hidden neuron and the j^{th} feature of x has to be the opposite of that between the i^{th} neuron and the j^{th} feature of y, which simply corresponds to consider $z_j = x_j - y_j$ as the j^{th} input to the network. Thus, the function implemented by each hidden neuron i is

$$y_i = g\left(\sum_j v_{x_j,i}\,(x_j - y_j)\right) \ .$$

This neural architecture also satisfies the self-equivalence property, since the network output is zero when the input pair is $< x, x >$. In fact, since g and s are anti-symmetric functions,

$$y_i = g\left(\sum_j v_{x_j,i}\,(x_j - x_j)\right) = 0$$
$$f(z) = s(\sum_i w_i y_i) = 0 \ .$$

The hyperbolic tangent can be used as transfer function both for the hidden and output neurons. When training the model, for each pair of inputs $< e_1, e_2 >$, the target is assigned as

$$\begin{cases} t = 1 \quad if\ e_1 \succ e_2 \\ t = 0 \quad if\ e_1 \sim e_2 \\ t = -1\ if\ e_1 \prec e_2 \end{cases} \ .$$

However, in the experimental phase, we noticed that the model obtains better performances when trained without using the equivalence relationship. In fact, the network is still able to learn the "no-preference" relationship hidden in the training set even without using explicit examples.

3 The ranking algorithm

A trained neural network comparator can be exploited to rank objects by means of a sorting algorithm, an one-vs-all scoring scheme, or a page-rank like approach.

Ranking by a sorting algorithm

In this approach, the trained preference function is used as the comparison function in a sorting algorithm. The set of objects can be ordered with $O(n \log n)$ time complexity, according to the ordering defined by the preference function. It must be noticed that the proposed model does not realize a perfect comparison function since the transitivity property is not guaranteed. This could result in the definition of a partial ordering for which different rankings are admissible. However, the experimental results show that, when the transitivity is implicit in

the training set, it is learnt quite well by the proposed model. In these experiments, the same sorting algorithm was applied to different shufflings of the same objects, and different sorting algorithms were used on the same set of objects. The obtained final orderings were the same in most of the cases and seldom they differed only by a very small number of object positions ($1-5$ over 1000).

Ranking by one-vs-all scoring

The objects are ranked by exploiting a sort of "championship" consisting in all the "matches" between pairs of competitors. The i^{th} object is compared with the $n-1$ remaining objects using the trained preference model. If $o_i \succ o_j$, o_i wins the match and its total score is incremented by 1, otherwise the score of o_i is unchanged. When $o_i \sim o_j$, there is a draw and the corresponding scores are not modified. This algorithm has an $O(n^2)$ time complexity, since it requires $n(n-1)/2$ matches.

Ranking by page-rank scoring

This ranking algorithm exploits the PageRank algorithm [7]. In particular, the neural network comparator is used to build the preference graph on the objects: the sign of the network output is used to determine the direction of the arc between the nodes corresponding to the compared objects. The graph adjacency matrix is normalized, such that the elements on each row sum to 1, and the PageRank vector $x \in \mathbf{R}^N$ is computed by the following iterative computation

$$\begin{cases} x^{t+1} = d \cdot W \cdot x^t + \frac{(1-d)}{N} \cdot \mathbf{1} \\ x^0 = \frac{1}{N} \cdot \mathbf{1} \end{cases}$$

where $W \in \mathbf{R}^{N \times N}$ is the transpose of the normalized adiacency matrix, N is the number of nodes in the graph (the number of objects to rank), and $d \in [0,1]$ is the dumping factor, typically set to 0.85. The resulting score vector x is then used to order the objects.

4 Experimental results

The performances of the proposed schemes have been evaluated on the LETOR dataset [1], a package of benchmark datasets[1] for LEarning TO Rank, released by Microsoft Research Asia. This dataset collects objects which correspond to query-document pairs on the OHSUMED and TREC document collections. The documents are represented using several classical information retrieval (IR) features, such as term frequency, inverse document frequency, BM25, language models, and other features proposed in the recent literature, such as HostRank, Feature propagation and Topical PageRank. For the experiments reported in this

[1] http://research.microsoft.com/users/LETOR/

paper, we considered only the TREC collections, since the OHSUMED dataset considers three relevance degrees. LETOR contains two TREC datasets, TD2003 and TD2004. TD2003 is composed by 50 sets of documents from TREC 2003, each one containing the 1000 most relevant documents returned to one out of 50 different queries. Similarly, TD2004 contains documents from the TREC 2004 returned to 75 different queries. Each query-document pair is represented by 44 features described in the LETOR technical report. A label is provided for each document to define its actual relevance with respect to the corresponding query, defining the set R of relevant documents and the set NR of the not-relevant ones. For all queries, the number of relevant documents is roughly the 1% with respect to the whole set of documents. Each dataset is partitioned into five subsets in order to apply a 5-fold cross-validation. Thus, for TD2003 there are 10 queries for each subset, whereas for TD2004 each subset collects the results of 15 queries. For each fold, three subsets are used for training, one subset for validation, and one for testing.

As suggested in the LETOR technical report, the feature vectors have been normalized to $[0, 1]$. In particular, if N^i documents d^i_j, $j = 1, ..., N^i$, are returned as result to the i-th query, and $x^i_{j,r}$, with $j = 1, ..., N^i$, indicates the r-th feature of the document d^i_j, the corresponding normalized feature $\hat{x}^i_{j,r}$ is

$$\hat{x}^i_{j,r} = \frac{x^i_{j,r} - average_{s=1,...,N^i}(x^i_{s,r})}{max_{s=1,...,N^i}(x^i_{s,r})} \tag{2}$$

The learning set is built by selecting a pair $< x, y >$ of documents, each from one of the two relevance classes. The corresponding target value for the network output is 1 if $x \in R$ and $y \in NR$, -1 otherwise. The number of hidden neurons was chosen by a trial-and-error procedure using the validation set.

To evaluate the accuracy of the comparison function learnt by the neural network, the network output is compared with respect to the actual preference order for each pair of documents in the test set, as defined by the relevance classes to which the two documents belong. The evaluation considers only pairs of documents belonging to different classes. In fact, the available labeling allows only a clear preference ordering when the two documents belong to different classes, and it is too restrictive to assume a strict equivalence relationship when they belong to the same class. Documents inside the same class can have any ordering, since no information is provided to prefer one ordering to the other. Finally, it should be noticed that if the network provides an erroneous value for the comparison of the pair $< x, y >$, then, due to its anti-symmetry, also the comparison of $< y, x >$ is wrong.

For this task, the best performances were obtained using a network with 20 hidden neurons. The total accuracy, averaged on the 5 folds, is quite different on the two datasets. On TD2003, the neural network was able to compare correctly $83,53\% \pm 0.1$ of the test pairs, while on TD2004 the performances were significantly better, yielding an accuracy of $96,51\% \pm 0.05$. This is probably due to the different characteristics of the two document sets and the related relevance criterion. However, the results show that, in the best case, the neural network

comparator was able to approximate the pairwise preference ordering with an error at most of 3.5% on the considered pairs.

Since the final goal is to rank the results sets, the neural network comparator was exploited to order the documents in each set using the three algorithms described in Section 3. In this setting, a good ordering is characterized by the presence of the documents belonging to the relevant class in the top positions. The obtained results are compared to those reported in [1]: RankSVM [2], RankBoost [3], FRank [4], ListNet [5], and AdaRank [6].

The performances of these algorithms are compared using the ranking measures proposed in the LETOR technical report, defined as follows.

- **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.

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

- **Mean average precision (MAP)** — Given a query q, the average precision is evaluated as

$$AP_q = \frac{\sum_{n=1}^{N} P@n \cdot rel(n)}{\text{total relevant docs for } q}$$

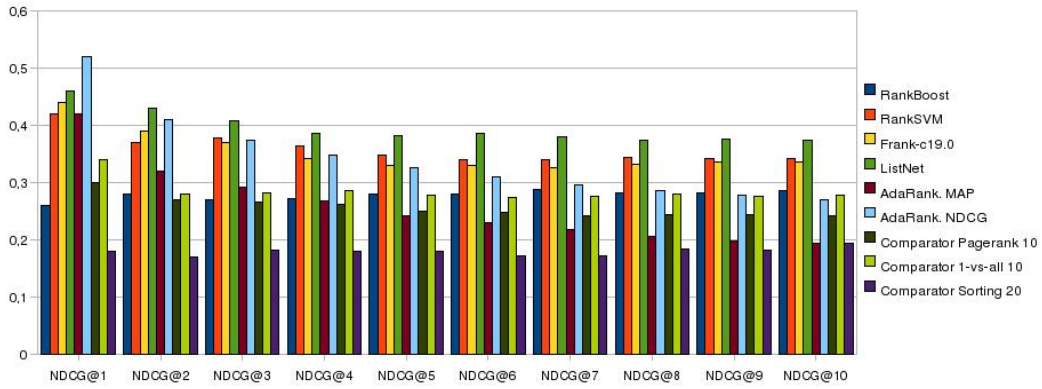
  where N is the number of documents in the result set of q and $rel(n)$ is 1 if the n^{th} document in the ordering is relevant, 0 otherwise. Thus, AP_q averages the values of the $P@n$ when n equals the positions of each relevant document. Finally, the MAP value is computed as the mean of AP_q over a set of queries.
- **Normalized discount cumulative gain (NDCG)** — This measure is able to handle multiple levels of relevance. The NDCG value of a ranking list at position n is calculated as

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

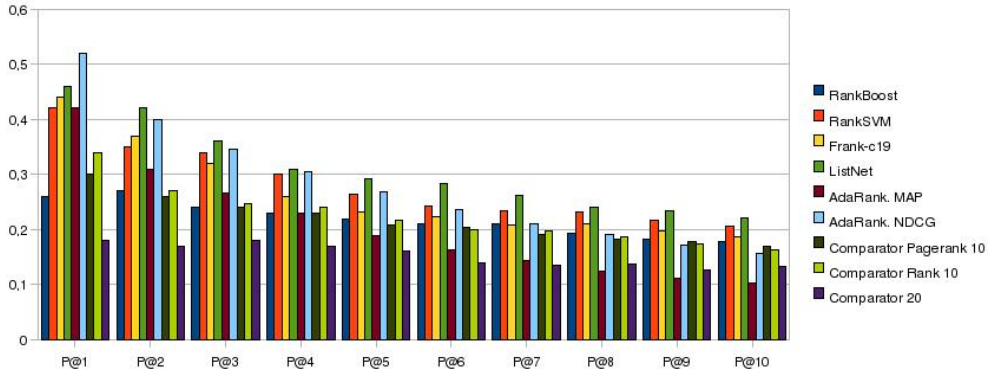
  where $r(j)$ is the rating of the j^{th} document in the list (0 is the worst), and Z_n is chosen such that the ideal ordering (the DGC-maximizing one) gets a NDCG score of 1.

For each ranking algorithm, we report the results for the best neural network architecture. In particular, for the experiments using the TD2003 dataset, the best performances were obtained with 10 hidden neurons for the one-vs-all and the PageRank algorithms, whereas 20 hidden neurons were used for the neural network exploited in the sorting algorithm. The results on TD2003 dataset are shown in Figure 1. All the three algorithms show values of $P@n$ and $NDCG@n$ that are comparable to the RankBoost, AdaRank-MAP, and AdaRank-NDCG algorithms. In particular, the results improve when increasing the parameter n.

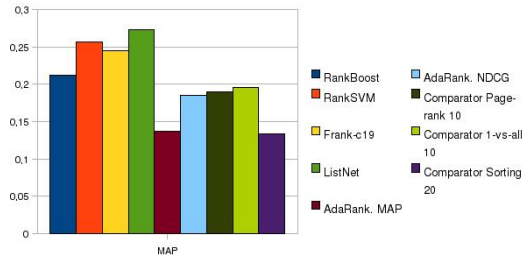
In the experiments on the TD2004 dataset, 20 hidden neurons were used for the neural network exploited in the one-vs-all and PageRank algorithms,



(a) NDCG@n



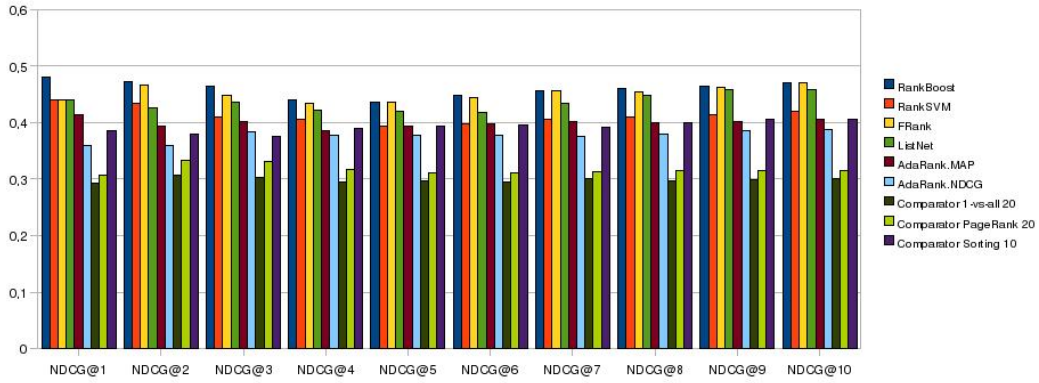
(b) P@n



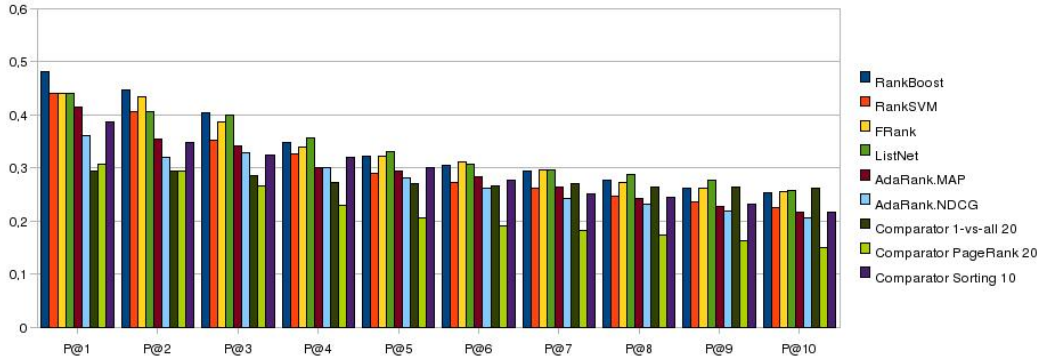
(c) MAP

Fig. 1. Results on the test set of TREC2003.

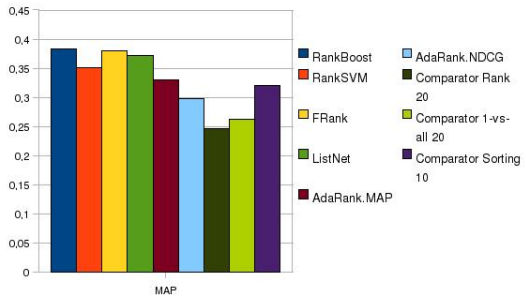
whereas the neural network used as comparison function in the sorting algorithm had 10 hidden neurons. The results are shown in Figure 2. We can notice that the NDCG of the sorting algorithm is better than the values of the AdaRank-



(a) NDCG@n



(b) P@n



(c) MAP

Fig. 2. Results on the test set of TREC2004.

MAP and AdaRank-NDCG algorithms. Moreover, for increasing values of n, the algorithm reaches the performances of RankSVM.

5 Conclusions

In this paper a connectionist approach to the preference learning task has been proposed. A neural network is trained using patterns formed by pairs of examples. Given an ordered input pair, the comparator decides which of the two objects is preferable. The model is then exploited in a ranking algorithm to obtain a total ordering over a set of objects. Experiments were carried out to evaluate the performance of the proposed algorithm using the datasets TD2003 and TD2004, available among the LETOR benchmarks. RankingSVM, RankBoost, FRank, ListNet and AdaRank were compared with the proposed approach and the results show that the neural comparator combined with different ranking algorithm obtains comparable performances with respect to all the methods, except AdaRank, that is outperformed.

References

1. Liu, T.Y., Xu, J., Qin, T., Xiong, W., Li, H.: LETOR: Benchmarking learning to rank for information retrieval. In: SIGIR 2007 – Workshop on Learning to Rank for Information Retrieval, Amsterdam, The Netherlands (2007)
2. Cao, Y., Xu, J., Liu, T.Y., Li, H., Huang, Y., Hon, H.W.: Adapting ranking SVM to document retrieval. In: Proceedings of ACM SIGIR 2006, New York, NY, USA, ACM (2006) 186–193
3. Freund, Y., Iyer, R., Schapire, R.E., Singer, Y.: An efficient boosting algorithm for combining preferences. In Shavlik, J.W., ed.: Proceedings of ICML'98, Morgan Kaufmann Publishers, San Francisco, USA (1998) 170–178
4. Tsai, M.F., Liu, T.Y., Qin, T., Chen, H.H., Ma, W.Y.: FRank: a ranking method with fidelity loss. In: Proceedings of the ACM SIGIR 2007, New York, NY, USA, ACM (2007) 383–390
5. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: Proceedings of ICML 2007, New York, NY, USA, ACM (2007) 129–136
6. Xu, J., Li, H.: AdaRank: a boosting algorithm for information retrieval. In: Proceedings of ACM SIGIR 2007, New York, NY, USA, ACM (2007) 391–398
7. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web. (1998)