Use of Supervised Learning to Predict Directionality of Links in a Network

Sucheta Soundarajan and John E. Hopcroft

Cornell University, Ithaca NY 14853, USA

Abstract. Often, the information contained in network data is incomplete. Many avenues of research are aimed at addressing this incompleteness. For example, the link prediction problem attempts to identify which missing links are most likely to exist in the complete network. In this paper, we consider a related, but different, problem: predicting the directions of links in a directed network. We treat this problem as a supervised learning problem in which the directions of some edges are known. We calculate various features of each known edge based on its position in the network, and use a Support Vector Machine to predict the unknown directions of edges. We consider four networks, and show that in each case, this method performs significantly better than other compared methods.

1 Introduction

Over the past few decades, network analysis has become an increasingly important way for scientists to study the relationships, interactions, and roles of individuals in groups. Although scientists and engineers have developed sophisticated, valuable techniques for studying networks, any such methods are inherently limited by the quality of the data.

Much of available data is incomplete. In many cases, actual nodes or edges may be missing from a network. For example, it is believed that in some genetic networks, where links are experimentally determined, fewer than 1% of the edges have been discovered. Even in cases where information is not gathered experimentally, networks may be incomplete due to other factors, such as lack of participation (e.g., individual people may choose not to participate in Facebook, and even if they do participate, they may not connect to all of their actual friends). In other cases, while the existence of nodes or links may be known, we may be missing information about their characteristics.

This type of information is valuable to many social network analysis techniques: for instance, community detection algorithms will work more accurately on a more complete network, and algorithms to predict the flow of information through a network rely on accurate knowledge of link directionality. Thus, inferring this sort of missing information is an especially active and important research area.

In this paper, we consider the problem of predicting the direction of links. In much of network data, links may be reported as undirected, and yet the underlying network may actually consist of directed links. For example, in Facebook, all links (friendships) must be reciprocated (in contrast with a network like Twitter, where one user may follow another without reciprocation). However, even in a network such as Facebook, a link was likely initiated by one person and may be stronger in that direction. Although a researcher studying a Facebook network is given undirected data, information about the underlying direction of the links may help, for instance, to more accurately predict the spread of news or trends through the network.

We treat this problem as a supervised learning problem. We assume that we are given the directions of some links, and attempt to use this information to predict the directions of other links. In the example of Facebook, such a situation may arise if one can use information about the timing of similar posts, and then infer the direction of some friendships (e.g., if one user consistently posts articles, news, or events before another user, we might infer that the second user is following the first user, and so the edge goes from the second user to the first user). See Adamic, et al. [2] for an analysis of such a situation. Although one might use this type of information to predict the directions of some links, it is unlikely that similar information will be available for every pair of linked users. In such cases, one can apply supervised learning techniques to predict the unknown directions of these links.

In our experiments, we consider four networks from different domains, each containing directed links. We train a Support Vector Machine (SVM) classifier on some of these links, and use this model to predict the directions of other links (the directionality of which is withheld, but the existence is assumed). For each link, we create a feature vector that includes information about its position in the network. Some of these features are obtained from other algorithms intended to solve this same problem, while other features represent more general node, edge, and network features. Each edge (a, b) is labeled with one of three possible class memberships, indicating whether the edge is from a to b, b to a, or both. We compare our algorithm to four other methods, and show that it is the best performer on every network.

This paper begins with an overview of related work, including work from the areas of link prediction, as well as detailed discussions of two other methods for predicting link directionality. We then discuss the methodology used in our work, beginning with a description of our datasets and calculations used to produce the feature vectors, and continuing with detailed information on our experiments. We next discuss and analyze our results, and conclude with a discussion of future work.

2 Related Work

Much work has been done in the related area of link prediction, and some of these methods have proven valuable for the problem of predicting link direction. In

this section, we will begin with a description of relevant link prediction methods. We then describe various methods for predicting link directions, including some that rely on external information (such as timing of blog posts), and others that use only the structure of the network.

2.1 Link Prediction

Many popular link prediction methods use information about the structure of the network to determine the probability that two nodes are connected. One simple method, the Common Neighbors metric, simply calculates the number of neighbors that two nodes share, with the assumption that nodes with many shared neighbors are likely to be connected. This method and its variants are fast, simple, and reasonably accurate [3]. Although these methods are not directly useful for our task in this paper, they demonstrate that simple, local metrics can be valuable in inferring missing information in networks.

These metrics and others can be incorporated into a supervised learning framework, as we do in this paper. For example, Al Hasan et al. approach this problem by applying SVM and k-Nearest Neighbors methods to feature vectors that include both topological features and external features (such as keywords in papers), with good results [1].

2.2 Predicting Link Direction

Here, we discuss three methods for predicting link direction. The first method resembles ours, in that it uses a supervised learning method; however, it uses external, non-topological information. The other two methods are ranking methods, intended to place the nodes into a hierarchy. Under these methods, link directionality is assumed to go from lower-ranked nodes to higher-ranked nodes.

Supervised Learning Adar and Adamic consider the problem of tracking information flow through blogspace [2]. Bloggers online often see posts or information in another blog, and then repost that information to their own blog. However, this information is often not cited, and so it may be unclear where the blogger learned the information.

In some cases, bloggers do cite the source of their posts, or explicit links may be declared. Adar and Adamic use these cases as positive examples in their classifier models. The features they consider include textual similarity between blog posts, similarity between posted URLs, and relative timing of blog posts. A two-class SVM trained on these features resulted in high cross-validation scores, demonstrating the value of a supervised learning framework to this problem. This work differs from ours in that it relies on external information beyond the topology of the network.

PageRank PageRank is a very popular, effective algorithm that uses the topology of a directed graph to rank nodes according to their importance in the network [7]. Although this algorithm is probably best known for its use in online search engines, it can also be used as a predictor of link directionality. The output of the algorithm represents the probability that a random walk on the network will arrive at a particular node. The algorithm also includes a 'restart value,' or probability that the random walk will restart back to its first node on any given step of the walk.

A node's PageRank depends on the PageRank of other nodes: if many highly ranked nodes point to some node, then that node is believed to be of high importance, and so will have a high PageRank score. A PageRank score can be used to predict link direction by assuming that an edge between a node with a low PageRank and a node with a high PageRank is directed from the former to the latter.

Leader-Follower Ranking We also consider another algorithm for ranking nodes, again with the assumption that edges go from lower-ranked nodes to higher-ranked nodes. This algorithm, by Guo et al. [4], is based on a recursive algorithm that partitions the network into 'leaders' and 'followers.' The user specifies a value α between 0 and 1 that defines the fraction of leaders at each step. In each step of the algorithm, for each node n, the algorithm calculates δ_n, defined as the difference between node n's in-degree and out-degree. Nodes with a high δ_n are considered leaders, whereas nodes with a low δ_n are considered followers.

After partitioning the graph into leaders and followers, the algorithm recursively calls itself again on each of these two sets. At the lowest level, if the number of nodes is less than $\frac{1}{\alpha}$, the algorithm returns the nodes ranked in order of their δ_n. When joining together two sets, the algorithm places the followers below the leaders.

Formally,

$$\mathrm{R}_G(x) = \begin{cases} \mathrm{D}_x & \text{if } |G| < 1/\alpha \\ \mathrm{R}_{G_L(x)} & \text{if } |G| \geq 1/\alpha \text{ and } x \in L \\ |L| + R_{G_F(x)} & \text{if } |G| \geq 1/\alpha \text{ and } x \notin L \end{cases}$$

where G is the graph currently being considered, $R_G(x)$ is the ranking of node x, D_x is the ranking of node x induced by the δ_n values, L is the set of leaders in G (and G_L is the subgraph of G induced by L), and F is the set of followers in G (and G_F is the subgraph of G induced by F).

The output of this algorithm is a list of nodes, where the index of the node corresponds to its ranking. Because no two nodes can share the same position in a list, it is impossible for two nodes to have an equal rank. Because in our problem of predicting link directions, we want to allow for reciprocated links, we thus modify the algorithm slightly to allow nodes to share a rank (we accomplish this by allowing leaders who share a δ_n to have the same rank, rather than requiring a completely ordered list).

To determine the optimal α value, one can consider several different values, and determine which value produces a ranking that conforms best to the actual directions of edges in the graph.

3 Methodology

We treat the problem of predicting link direction as a classification problem. To accomplish this, we collect a set of four directed networks. For each network we create a set of training edges, for which directionality is known, and a set of test edges, for which directionality is withheld. For edges in the test set, we assume that the existence of that edge is known, but the direction is unknown. For each edge, we calculate a feature vector that contains information about that edge's location in the network, and then determine how well a classifier can use this feature vector to predict edge directionality. In this section, we first describe the network datasets used in our experiments, giving background information and statistics on each. We then discuss the various network, node, and edge features used to characterize edges in the networks. We conclude with a description of our experimental methodology, including information on the creation of test and training sets, as well as a brief background on the classification method used.

3.1 Datasets

In this section, we describe the datasets used in our experiments. We consider 4 directed networks from different domains. Three of these networks (Amazon, Epinions, and Slashdot) are quite large, and so for these networks we considered a 5000 node subgraph of each, where the 5000 nodes were chosen using a breadth-first search beginning from a randomly chosen node. All datasets were downloaded from the online dataset collection at SNAP, the Stanford Network Analysis Project.

Amazon "Amazon" is a product co-purchasing network obtained from the online retailer Amazon.com [9]. For each item, Amazon.com reports up to five other items that were frequently bought with that item. An edge from node A to node B indicates that customers who bought item A also frequently bought node B. The original Amazon network contained nearly 300,000 nodes, and as described above, we consider a 5000 node subset of this full network.

Epinions "Epinions" is a social network from the review website Epinions.com [12]. Each node represents a user of the website, and a link from user A to user B indicates that user A 'trusts' user B's reviews. This network contains approximately 75,000 nodes, and as with Amazon, we consider a 5000 node subgraph of the larger network.

Slashdot "Slashdot" is a social network from the technology website Slashdot.com [11]. In this website, users submit news and reviews, and are able to mark other users as friends or foes. Each node represents one user, and a directed link from A to B indicates that user A has tagged user B as either a friend or foe. This network originally contained approximately 80,000 nodes, so we again consider a 5000 node subgraph.

Wiki "Wiki" is a social network from the free online encyclopedia Wikipedia.com, which contains content created and edited by users [10]. Some of the users are elected to be administrators with special privileges. In this network, each node is one user, and a directed link from user A to user B indicates that user A participated in user B's election. This network is fairly small, containing only 7115 nodes, and so we do not need to consider a subgraph.

3.2 Network Features

To produce features for each edge, we consider two graphs: N, the directed network formed from the edges in the training set, and G, an undirected network formed from edges in both the training and the test set (we include edges from the test set because in our problem, the directionality of the edge is unknown, but its existence is given). To create G, we simply ignore the directions on edges, and convert all of them to undirected edges. For edge (a, b) in the training or test sets for each network, we calculate 9 features, listed below. To calculate the degree and betweenness features, we consider the undirected graph G that is produced by converting each directed edge into an undirected edge, and for the various ranking features, we consider the directed graph N induced by the edges in the training set.

The first 5 features are calculated using the undirected graph G. These features have traditionally not been used to order nodes, but we consider them here in order to learn whether they can be useful in predicting the direction of an edge. For example, for an edge (a, b), we consider the degrees of both nodes a and b. It may be the case that nodes tend to be directed from low degree nodes to high degree nodes (or vice versa), and so this feature might increase prediction accuracy. The node betweenness features might play a similar role.

1. Degree of node a: The degree of a node is simply the number of edges adjacent to it. To calculate this value, we use undirected graph G.
2. Degree of node b
3. Node Betweenness for node a: The node betweenness feature, proposed by sociologist Linton Freeman, is a measure of a node's centrality in the network [5]. There are different forms of this metric: the one that we use is simply the fraction of all shortest paths between all pairs of nodes that pass through the node in question. That is, for a given node a, we consider all other pairs of nodes b, c, and determine how many of the shortest paths from node b to node c pass through node a. We then normalize this by the total number of

shortest paths between all pairs of nodes b, c. To determine node betweenness values, we use undirected graph G.

4. Node Betweenness for node b
5. Edge Betweenness for edge (a, b): The edge betweenness feature is defined similarly to the node betweenness factor, except it indicates the fraction of shortest paths that use a particular edge. For edge betweenness, we again use undirected graph G.
6. Leader-Follower Ranking for node a: To calculate the rank of a node, we apply the algorithm described in [4] to directed graph N. The original algorithm does not allow for two nodes to have equal rank (rather, it produces a list of nodes, where the index of a node is its rank), so as described earlier, we modify the algorithm slightly by allowing two nodes to share an index in the ordered ranking list. In this metric, a node that is highly ranked receives a low score (that is, it appears early in the list).
7. Leader-Follower Ranking for node b
8. PageRank of node a: To calculate a node's PageRank, we apply the well-known algorithm that calculates the probability that a random walk will include that particular node. In our calculations, we apply a restart factor of 0.15, representing the probability that the random walk returns to its starting node at any given step.
9. PageRank of node b

3.3 Experiments

To predict directionality of links, we use a Support Vector Machine (SVM) classifier. In our experiments, we assume that the existence of each edge is known, and attempt to predict its direction. For each network, we create a set of training links, for which the direction is known, and test links, for which direction is unknown. The training links constitute 90% of the edges in the network, while the other 10% of edges are withheld for testing.

For some of the features described above, we calculate feature values using a graph G containing undirected links from both the training and test sets; for the other features, we create a graph N that uses only directed links from the training set. Because we need every node to appear in N, some caution is necessary in partitioning the edges into training and test sets; in particular, we must ensure that the edges in the training set describe a graph that has only one component, and that we do not create 'orphan' nodes by placing all of a node's edges into the test set. Thus, to produce the training set, we first find a minimum spanning tree on network G, and then continue adding edges to this tree until 90% of the edges from the network are present.

For each edge (a, b) in the training set, we assign one of three class labels: Class 0 indicates that the edge is reciprocated (both nodes a and b link to each other), Class 1 indicates that the edge is directed from node a to node b, and Class 2 indicates that it is directed from node b to node a. From this set of edges, we randomly sample a maximum of 1000 elements from each class to train the

SVM classifier. Network Amazon had a small number of reciprocated edges, and so to maintain class balance, each class contains only 400 elements.

For the test set, we again sample up to 1000 elements from each class (again, less for Amazon). In both the training and test sets, it is possible that some edges appear twice, as both (a, b) and (b, a). These two elements would have different class labels (unless the edge is reciprocated, in which case both would be labeled as Class 0), and the feature vector for element (b, a) would simply be a reordering of the feature vector for element (a, b).

We then apply the SVM classifier from the LibSVM software package [6] to produce a classifier model for these training sets. A SVM classifier accepts as input a set of data points (feature vectors) labeled with classes, and then projects the feature vector into a higher dimensional space and attempts to divide the classes with a hyperplane. Because data is typically noisy, the method that we use allows for a 'soft margin,' in which points may appear on the wrong side of the dividing hyperplane, with a penalty. In order to allow for multiple classes, LibSVM creates many two-class models, one for each pair of classes, and then combines these models into a single multi-class model. In this software, a cross-validation grid search is used to select optimal parameters (such as for the soft margin penalty) for the SVM.

When the model is applied to the test set, each element in the test set is assigned a probability vector, indicating the probability that the element belongs to each of the three classes. An element is then assigned to whichever class receives the bulk of its probability mass. However, because each element is assigned a probability vector, we are able to identify relationships between the three different classes: for example, we might observe that elements in class 1 (edges directed from node a to node b) are also similar to elements in class 0 (reciprocated edges), but are quite different from elements in class 2 (edges from node b to node a).

We perform 4 sets of experiments. First, we create a SVM model using the full feature vectors (SVM-Full). Next, we consider only those features that are obtained from the Leader-Follower Ranking algorithm (SVM-LFRank). Then, we consider only those features obtained from the PageRank algorithm (SVM-PR). Finally, we measure accuracy by using both PageRank and Leader-Follower (SVM-PR-LFRank). Although both PageRank and our modified version of the Leader-Follower Ranking algorithm allow for two nodes to have equal rank (thus predicting a reciprocated edge between the two nodes), it is likely that there is some small, non-zero difference between two nodes' rankings that would still indicate a reciprocated edge; that is, if two nodes have very similar, but slightly different, rankings, we may still wish to predict that each is linked to the other. Training a SVM model on these features allows us to discover this similarity threshold, and so allows for more accurate predictions.

We additionally calculate accuracy using PageRank and Leader-Follower without an SVM.

Table 1. Accuracy scores for each method and network.

	SVM-Full	SVM-PR	SVM-LFRank	SVM-PR-LFRank	PageRank	Leader-Follower
Amazon	68%	67%	64%	65%	61%	58%
Wiki	78%	60%	76%	77%	47%	59%
Epinions	69%	59%	65%	65%	53%	60%
Slashdot	73%	44%	67%	68%	41%	62%

4 Results

Our results show that using a SVM classifier framework to predict directions of links is successful. Our first experiment, in which we used the entire feature vector, resulted in higher accuracy scores than using feature vectors containing either PageRank or the Leader-Follower Ranking scores alone, and also higher than the accuracy obtained by using PageRank and Leader-Follower together. As expected in all of these results, Classes 1 and 2 behave close to symmetrically.

Table 1 contains the results of our experiments. The columns correspond to the accuracy scores from 6 different experiments- the four SVM experiments described above, as well as using PageRank and Leader-Follower Ranking scores alone, without a SVM. The percentage represents the fraction of elements from the test set that were correctly classified into the proper class.

Notably, SVM-Full outperforms every other method for each of the 4 networks. Although in some cases, either SVM-PR, SVM-LFRank, or SVM-PR-LFRank perform nearly as well, these other three methods are much less consistent (for example, SVM-PR scores nearly as well as SVM-Full on network Amazon, but much worse on network Slashdot). SVM-PR-LFRank also performs fairly well, but is typically several percentage points behind SVM-Full. The two non-SVM methods tended to perform poorly. Further examination of the data shows that these methods tend to misclassify elements from Class 0 (reciprocal links) as one of the other two classes. This is to be expected, because, as described earlier, for these methods to classify a link as reciprocal, the two nodes must have exactly the same rank. Using a SVM allows some variability in this rank comparison.

Table 2 describes how the elements of each class were classified. For example, 54.2% of elements from Class 0 in Amazon were classified as Class 0, 19.7% as Class 1, and 26.1% as Class 2. We see that, typically, the SVM achieves very high accuracy rates for the directed classes, but lower accuracy for the undirected class. When a directed edge is misclassified, it is usually classified as undirected, and only very rarely is it classified as belonging to the other class of directed edges.

Table 2. SVM-Full Classification. The value in row R, column C is the fraction of elements in class R that were classified as class C.

		Class 0	Class 1	Class 2
Amazon	Class 0	54.2%	19.7%	26.1%
	Class 1	22.1%	73.1%	4.7%
	Class 2	16.2%	7.0%	76.8%
Wiki	Class 0	81.0%	10.4%	8.6%
	Class 1	23.0%	76.3%	0.6%
	Class 2	19.4%	2.7%	77.9%
Epinions	Class 0	57.4%	21.1%	21.4%
	Class 1	19.8%	75.8%	4.5%
	Class 2	21.3%	3.7%	75.0%
Slashdot	Class 0	53.5%	19.3%	27.1%
	Class 1	15.3%	81.2%	3.5%
	Class 2	11.1%	4.1%	84.9%

Table 3 contains the average feature values for each class. Notably, the $LF_b - LF_a$ and $PR_b - PR_a$ features tend to show high variance between the different classes: Class 1 always has a low value of $LF_b - LF_a$, and a high value of $PR_b - PR_a$, while the opposite holds true for Class 2. Class 0 typically has average values close to 0 for these scores. However, recall from Table 1 that these metrics alone gave poor accuracy. This occurs for two reasons: first, these methods will only predict a reciprocal link if the two nodes have identical ranks, and second, because although they perform well on average, there are many elements where they fail.

These results demonstrate two key points. First, although ranking methods such as PageRank and Leader-Follower Ranking score reasonably well on their own, they are vastly improved with use of a supervised classifier. Second, relatively simple metrics such as degree, node betweenness, and edge betweenness can further boost the accuracy of methods for predicting the direction of a link.

5 Conclusion and Future Work

In this paper, we have considered the problem of predicting the directions of links in a network, motivated by the frequent lack of complete network data. Solutions to this problem may be useful for networks such as Facebook, which represents every link as undirected, even though the underlying social network is likely to be directed. Determining the directions of links will allow researchers

Table 3. Average Feature Value per Class: D_a is degree of node a, NB_a is node betweenness of node a, EB is edge betweenness, LF_a is leader-follower ranking of node a, PR_a is PageRank of node a.

		D_a	D_b	NB_a	NB_b	EB	$LF_a - LF_b$	$PR_a - PR_b$
Amazon	Class 0	7.8	8.1	0.0015	0.0018	2.78e-05	10.4	4.5e-0.5
	Class 1	6.3	26.3	0.0013	0.0142	9.40e-05	-1550.0	0.0015
	Class 2	25.8	6.2	0.0146	0.0019	10.0e-05	1532.0	-0.0016
Wiki	Class 0	219.2	214	0.0044	0.0041	1.01e-07	-214.1	7.3e-05
	Class 1	195.4	124.4	0.0040	0.0022	1.21e-07	-2905.6	0.0003
	Class 2	120.5	191.1	0.0020	0.0040	1.25e-07	3656.7	-0.0003
Epinions	Class 0	169.0	180.3	0.0021	0.0033	2.40e-08	-3.7	2.45e-05
	Class 1	146.6	210.1	0.0043	0.0028	3.96e-08	-1762.4	0.0007
	Class 2	213.5	164.4	0.0029	0.0058	3.92e-08	1741.1	-0.0007
Slashdot	Class 0	91.8	95.4	0.0018	0.0020	1.38e-07	-98.9	-6.69e-07
	Class 1	83.4	84.8	0.0015	0.0014	1.28e-07	-2335.4	0.0002
	Class 2	84.7	81.7	0.0013	0.0014	1.23e-07	2264.1	-0.0002

to more accurately perform network analysis tasks, such as tracing the spread of information through the network.

We treat this problem as a supervised learning problem in which the directions of some edges are known and the directions of other edges are unknown. We considered 4 network datasets from different domains, each containing both directed and reciprocated edges. For each network, we partitioned the edges into a training set, containing 90% of the edges, and a test set containing the remaining 10% of the edges. For each edge, we calculated several features related to the edge's position in the network topology. To calculate these features, we used two graphs: an undirected version of the complete graph (including edges in the test set), and a directed version of the graph represented by the training set. We compared our methods to PageRank and Leader-Follower Ranking, two other methods for predicting link directionality. Both of these methods rank nodes based on their importance in their networks, with the assumption that links are likely to go from low-value nodes to high-value nodes. In our experiments, we saw that the SVM trained on the full feature vector produced results substantially better than other methods. A detailed analysis of our results showed that directed links most often tended to be confused with undirected links, rather than links of the other direction. Our method achieved very high accuracy on the directed links.

Interestingly, although the two ranking methods showed high inter-class variability, they performed much more poorly than the other methods at classifica-

tion. When the scores from these methods were used in a classifier, we were able to achieve much higher accuracy results, even though there was no additional information. This is likely due to these methods' poor ability to identify reciprocated links. This issue can be remedied through use of a SVM; however, adding further network topological information to these features provided an additional gain in accuracy. Of the additional features we considered, node degree seemed the most valuable in general, though the node betweenness and edge betweenness features were occasionally valuable on individual networks.

This area of research has many interesting potential future directions. Most obviously, it is worthwhile to determine which features (including ones not considered in this paper) are most valuable for this task. We are also interested in incorporating external metadata (as in [2]), in order to learn how much additional accuracy such metadata can provide when used in conjunction with the features in this paper.

6 Acknowledgements

Funding for this work is provided by grant AFOSR FA9550-09-1-0675.

References

1. Al Hasan, M., Chaoji, V., Salem, S., Zaki, M.: Link prediction using supervised learning. SIAM Workshop on Link Analysis, Counterterrorism and Security with SIAM Data Mining Conference, (2006)
2. Adar, E., Adamic, L.: Tracking information epidemics in blogspace. '05 Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence, (2005)
3. Lu, L., Zhou, T.: Link prediction in complex neworks: a survey. Phys. A: Stat. Mech., 390 (6), 1150-1170 (2011)
4. Gup, F., Yang, Z., Zhou, T.: Predicting link directions via a recursive subgraph-based ranking. http://arxiv.org/abs/1206.2199 (2012)
5. Freeman, L.C., Borgatti, S.P., White, D.R.: Centrality in valued graphs: a measure of betweenness based on network flow. Social Networks, 13, 141-154 (1991)
6. Chang, C.-C., Lin, C.-J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2 (2011)
7. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the Web. Technical Report, Stanford Digital Libraries (1998)
8. Stanford Network Analysis Project: http://snap.stanford.edu/index.html, accessed 2012
9. Leskovec, J., Adamic, L., Adamic, B.: The dynamics of viral marketing. ACM Transactions on the Web, 1(1), (2007)
10. Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. CHI 2010, 1361-1370, (2010)
11. Leskovec, J., Lang, K., Dasgupta, A., Mahoney, M.: Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics, 6(1), 29-123, (2009)
12. Richardson, M., Agrawal, R., Domingos, P.: Trust Management for the Semantic Web. ISWC (2003)