RankMerging: Learning to rank in large-scale social networks

Lionel Tabourier[1,2], Anne-Sophie Libert[1], and Renaud Lambiotte[1]

[1] naXys, University of Namur, 8 Rempart de la Vierge, 5000 Namur, Belgium
[2] LIP6, University Pierre and Marie Curie, 4 Place Jussieu, 75252 Paris, France

Abstract In this work, we consider the issue of unveiling unknown links in a social network, one of the difficulties of this problem being the small number of unobserved links in comparison of the total number of pairs of nodes. We define a simple supervised learning-to-rank framework, called *RankMerging*, which aims at combining information provided by various unsupervised rankings. As an illustration, we apply the method to the case of a cell phone service provider, which uses the network among its contractors as a learning set to discover links existing among users of its competitors. We show that our method substantially improves the performance of unsupervised metrics of classification. Finally, we discuss how it can be used with additional sources of data, including temporal or semantic information.

1 Introduction

Link prediction is a key field of research for the mining and analysis of large-scale social networks. The reasons lie in its many practical applications: going from recommendation strategies for commercial websites [20] to recovering missing links in incomplete data [31]. Link prediction also has significant implications from a fundamental point of view, as it allows for the identification of the elementary mechanisms behind the creation and decay of links in time-evolving networks [17]. For example, triadic closure, at the core of standard methods of link prediction, is considered as one of the driving forces for the creation of links in social networks [16].

In their seminal formulation, Liben-Nowell and Kleinberg [18] present link prediction as follows: considering a snapshot of a network at time t, the problem is to predict which links will be present at a future time $t' > t$. A standard way to solve this binary classification problem consists in ranking pairs of nodes according to a scalar metric, correlated with the existence of interactions between nodes. Most of these metrics are based on the structural properties of the network of known interactions, either on local, e.g. the number of common neighbors, or on global features, e.g. random walk or hitting time — see [22] for a survey. Other sources of information can be considered, in particular node attributes, such as age or gender [23,2] or geographic location [28]. In certain cases, for example online networks, the users profiles may include rich semantic information as [4].

Interaction attributes, such as frequencies [29], or the time elapsed since the last interaction [27], may also be used as an additional source of information.

However, these ranking features are known to be domain-specific (e.g., [8]). Moreover, as links can play different roles in social networks, they are expected to be surrounded by different types of environments and thus to be best identified by different topological features. For these reasons, schemes based on a single metric are prone to misclassification. A way to circumvent this problem is to combine different metrics for link prediction. Most of the available solutions are unsupervised, such as Borda's method or Markov chain ordering [11]. On the other hand, supervised methods have proven efficient on specific link prediction problems. For instance, the authors of [19] considered degree categories to predict future links in a phonecall network; in both [14] and [8], supervised frameworks are defined to predict links in multimodal networks. Recent works have combined classic tools, such as classification trees, support vector machine, or neural networks, which have been shown to outperform their unsupervised counterparts for predicting links in biological networks and scientific collaboration networks [3,26,8]. However, these methods do not allow the user to set the number of predictions according to his needs, whereas ranking methods are suited to this purpose. Supervised methods built to improve ranking systems have mostly been designed in the context of information retrieval tasks, such as document filtering, spam webpage detection, recommendation or text summarization [21,30]. As such, they primarily aim at high precision on the top-ranked items, and stress the relative ranking of two items [12,5,6].

Here, we propose a simple yet efficient *learning-to-rank* supervised framework specifically designed to uncover links in social networks, by combining rankings obtained from different sources of information. Throughout this article, our Ariadne's thread is the examle of a mobile phone service provider (PSP). Because PSPs only have access to records involving their own clients, they have an incomplete view on the social network as a whole. In several practical applications, however, this missing information is crucial. An important example is churn prediction, that is the detection of clients at risk of leaving to another PSP. This risk is known to depend on the local structure of the social network [7,25]. Therefore, the knowledge of connections between subscribers of other providers is crucial for the design of efficient customer retention campaigns. As we discuss further below, a direct application of our method is the prediction of links on the other side of the frontier accessible to a PSP.

This article is organized as follows. In Section 2, we describe the mobile phone dataset and how it is processed in order to model the situation of a PSP confronted to churn. In Section 3, we briefly present how classic unsupervised learning methods can be applied to the problem under consideration. In Section 4, we develop a supervised machine learning framework, called *RankMerging*, which improves the quality of predictions by aggregating the information from various metrics. We also compare our results to those of standard supervised methods, and show that *RankMerging* is particularly suited to social networks where information is partial and noisy.

2 Dataset

The dataset is a call detail record of approximately $14 \cdot 10^6$ phonecalls of anonymized subscribers of a European PSP during a one month period. Users are described as nodes and an interaction between two users is a directed link. The total number of phone calls between nodes i and j is denoted by the *weight* $w(i,j)$ of this link. In order to filter out calls which are not indicative of a lasting social relationship, we only consider calls on bidirectional links, i.e. links that have been activated in both directions. After this filtering has been applied, interactions between users are considered as undirected. The resulting social network is composed of 1,131,049 nodes, 795,865 links and 10,934,277 calls. From now on, we denote W the activity of a node, that is the sum of the weights of its adjacent links.

A PSP is usually confronted to the following situation: it has full access to the details of phone calls between its subscribers, as well as between one of its subscriber and a subscriber of another PSP. However, connections between subscribers of other PSPs are hidden. In order to simulate this situation from our dataset, we divide our data into two artificial PSPs, A and B. The link to predict are selected in a random way: we split the users of the network into 3 sets (i) A_1 and A_2 nodes are nodes of the first PSP – links among A_2 nodes are links to discover during the learning task, (ii) B nodes belong to the second PSP. Users have been randomly assigned to A_1, A_2 and B according to the proportions 50, 25, 25%. During the learning process, links inside $A_2 \cup B$ are removed. The resulting network ($\mathcal{G}_{Learn}$) contains 848,911 nodes and 597,538 links and we aim at predicting the links among A_2 nodes (49,731 links). During the test phase, the network $\mathcal{G}_{Test}$ contains 1,131,049 nodes and 746,202 links. All the links are thus considered, except for the 49,663 links inside B which we aim at predicting. Notice that according to our simulation, the learning set and the test set are derived from the same distributions, while it could not be the case in situations involving several PSPs. However, this assumption is fair to compare the performances of our method to other prediction techniques.

3 Unsupervised learning

3.1 Prediction evaluation

The quality of a ranking metric is assessed by measuring precision (**Pr**), recall (**Rc**) and **F**-score. Previous works have emphasized the dramatic effect of class imbalance (or skewness) on link prediction problems in social networks, especially in mobile phone networks [19,6]. The fact that the network is sparse and that there are many more pairs of nodes than links makes the prediction and its evaluation tricky, the typical order of magnitude of the classes ratio for a social network made of N nodes being $O(1/N)$ [19]. In the case of unbalanced classes, performance predictors such as the ROC curve are known to be inappropriate – see [10]. In general, the definition of a *good* prediction and thus of an adequate quality estimator depends on the purpose of the link prediction. For example,

the goal of an efficient search engine is to provide highly relevant information on a small amount of items. In contrast, a PSP confronted to churn looks for an appropriate trade-off between precision and recall, in order to detect potential churners, without flooding clients with discount offers. For this reason, we aim at improving both precision and recall over a large range so that a PSP would be able to tune the prediction parameters according to its commercial strategy.

3.2 Ranking metrics and results

In this work, we focus on structural features where each pair of nodes is assigned a score based on topological information, and then ranked according to this score. A large number of metrics have been used in the past, see for example [18,22]. The goal of this paper is not to propose elaborate classifiers, but to present a method that takes advantage of how complementary they are. We have therefore chosen classic metrics and generalized them to the case of weighted networks (other generalizations exist in the literature, e.g. [24]), denoted by index w. Their unweighted version is recovered by setting all weights to 1.

Local features. A class of metrics are local (also called *neighborhood rankers*) as they only rank links among nodes which are at most at distance 2. In the following, $\mathcal{N}(i)$ denotes the set of neighbors of node i.

- *Common Neighbors index (CN)*, based on the weighted number of common neighbors shared by nodes i and j:

$$s_{CN_w}(i,j) = \sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} w(i,k).w(j,k)$$

- *Jaccard index (Jacc)*:

$$s_{Jacc_w}(i,j) = \frac{\sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} w(i,k) + w(j,k)}{W(i) + W(j)}$$

- *Adamic-Adar index (AA)*:

$$s_{AA_w}(i,j) = \sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} \frac{1}{log(W(k))}$$

Global features. Another class of features are global (or *path rankers*), since they are calculated by using the whole structure of the network, and allow for the ranking of distant pairs of nodes:

- *Katz index (Katz)* [15], computed from the number of paths from node i to node j of length l, i.e. $\nu_{ij}(l)$ according to the following expression (β is a bounded parameter):

$$s_{Katz}(i,j) = \sum_{l=1}^{\infty} \beta^l \nu_{ij}(l)$$

  Note that in the weighted case, the number of paths is computed as if links were multilinks.

– *Random Walk with Restart index (RWR)*, $s_{RWR_w}(i,j)$ is defined as the probability that a random walker starting on node i, going from a node k to a node k' with probability $p.w(k,k')/W(k)$ and returning on i with probability $1-p$, is on j in the steady state of the process.
– *Preferential Attachment index (PA)*, based on the observation that active nodes tend to connect preferentially in social networks [13]:

$$s_{PA_w}(i,j) = W(i).W(j)$$

Both *Katz* and *RWR* are computed using infinite sums, which will be approximated by keeping only the dominating four first terms to reduce the computational cost, meaning that we can only predict links between pairs of nodes at a maximum distance of 4. Moreover, in order to address the problem of class imbalance, known to hinder the performance of PA (e.g. [19]), we have restricted the ranking in this case to pairs of nodes at a maximum distance of 3. Notice that with larger maximum distances, we can increase the maximum recall that can be reached, but at the cost of a very low precision for these predictions.

3.3 Borda's method

The main purpose of this work is to develop a framework to exploit a set of α rankings for link prediction. As we will show in the next section, this problem can be solved in a supervised setting by identifying regions of a α-dimensional space associated to a high performance. Here, we present an unsupervised way to merge ranking features based on social choice theory [11,26]. Borda's method is a *rank-then-combine* method originally proposed to obtain a consensus from a voting system [9]. Each pair is given a score corresponding to the sum of the number of pairs ranked below, that is to say:

$$s_B(i,j) = \sum_{\kappa=1}^{\alpha} |r_\kappa| - r_\kappa(i,j)$$

where $|r_\kappa|$ denotes the number of elements ranked in r_κ. This scoring system may be biased toward global predictors by the fact that local rankings have less elements. To alleviate this problem, unranked pairs in ranking r_κ but ranked in $r_{\kappa'}$ will be considered as ranked in r_κ on an equal footing as any other unranked pair and below all ranked pairs. Borda's method is computationally very cheap, which is a highly desirable property in our case. A comprehensive discussion on this method can be found in [11].

3.4 Results

We plot the results obtained on $\mathcal{G}_{Learn}$ to predict $A_2 - A_2$ links for the above classifiers. For the sake of readability, we only represent a selection of them on Figure 1. The evolution of the **F**-score significantly varies from one classifier to another. For example, it increases sharply for *CN*, and then more slowly until reaching its maximum, while RWR_w rises smoothly before slowly declining. As expected, Borda's aggregation improves the performance of the classification,

especially considering the precision on the top-ranked pairs. Given the difficulty of the task, **Pr** is low on average: for instance, when **Rc** is greater than 0.06, **Pr** is lower than 0.3 for all estimators.

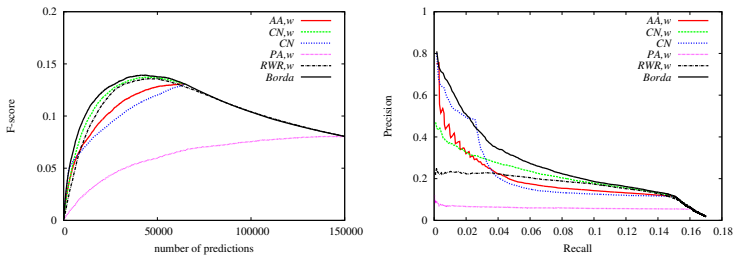

Figure 1. Results on the learning set for various structural classifiers. Left: **F**-score as a function of the number of predictions. Right: **Pr** versus **Rc** curves.

4 *RankMerging* framework

The ranking methods presented in the former section use structural information in complementary ways. In systems such as social networks, where communication patterns of different groups, e.g. family, friends, coworkers etc, are different, one expects that a link detected as likely by using a specific ranking method may not be discovered using another one. This intuition is supported by measuring the correlation between rankings. For example, the rankings produced with s_{CN} and $s_{Jacc,w}$ scores have a 0.052 Spearman correlation coefficient on the test set, which means that some pairs are highly ranked according to one method, but not to the other one. In this section, we design a supervised machine learning framework to aggregate information from various ranking techniques for link prediction in social networks. In a nutshell, it does not demand for a pair to be highly ranked according to all criteria (like with a consensus rule), but at least one. The whole procedure is referred to as *RankMerging*. An implementation and user guide of the algorithm are available on `http://lioneltabourier.fr/program.html`.

4.1 Principle

We first consider the training set to learn the parameters. Let's suppose that we have α different rankings $r_1, ..., r_\alpha$ from the unsupervised learning phase. Each ranked pair is either a true or false positive prediction (tp or fp). We are looking for the combination of rankings that brings the maximum number of tp predictions. Let $\Sigma_i(\rho_i)$ be the number of tp for pairs ranked from position 1 to ρ_i, ρ_i will be named *sliding index* of r_i. Now if we consider simultaneously the α rankings, our goal is to compute the optimal values of the ρ_i for a fixed

number of predictions n so that the total number of *tp*, denoted $\mathcal{S}(\rho_1, ..., \rho_\alpha)$, is maximum. It is important to note that a link can only be predicted once: if a pair has been predicted using ranking r_i and then included in the solution associated to $\mathcal{S}$, it cannot be predicted by r_j. It means that we have to consider the links already predicted to guess the future ones, which makes this problem hard to solve exactly. For this reason, the problem is solved heuristically, by using the training algorithm described in Algorithm 1.

The central idea is to find at each step the ranking with the highest number of true predictions in the next g coming steps. For that purpose, we define the *window* $\mathcal{W}_i$ as the set of links predicted according to ranking r_i in the next g steps. Note that links already predicted are not considered in $\mathcal{W}_i$. The number of *tp* in $\mathcal{W}_i$ is its *quality*, denoted $\chi(i)$. The ranking corresponding to the highest quality value is selected (in the case of a tie, we choose randomly), and we add its next top-ranked pair to the output ranking of the learning phase r_M^L. Throughout the process, the sliding indices ρ_i are registered, these values are the essential outputs of the training phase, as they record which ranking contributed to the merged ranking. Then, the indices σ_i which indicate the end of the windows are updated so that $\mathcal{W}_i$ contain exactly g pairs, $\chi(i)$ are updated too, and the process is iterated until r_M^L contains a predefined number of pairs T. To summarize, we are looking for a maximum number of true predictions $\mathcal{S}$ by local search. An important benefit of our learning algorithm is that it needs to go through each ranking only once, so if we have α rankings, it implies a $O(\alpha N)$ temporal complexity. A brute force algorithm demands to consider all possible rankings combinations, meaning that α different possibilities for each prediction must be considered, that is $O(\alpha^N)$ complexity.

Table 1 gives an example of the two first steps of the merging process between two rankings r_A and r_B with $g = 5$. Pairs in the windows are represented with a gray background. Initially, there are 4 *tp* in $\mathcal{W}_A$ and 3 *tp* in $\mathcal{W}_B$, it means that $\chi_A > \chi_B$, so the first link selected is the top-ranked pair available in $\mathcal{W}_A$: $(1, 2)$ (green background). This pair is therefore excluded from the ranking r_B (barred item on red background). At the next step, we have $\chi_A = \chi_B = 4$, the ranking with highest quality is then selected randomly, we suppose here that r_B has been selected so that the next link included in r_M^L is $(5, 18)$. At this step, according to the previously defined notations, $\rho_1 = 1$ and $\Sigma_1(\rho_1) = 0$, while $\rho_2 = 1$ and $\Sigma_2(\rho_2) = 1$.

The test phase of the procedure consists in combining rankings on the test network $\mathcal{G}_{Test}$ according to the ρ_i learned on the training network $\mathcal{G}_{Learn}$. This process does not demand to define sliding windows. The practical implementation is simple: at each step, we look up for the ranking chosen according to the learning process and select the corresponding ranking r_i on the test set. Its highest ranked pair is then added to the merged ranking of the test phase r_M^T if it has not been included previously; if it is already in r_M^T, then we go to the next highest ranked pair of r_i until we have found one which is not in r_M^T. The number of pairs that can be predicted is different in $\mathcal{G}_{Learn}$ and $\mathcal{G}_{Test}$ (resp. n_L and n_T). The implementation should then take into account a scaling factor:

Algorithm 1: *RankMerging* method: training algorithm.

inputs : table of rankings $\mathcal{R}$ ($r_i = \mathcal{R}[i]$); real edge list E;
maximum number of predictions T; g;
outputs: sliding index table ρ; merged ranking r_M^L;
`// initialization:`
begin
  $\mathcal{W}[i] \leftarrow g$ first links in $\mathcal{R}[i]$; `// pairs in window` i
  $\chi[i] \leftarrow |\mathcal{W}[i] \cap E|$; `// quality of window` i
  $\rho[i] \leftarrow 0$; `// sliding index of` r_i `= start index of window` i
  $\sigma[i] \leftarrow g$; `// end index of window` i
  $n \leftarrow 0$; `// counter of the number of predictions`

while $\underline{\text{n} \leq \text{T}}$ **do**
  $i_{max} \leftarrow$ index corresponding to maximum $\chi[i]$;
  $r_M^L[n] \leftarrow \mathcal{R}[i_{max}][\rho[i_{max}]]$;
  n $\leftarrow$ n+1;
  $\rho[i_{max}] \leftarrow \rho[i_{max}] + 1$;
  $\forall i$ Update$(\mathcal{W}[i], \chi[i], \rho[i], \sigma[i], r_M^L[n])$;

Procedure: Update$(\mathcal{W}[i], \chi[i], \rho[i], \sigma[i], r_M^L[n])$:
begin
  if $\underline{r_M^L[n] \in \mathcal{W}[i]}$ **then**
    $\mathcal{W}[i] \leftarrow \mathcal{W}[i] \setminus r_M^L[n]$
  while $\underline{|\mathcal{W}[i]| \leq g}$ **do**
    $l \leftarrow \mathcal{R}[i][\sigma[i]]$;
    $\sigma[i] \leftarrow \sigma[i] + 1$;
    if $\underline{l \notin r_M^L}$ **then**
      $\mathcal{W}[i] \leftarrow \mathcal{W}[i] + l$;
  $\chi[i] \leftarrow |\mathcal{W}[i] \cap E|$;

r_A	tp	r_B	tp
(1,2)		(5,18)	x
(1,4)	x	(1,2)	
(5,6)	x	(8,9)	
(6,12)	x	(5,6)	x
(5,18)	x	(7,11)	x
(3,4)		(6,9)	x
(4,9)	x	(1,14)	
(7,11)	x	(2,9)	

$\longrightarrow$

r_A	tp	r_B	tp
(1,2)		(5,18)	x
(1,4)	x	~~(1,2)~~	
(5,6)	x	(8,9)	
(6,12)	x	(5,6)	x
(5,18)	x	(7,11)	x
(3,4)		(6,9)	x
(4,9)	x	(1,14)	
(7,11)	x	(2,9)	

$\longrightarrow$

r_A	tp	r_B	tp
(1,2)		(5,18)	x
(1,4)	x	~~(1,2)~~	
(5,6)	x	(8,9)	
(6,12)	x	(5,6)	x
~~(5,18)~~	x	(7,11)	x
(3,4)		(6,9)	x
(4,9)	x	(1,14)	
(7,11)	x	(2,9)	

Table 1: Two steps illustrating the merging algorithm with rankings r_A and r_B ($g = 5$). Pairs predicted (i.e. $\in r_M^L$) have green backgrounds. Pairs with gray backgrounds are in the windows $\mathcal{W}_A$ and $\mathcal{W}_B$. Barred pairs with red backgrounds have already been predicted and cannot be selected anymore.

if ranking r_i has been selected at step s on the learning set, then r_i should be selected at step $\left\lceil s \cdot \frac{n_T}{n_L} \right\rceil$ on the test set.

4.2 Benchmarks for comparison

In order to assess the efficiency of *RankMerging*, we compare its performances to existing techniques. The first one is Borda's method, introduced above. We also consider classic supervised techniques. As we aim at handling large network datasets, we restricted ourselves to computationally efficient ones, namely nearest neighbors (NN), classification trees (CT) and *AdaBoost* (AB). We have used implementations from Python scikit learn toolkit[1]. These techniques are not specifically designed for ranking tasks, so we obtain different points in the precision-recall space by playing with the algorithms parameters, respectively the number of neighbors (NN), the minimum size of a leaf (CT), and the number of trees (AB).

4.3 Protocol and results

Protocol. According to the description in 4.1, ρ_i are computed on $\mathcal{G}_{Learn}$ to discover links among A_2 nodes, and then used to merge rankings on $\mathcal{G}_{Test}$ to discover links between B nodes, applying the scaling factor $n_T/n_L \approx 1.5$ to adapt the ρ_i learnt to the test set and cross-validation is therefore made through a simple hold-out strategy. Determining adequate parameters is not an issue here. The user may indeed aggregate as many rankings as he wants: the merging process is such that the addition of a supplementary ranking is computationally cheap[2], and if a ranking does not bring additional information, it will simply be ignored during the learning process. This property helps *RankMerging* to avoid strong overfitting effects. As of the value of g, our numerical experiments show that the performance of the algorithm is robust over a large range of values (see Table 2), and we extrapolate the best g value on $\mathcal{G}_{Learn}$ for the aggregation on $\mathcal{G}_{Test}$.

Results. We plot on Figure 2 the evolution of the **F**-score and the precision-recall curve obtained with *RankMerging*, for $g = 200$, aggregating the rankings of the following classifiers: AA_w, CN_w, CN, $Jacc_w$, $Katz_w$ ($\beta = 0.01$), PA_w, RWR_w ($p = 0.8$) and Borda's method applied to the seven former ones. We observe that *RankMerging* performs better than Borda, especially for intermediary recall values. This was expected, as *RankMerging* incorporates the information of Borda's aggregation here. In fact, the method has been designed so that *any* unsupervised ranking can be aggregated without any performance loss, so that it should outperforms any unsupervised method taken into account during the learning phase. We measure the area under the **Pr**-**Rc** curves to quantify the performances with a scalar quantity. *RankMerging* increases the area by 8.3% compared to Borda. Concerning the supervised benchmarks, we observe that they perform well, but only for a low number of predictions (comparable to Borda for approximately 1000 to 2000 predictions). Unsurprisingly, *AdaBoost* is

[1] `http://scikit-learn.org/`

[2] Here, the running time is a few seconds on a standard personal computer.

an ensemble method and outperforms *Decision Trees* and *Nearest Neighbors* for an optimal parameter choice, but the performances are of the same order of magnitude, in line with the observations in [1]. As formerly stated, these methods are not designed to cover a wide range of the precision-recall space, and therefore perform very poorly out of their optimal region of use. On the minus side, *RankMerging* has been designed for classification problems with large number of predictions. The window size g implies an averaging effect which causes the method to lack efficiency on the top-ranked items, as can be seen on Fig. 2. As a consequence, it is not suited to problems with low number of predictions, as it is often the case for information retrieval tasks for example.

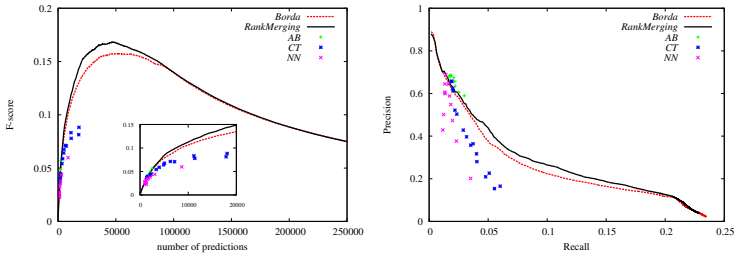

Figure 2. Results of *RankMerging* on the test set ($g = 200$), compared to benchmarks. Left: **F**-score as a function of the number of predictions. Right: **Pr** versus **Rc** curves.

We evaluate the influence of the structural metrics in Table 2. As can be seen, the addition of a ranking does not decrease the quality of the merging process – except for small variations which are considered as statistical fluctuations. A user may therefore aggregate any ranking whatever the source of information is. The dependency on the value of g is also shown in Table 2, and results indicate that the performances are close to the maximum within the interval $[100; 300]$ on both the learning and test sets. This observation suggests the possibility of tuning g in the testing phase from the values of g during the learning process.

AA_w	CN_w	CN	$Jacc_w$	$Katz_w$	PA_w	RWR_w	$Borda$	imp.(%)
x	x	x	x	x	x	x	x	8.3
x	x	x	x	x	x	x		3.2
x	x	x	x	x	x			-0.7
x	x	x	x	x				-1.0
x	x	x	x					-2.0
	x	x	x	x	x	x	x	8.2
		x	x	x	x	x	x	8.1
			x	x	x	x	x	3.7
				x	x	x	x	3.8

g	imp.(%)
10	-0.8
100	5.5
200	5.4
300	5.2
400	5.0
500	4.7
1000	4.0
2000	2.7

g	imp.(%)
10	2.7
100	8.2
200	8.3
300	7.9
400	7.4
500	7.2
1000	6.4
2000	5.6

Table 2: Left: Improvement (in %) to Borda's method of the area under the curve in the precision-recall space, for the aggregation of different rankings. Right: Improvement to Borda's method of the area under the curve in the precision-recall space, for different values of g; left: learning set, right: test set.

5 Conclusion

We presented *RankMerging*, a supervised machine learning framework which combines rankings from any unsupervised classifier to improve the performance of link prediction. This method is straightforward and computationally cheap as its complexity is $O(n.\alpha)$, where α is the number of rankings aggregated and n the number of predictions. It is adapted to prediction in social networks, as n can be tuned according to the users' needs. On the other hand, the precision on top-ranked items is not as high as the results yielded by supervised methods designed for information retrieval. In the case of a PSP, considered in this paper, this parameter would adjust the number of predictions to its commercial strategy.

So far, we have exclusively focused on structural information in order to predict unknown links. However, the framework is generic and any feature providing a ranking for likely pairs of nodes can be incorporated. Additional structural classifiers are an option, but other types of attributes can also be considered, such as the profile of the users (age, hometown etc.), or timings of the interactions. In the latter case, for instance, if i and j are both interacting with k within a short span of time, it is probably an indication of a connection between i and j. From a theoretical perspective, *RankMerging* provides a way to uncover the mechanisms of link creation, by identifying which sources of information play a dominant role in the quality of a prediction. The method could be applied to other types of networks, especially when links are difficult to detect. Applications include network security, for example by detecting the existence of connections between machines of a botnet, and biomedical engineering, for screening combinations of active compounds and experimental environments in the purpose of medicine discovery.

Acknowledgements

We thank E. Viennet, M. Danisch and anonymous reviewers for useful bibliographical indications. This paper presents research results of the Belgian Network DYSCO, funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors. We also acknowledge support from FNRS and the European Commission Project *Optimizr*.

References

1. M. Al Hasan *et al.* Link prediction using supervised learning. In SDM'06: Workshop on Link Analysis, Counter-terrorism and Security, 2006.
2. L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In WSDM'11, pages 635–644. ACM, 2011.
3. N. Benchettara *et al.* Supervised machine learning applied to link prediction in bipartite social networks. In ASONAM'10, pages 326–330. IEEE, 2010.
4. C. Bliss *et al.* An evolutionary algorithm approach to link prediction in dynamic social networks. arXiv:1304.6257, 2013.

5. C.J.C. Burges *et al.* Learning to rank using an ensemble of lambda-gradient models. Journal of Machine Learning Research-Proceedings Track, 14:25–35, 2011.
6. P.M. Comar *et al.* Linkboost: A novel cost-sensitive boosting framework for community-level network link prediction. In ICDM'11, pages 131–140. IEEE, 2011.
7. K. Dasgupta *et al.* Social ties and their relevance to churn in mobile telecom networks. In EDBT'08, pages 668–677. ACM, 2008.
8. D. Davis *et al.* Supervised methods for multi-relational link prediction. Social Network Analysis and Mining, 3(2):127–141, 2013.
9. J.C. de Borda. Mémoire sur les élections au scrutin. 1781.
10. C. Drummond and R.C. Holte. Explicitly representing expected cost: An alternative to roc representation. In KDD'00, pages 198–207. ACM, 2000.
11. C. Dwork *et al.* Rank aggregation methods for the web. In WWW'01, pages 613–622. ACM, 2001.
12. Y. Freund *et al.* An efficient boosting algorithm for combining preferences. Journal of machine learning research, 4:933–969, 2003.
13. H. Jeong *et al.* Measuring preferential attachment in evolving networks. EPL, 61(4):567, 2003.
14. H. Kashima *et al.* Link propagation: A fast semi-supervised learning algorithm for link prediction. In SDM'09, volume 9, pages 1099–1110. SIAM, 2009.
15. L. Katz. A new status index derived from sociometric analysis. Psychometrika, 18(1):39–43, 1953.
16. G. Kossinets and D.J. Watts. Empirical analysis of an evolving social network. Science, 311(5757):88–90, 2006.
17. J. Leskovec *et al.* Microscopic evolution of social networks. In KDD'08, pages 462–470. ACM, 2008.
18. D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. JASIST, 58(7):1019–1031, 2007.
19. R.N. Lichtenwalter *et al.* New perspectives and methods in link prediction. In KDD'10, pages 243–252. ACM, 2010.
20. G. Linden *et al.* Amazon. com recommendations: Item-to-item collaborative filtering. Internet Computing, 7(1):76–80, 2003.
21. Y.T. Liu *et al.* Supervised rank aggregation. In WWW'07. ACM, 2007.
22. L. Lü and T. Zhou. Link prediction in complex networks: A survey. Physica A, 390(6):1150–1170, 2011.
23. Z. Lu *et al.* Supervised link prediction using multiple sources. In ICDM'10, pages 923–928. IEEE, 2010.
24. T. Murata and S. Moriyasu. Link prediction of social networks based on weighted proximity measures. In ICWI, pages 85–88. IEEE, 2007.
25. B. Ngonmang *et al.* Churn prediction in a real online social network using local community analysis. In ASONAM'12, pages 282–288. IEEE, 2012.
26. M. Pujari and R. Kanawati. Supervised rank aggregation approach for link prediction in complex networks. In WWW'12 Companion. ACM, 2012.
27. T. Raeder *et al.* Predictors of short-term decay of cell phone contacts in a large scale communication network. Social Networks, 33(4):245–257, 2011.
28. S. Scellato *et al.* Exploiting place features in link prediction on location-based social networks. In KDD'11, pages 1046–1054. ACM, 2011.
29. T. Tylenda *et al.* Towards time-aware link prediction in evolving social networks. In SNA-KDD'09, page 9. ACM, 2009.
30. F. Wei *et al.* irank: A rank-learn-combine framework for unsupervised ensemble ranking. JASIST, 61(6):1232–1243, 2010.
31. T. Zhou *et al.* Predicting missing links via local information. EPJB, 2009.