Network community detection with edge classifiers trained on LFR graphs (extended version)

Twan van Laarhoven[1] and Elena Marchiori[2] *

Department of Computer Science, Radboud University
Nijmegen, The Netherlands

Abstract

A popular method for generating graphs with known community structure is the Lancichinetti-Fortunato-Radicchi (LFR) model. These graphs have been used for assessing the performance of existing community detection algorithms. In this paper we investigate the use of LFR graphs as training data for learning classifiers that discriminates between edges that are 'within' and 'between' communities. To this aim, we introduce a new principled method to train a linear classifier, such that the connected components of the 'within' edges match the training community structure of a given LFR graph. The LFR generator has a parameter that controls the extent to which communities are mixed, and hence harder to detect. Results show that classifiers trained on graphs with more mixing also work well when tested on LFR benchmark graphs generated using a less mixing, while they achieve mixed performance on real-life networks, with a tendency towards finding many communities.

1 Introduction

Community detection Community detection is the task of identifying communities in a graph. Informally, a community is a set of nodes, such that there are many edges inside the community and relatively few edges linking it to the rest of the graph (see the recent review by Fortunato [2010]). Here we consider the traditional view of community structure of a graph as a partition of its nodes into groups such that each group is a community.

A simple way to find communities in a graph is to identify and remove edges that connect nodes belonging to different communities. The connected components of the resulting graph are the communities (see for instance the methods proposed by Newman and Girvan [2004], Radicchi, Castellano, Cecconi, Loreto, and Parisi [2004]).

The question of community detection then becomes a question of how to pick the set of edges to remove.

In this paper we investigate the effectiveness of a direct approach to this task, based on supervised learning. Specifically, we train a classifier to distinguish between edges to keep and edges to be removed. To this aim we introduce a new training method, that optimizes a linear classifier for this task.

Using supervised learning requires training data. In our context, we need graphs with a known community structure. Unfortunately, the communities are not usually known for real world graphs.

Nevertheless researchers have developed models that generate graphs with a known community structure, i.e. Girvan and Newman [2002], Lancichinetti, Fortunato, and Radicchi [2008]. These models are usually intended as benchmarks, for assessing the capability of community detection methods to find the

*This is an optional funding source acknowledgement.

correct community structure of a graph [see Lancichinetti and Fortunato, 2009]. In particular, recently Lancichinetti, Fortunato, and Radicchi [2008] have introduced the LFR benchmark. The LFR model accounts for the fact that complex networks are characterized by heterogeneous distributions of degree and community sizes, and that the degrees of nodes, as well as the sizes of communities follow a power law distribution. Moreover, the LFR graphs can be built efficiently, since the complexity of the construction algorithms is linear in the number of edges of the graph.

This paper investigates the use of LFR graphs as training data for building a classifier for discriminating between edges that are in communities versus edges that are between them. We consider a simple setting: linear classifiers acting on local features of an edge. Such features can be efficiently computed since they are based only on the degree of the edge's nodes and the number of triangles containing that edge. The following two main questions are investigated:

- Is there one classifier trained on a specific type of LFR graphs that also generalizes well to test graph instances generated using different LFR parameter settings?
- If such a classifier exists, what is its performance on real world graphs?

Extensive experiments indicate that we can build such a classifier. Its generalization performance on the considered real world graphs is mixed, with a tendency to detect a high number of clusters. These results indicate that it is possible to learn a simple supervised model based on few local topological features for identifying the community structure in artificial and real world networks.

Notation A graph $G = (V, E)$ consists of a set V of nodes and a set $E \subseteq V \times V$ of edges between them. Each edge $ab \in E$ connects two nodes $a, b \in V$. In this paper we consider undirected graphs, where $ab \in E$ if and only if $ba \in E$. Nodes connected by an edge are called *adjacent*. The degree d_a of node a is the number of nodes adjacent to it.

A path p through a graph is a sequence of nodes, such that there is an edge between each node to the next node in the sequence. We write $p = a, \ldots, b$ for a path from a to b; and E_p for the set of edges along the path.

A *community structure* on a graph is an equivalence relation $\sim$ on its nodes. We write $a \sim b$ to denote that nodes a and b belong to the same community.

Outline In section 2 we show how classifiers can be used for community detection. In section 3 we argue that a classifier trained directly on the edges of a graph can be suboptimal for the purpose of finding communities; and we therefore propose an alternative training method. In section 4 we address the problem of training data. In section 5 we report the results of experiments on artificial and real world graphs.

2 Community detection by classifying edges

The problem of detecting communities in a graph can be formulated as a binary classification task, where the goal is to decide whether each edge is 'within' a community or 'between communities'. A trained classifier is a function h that assigns a score to each edge. Given these scores, we can construct the *reduced subgraph*, containing only edges ab for which $h(\mathbf{x}_{ab}) > 0$. In other words, the reduced graph contains only edges classified as being 'within a community'. Then we report the connected components of this reduced graph as the original graph's communities.

Previous work has also used fixed score functions based on local features [Muff, Rao, and Caflisch, 2005, Radicchi, Castellano, Cecconi, Loreto, and Parisi, 2004, Ahn, Bagrow, and Lehmann, 2010]. An important advantage of training a classifier over using such a score function is that we can use more features than just the degrees and number of triangles. If more relevant edge features are available, we can expect to train a better classifier. On the other hand, incorporating extra features into a fixed score function is much more difficult.

2.1 Features

In order to use a classifier on edges, we need to associate a feature vector $\mathbf{x}_{ab}$ to each edge ab. There are several simple features that can be included,

- The degrees of two endpoints of the edge, d_a and d_b.
- The number of paths of length 2, 3, 4, etc. between a and b.
- Betweenness of the edge, as used by Girvan and Newman [2002].
- The mean number of triangles for all edges incident to node a, or other summary statistics.
- Nonlinear combinations of the above features, for example the number of triangles squared.
- Features specific to the graph under consideration.

For undirected undirected graphs, the features should be symmetric. That is, $\mathbf{x}_{ab} = \mathbf{x}_{ba}$. Therefore care must be taken when using features of nodes, such as the degree. A simple solution is to use minimum and maximum values, $\min(d_a, d_b)$ and $\max(d_a, d_b)$. An equivalent alternative would be to use the sum and absolute difference of the feature values.

The use of summary statistics as features is inspired by Satuluri, Parthasarathy, and Ruan [2011]. Their approach is to keep only the $\sqrt{d_a}$ highest ranked edges for each node a, where the edges are ranked according to some fixed score function. In that paper, the goal is graph sparsification, but we believe that the principle can also be used to help community detection. By including summary statistics, the classifier could for example learn to keep only edges with a number of triangles that is above the average of both adjacent nodes.

For the remainder of this paper we will use only minimum and maximum degrees, number of triangles (i.e. paths of length 2), and the mean values of these features for the two endpoints of each edge. This gives a total of 9 features, all of which can be calculated efficiently. Because the features are local, this calculation can in principle also be done in parallel for different parts of the graph.

2.2 Score functions as classifiers

Several scoring functions have been used in the literature to rank edges. Many of them can be formulated as linear classifiers using some of the above mentioned features. For instance, one of the score functions proposed by Radicchi et al. [2004] is the fraction of possible triangles that contain an edge ab,

$$s_{\text{Radicchi}}(ab) = \frac{t_{ab} + 1}{\min(d_a - 1, d_b b - 1)}, \tag{1}$$

where t_{ab} is the number of triangles containing the edge ab, which is equivalent to the number of paths of length 2 between a and b.

In their algorithm, the edges are ranked according to this score. The algorithm then iteratively removes the edge with the lowest score from the graph, until some stopping criterion is reached. At that point, the graph will contain only edges with scores greater than some threshold τ. That is, edges ab such that $s_{\text{Radicchi}}(ab) > \tau$. For the above score function we can rewrite this condition in the form of a linear classifier,

$$h(\mathbf{x}_{ab}) = \langle(\tau + 1, -\tau, 0, 1), \mathbf{x}_{ab}\rangle > 0,$$

where $\mathbf{x}_{ab} = (1, \min(d_a, d_b), \max(d_a, d_b), t_{ab})$.

The Jaccard similarity between the adjacency lists of a and b has also been used as a score function [Satuluri et al., 2011, Ahn et al., 2010]. If $\text{Adj}(a)$ denotes the set of nodes adjacent to a, this Jaccard similarity can be written as

$$s_{\text{Jaccard}}(ab) = \frac{|\text{Adj}(a) \cap \text{Adj}(b)|}{|\text{Adj}(a) \cup \text{Adj}(b)|}.$$

The condition $s_{\text{Jaccard}}(ab) > \tau$ can also be rewritten as a linear classifier, $\langle(0, -\tau, -\tau, 1 - \tau), \mathbf{x}_{ab}\rangle > 0$.

3 Learning methods

3.1 Edge-wise training

The learning problem looks like a standard linear classification problem. For each edge $ab \in E$ we could use the features $\mathbf{x}_{ab}$ defined above, and labels

$$y_{ab} = \begin{cases} +1 & \text{if } a \sim b \\ -1 & \text{if } a \not\sim b. \end{cases}$$

We call this the *edge-wise* classification problem. It can be solved by, for instance, a Support Vector Machine.

An edge-wise classifier will aim to minimize the expected number of errors. An error in this setting is an *edge* that is classified incorrectly. However, for the purpose of finding communities via connected components, not all incorrectly classified edges are equally bad.

Consider the graph in figure 1. When the classifier $t_{ab} \geq 2$ is used there are 4 edge errors, while the reduced graph has the correct community structure. On the other hand, using the classifier $t_{ab} \geq 1$ leaves the entire graph connected, but gives only 2 edge errors. Therefore an algorithm that minimizes the number of incorrectly classified edges might not perform optimally if the goal is to find the right community structure in the reduced graph.

An alternative approach would be to consider the problem of training a classifier for edges as a *weighted classification problem*, with a larger weight on false positives. That would require one to pick this weight, and as far as we know there is no good way to do this besides cross-validation. However, even with an optimal choice of the false positive weight, such a classifier can be sub-optimal.

Take the graph in figure 2. Here the training data has three communities, but the red diamonds could be considered errors in the community assignment. There are only four possible classifiers, deciding whether or not edges with 3 triangles are kept and whether or not edges with 0 triangles are kept. The

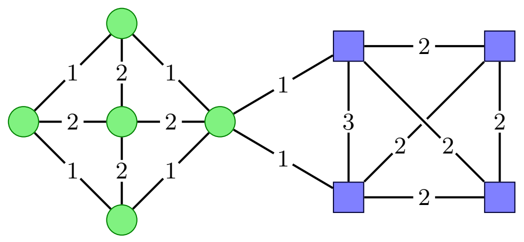

Figure 1: A graph with two communities, green circles and blue squares. As features for each edge only the number of triangles t_{ab} is used. Using the classifier $t_{ab} \geq 2$ yields the correct community structure, but 4 edge errors. Using the classifier $t_{ab} \geq 1$ leaves the entire graph connected, but gives only 2 edge errors.

optimal solution with regards to the normalized mutual information metric is to keep edges with 3 triangles, and remove the edge with 0 triangles. However, there is only one training instance of an edge with 0 triangles, and it is a within-community edge. So, a classifier trained that minimizes the number of incorrectly classified edges can not find the optimal solution.

3.2 Path-wise training

The goal of the learning method should be to find a classifier that directly leads to connected components that match the training community structure. The objective that is optimized by should reflect that. Instead of using edge-wise learning, we therefore propose a more principled approach, based on paths.

Two nodes a and b will be placed in the same community if they are in the same connected component in the reduced graph. That is the case if there is a path between them, where all edges in that path are retained. This can be denoted as

$$\text{connected}_h(a, b) = \exists(p = a, \ldots, b)\forall(cd \in E_p)h(\mathbf{x}_{cd}) > 0.$$

The quantity that should be minimized is then the number of pairs of nodes where connected_h is different from the community relation $\sim$. We call this the

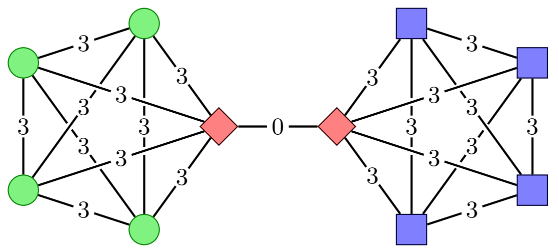

Figure 2: A graph with two cliques connected by a single edge. There are three communities (green circles, blue squares and red diamonds). When only the number of triangles t_{ab} are used as features, as indicated on the edges, then a classifier with no errors is impossible. The solution with the least path errors is to keep edges with 3 triangles and remove edges with 0 triangles. However, that solution is not optimal for any edge based loss, because only a within-community edge is removed.

path error,

$$\sum_{a,b\in V} \mathbf{1}[\text{connected}_h(a,b) \neq (a \sim b)]. \qquad (2)$$

We want to minimize the path error of the graph under consideration, but for that graph the community structure is of course unknown. Instead, we assume that the training data is representative of the community structure in the target graph. That means that the path error is approximated by the training path error, and we can minimize the latter instead.

Call the score $h(\mathbf{x}_{ab})$ assigned to an edge ab the *width* of that edge. The width of a path is then the minimum of the widths of the edges along that path. The *width of the widest path* $W_h(a,b)$ from a to b is the maximum of the widths of all paths from a to b,

$$W_h(a,b) = \max_{p=a,\dots,b} \min_{cd\in E_p} h(\mathbf{x}_{cd}). \qquad (3)$$

With this definition, the connected$_h$ predicate can be written as

$$\text{connected}_h(a,b) = W_h(a,b) > 0.$$

3.3 Path hinge loss

Analogously to the hinge loss for classification [Boser et al., 1992], we can now define the regularized *L2 path hinge loss* to be

$$L(h) = \|\mathbf{w}\|_2^2 + \frac{C}{|V|^2} \sum_{a,b\in V} \max(0, 1 - y_{ab} W_h(a,b))^2. \qquad (4)$$

where $h(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ is a linear classifier. We minimize $L(h)$ as a surrogate for the path error (2), because, unlike the path error, it is differentiable.

Unfortunately, this objective function is not convex, because of the minimum taken in W_h. Directly optimizing L is likely to lead to a poor local minimum. We therefore use a slight generalization of widest path widths, where only paths up to a fixed length k are considered. Let $L_k(h)$ denote the loss with paths restricted to have length at most k.

When $k = 1$, only paths that consist of a single edge are considered. So the objective $L_1(h)$ is the same as the standard edge-wise hinge loss. On the other hand, when $k \geq |V|$, we get the original path-wise loss, since no paths can be longer than the total number of nodes without containing cycles.

The graph might contain nodes a and b that are not connected by a path of length at most k. The width of the widest path between them would be $-\infty$, and loss as defined above would be infinite. To avoid this problem, we take the sum in (4) only over those nodes that are connected by a path. We also adjust the normalization from $C/|V|^2$ to C/m where m is the number of pairs of nodes that are connected by a path.

We use an off-the-shelve optimization package [Schmidt, 2005] for minimizing $L_k(h)$. To avoid local minima, we first minimize $L_1(h)$, and use the optimal h as a starting point for $L_2(h)$. In this way we gradually increase k until $k = |V|$.

3.4 Avoiding singletons

In extreme cases, all edges incident to a node could be classified as between community edges. In that case, the connected component containing that node would be a singleton. However, a single node alone is not a community. We found that we can significantly improve the quality of the clustering if we avoid such singletons.

To enforce that all communities consist of at least two nodes, we ensure that at least one incident edge per node is kept. In particular, for each node a, we keep the edge with the highest classifier score $h(\mathbf{x}_{ab})$, regardless of whether this score is positive.

4 LFR benchmark as training data

To learn a classifier, we need training data. This data must consist of one or more graphs with a known community structures. For the classifier to perform well on a testing graph, the training graphs must be similar to the testing graph. For real world applications, such training data can be hard to come by. It might be possible to manually label the communities in a small part of the graph, and use that as training data. However, this is still labor intensive work.

On the other hand, several artificial datasets exist for the community detection problem. These benchmarks are specifically constructed to closely resemble real world graphs. Therefore, we can use these artificial graphs as training data for an edge classifier.

Among the most advanced models is the LFR benchmark by Lancichinetti, Fortunato, and Radicchi [2008]. In this benchmark, the size of each community is drawn from a power-law distribution; as is the degree of each node. It has previously been observed that real world graphs also have such a power-law degree distribution Clauset et al. [2009]. Therefore, we hope that by using LFR graphs for training data, we can train classifiers that also work well on real-world testing graphs.

The LFR model has several parameters. The most important is the mixing parameter μ, that controls the fraction of edges that are between communities. Essentially this is the amount of noise in the graph. If $\mu = 0$ all edges are within community edges, if $\mu = 1$ all edges are between nodes in different communities.

Other parameters control the number of nodes, the distributions of community sizes, the distribution of degrees, etc. If something is known about the target graph, then these parameters should be chosen to match that graph. However, in this paper we do not try to match any particular graph. We therefore follow the settings used by Lancichinetti and Fortunato [2009]. They describe four benchmarks. Two with 'small communities' of between 10 and 50 nodes, and two with 'large communities' of between 20 and 100 nodes. Each graph has either 1000 or 5000 nodes in total.

Note that the older benchmark by Girvan and Newman [2002] can also be considered as an instance of the LFR benchmark. One obtains this model by fixing the community sizes to exactly 32 nodes, and the degree of each node to exactly 16.

5 Experiments

5.1 Artificial data

We first tested the performance of classifier based community detection on LFR benchmark graphs. We generated graphs using the settings described by Lancichinetti and Fortunato [2009], as well as for the Girvan and Newman [2002] benchmark[1]. This gives 5 classes of graphs, which we refer to as 'SMALL1000', 'SMALL5000', 'BIG1000', 'BIG5000' and 'GN'.

For each testing graph we used another graph generated with the same settings as training data. That means that the distribution of training and test graphs is the same. In the next section we will investigate how well the method generalizes when the

[1]We used the implementation from `http://sites.google.com/site/santofortunato`.

training data does not come from exactly the same model.

Since the true community structure is known for the benchmark graphs, we can calculate the similarity between any given clustering and the true one. To compare community structures, we used the normalized mutual information metric [Danon, Duch, Arenas, and Díaz-Guilera, 2005],

$$\mathrm{NMI}(\sim_1, \sim_2) = \frac{2I(\sim_1, \sim_2)}{H(\sim_1) + H(\sim_2)}.$$

This is a number between 0 and 1, that tells how much information one community structure gives about another one.

We tested two learning methods:

- an edge-wise weighted SVM, implemented LIBLINEAR [Fan, Chang, Hsieh, Wang, and Lin, 2008].
- the path hinge loss method described in section 3.3.

To select the hyper parameters of the learning methods, we used a grid search. For the path hinge method the only hyper parameter is C, which controls the regularization. For the edge-wise weighted SVM the hyper parameters are the regularization and the relative weight of within community edges.

Figure 3 shows the normalized mutual information as a function of the mixing parameter. The results for the two learning methods are virtually identical.

5.2 Generalization

Figure 4 shows the normalized mutual information for a classifier that was trained on a big community graph with 1000 nodes, and $\mu = 0.5$. The same classifier was used for all tests. As can be seen by comparing it to figure 3, the performance is similar to that obtained with classifier trained for the specific parameter settings. So, this classifier generalizes well to other parameter settings of the LFR benchmark.

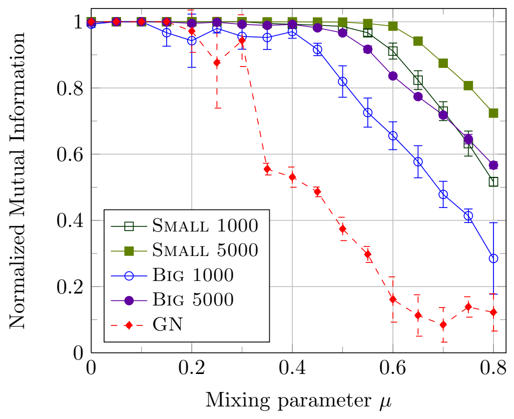

Figure 3: The performance of classifier based community detection on graphs generated with the LFR benchmark, as measured with Normalized Mutual Information. The error bars show the standard deviation across different training and testing datasets.

We also observed that classifiers trained on small communities benchmark graphs also work well for graphs with large communities, and vice-versa. Additionally, classifiers trained with a particular mixing parameter μ will still perform well on graphs using *less* mixing. The converse is not true, however, since graphs with low μ are essentially 'too easy'. They do not have any examples of between community edges, with larger number of triangles, which do appear with higher μ.

The reason for this generalization is that the 'good' linear classifiers that are found by the training algorithm are very similar. This suggests that for a particular set of features there might be a large class of graphs for which the same scoring or classification function is optimal. If the LFR benchmark is representative of real world data, then the classifiers we learned for artificial graphs, should also generalize to these real world graphs.

5.3 Real world data

In the previous section we have shown that a classifier trained on the BIG1000 dataset with $\mu = 0.5$

Table 1: Results on real world datasets.

	Normalized Mutual Information				Number of communities	
Dataset	Classifier	R. Weak	R. Strong	Infomap	Actual	Classifier
Zachary	0.649	0	0	0.568	2	4
Football	0.923	0.908	0.201	0.924	12	15
PolBooks	0.522	0	0	0.537	3	9
PolBlogs	0.134	0.014	0.014	0.340	2	322

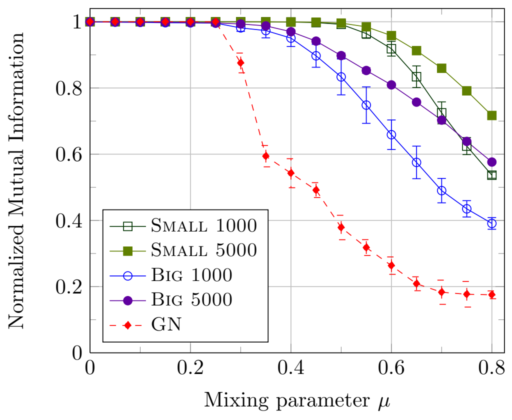

Figure 4: The performance of classifier based community detection on test graphs generated with the LFR benchmark. The same classifier is used in all cases, trained on a big community graph (Big1000) with $\mu = 0.5$

generalizes well to other LFR benchmark graphs. In this section we examine the applicability of this classifier to real world dataset. We consider the following datasets:

- Zachary's karate club [Zachary, 1977].
- Football: A network of American football games [Girvan and Newman, 2002].
- Political books: A network of books about US politics [Krebs, 2004].
- Political blogs: Hyperlinks between weblogs on US politics [Adamic and Glance, 2005].

We compare the results against two other community detection methods. First of all the method of Radicchi et al. [2004], because that method is similar to the classifier based method. It also uses the number of triangles and node degrees as features, and it too makes local decisions for each edge. They describe two variants of their stopping criterion, one based on 'weak communities', the other on 'strong communities'. We included both in our experiments, as R. Weak and R. Strong respectively.

Secondly, we compare with infomap by Rosvall and Bergstrom [2008]. This method represents the current state of the art in community detection [Lancichinetti and Fortunato, 2009].

Table 1 shows the results of the considered methods on the real world graphs. For the Zachary, Football and Political books networks, the results of the classifier are close to those of infomap. For the Political Blogs network the results are much worse. The classifier based method finds 322 communities, while the actual graph only has 2. The communities Figure 5

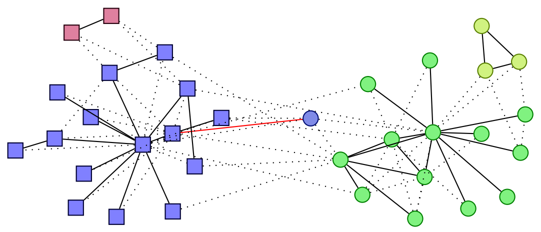

Figure 5: Zachary's karate club. The ground truth labeling has 2 communities, indicated as circles and squares. Classifier based community detection finds 4 communities, indicated with different colors. The classifier keeps one edge between communities (indicated in red), and removes many edges within the communities.

illustrates that even on small networks, the method has a tendency to remove too many edges, which results in too many clusters. For a small network, such as Zachary's splitting off some nodes leaves the cores of the communities intact, but for the political blogs network that does not appear to be the case.

6 Conclusion

We have shown how community detection can be approached as a classification problem: discriminate between edges to keep (within communities) and to remove (between communities). The idea is to apply a classifier to the edges of the graph, and then find the connected components of the subgraph obtained by deleting edges classified as removable.

In theory, a classifier that is trained to accurately distinguish within community edges from between community edges can not always find an optimal community structure of a graph. Therefore, we proposed an alternative loss that generalizes the hinge loss from edges to paths.

After training, the method can be efficiently applied to other graphs. This is because the decision of whether to keep an edge can be made locally, without knowledge of the other edges in the graph.

We showed that a simple linear classifier acting on few local topological features and trained on a single type of LFR graphs is capable to generalize well (that is, to identify the community structure) when applied to LFR graphs generated using different model parameter settings. This result indicates that one can easily learn a method for community detection for LFR graphs. Application of the resulting method (classifier) to real world networks shows that the classifier built on LFR graph is working well on some cases and bad on other cases. We show that a unsatisfactory result obtained on the PolBlogs graph is due to the type of classifier, since a direct training of the method on the graph itself does not improve the performance of the method.

An inherent limitation of the proposed method is the fact that it removes edges in one go, while existing algorithms for community detection based on edge removal incorporate an adaptive mechanism where the score of the edges is updated each time a (set of) edges is removed. How to incorporate such a mechanism in a supervised setting without making the training process too complex is an open issue.

We have focused on linear classifiers, because of their simplicity, and because of the similarity to previously used scoring functions. In general, when enough training data is available, non-linear classifiers such as kernel methods can outperform linear ones [Cristianini and Shawe-Taylor, 2000]. It remains to be seen whether that also holds for the purpose of community detection by classifying edges. However, preliminary experiments have shown that non-linear classifiers do not generalize as well.

For our experiments we have only used the node degrees, number of triangles, and mean values of these as features. It might be possible to improve the results by including additional features. In particular, we have noticed that not including the mean number of triangles and mean degrees of nodes reduces the performance. It would therefor be interesting to investigate whether including more summary statistics, for example quantiles of the number of triangles, can improve the results.

In this paper we have employed a simple post processing step to eliminate singleton clusters. However, this post processing is not taken into account by the training method. It might be possible to modify the loss function to also optimize the post processing step, for instance by locally selecting the number of edges to keep. This method of removing edges is closely related to the recent work of Satuluri et al. [2011] on graph sparsification, where the number of edges to keep is chosen per node. We have not yet investigated this idea further.

References

Lada Adamic and Natalie Glance. The political blogosphere and the 2004 u.s. election: Divided they blog. In *In LinkKDD 05: Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, 2005.

Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466:761–764, 2010.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51:661–703, November 2009. ISSN 0036-1445.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 1 edition, March 2000. ISBN 0521780195.

L. Danon, J. Duch, A. Arenas, and A. Díaz-Guilera. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 9008:09008, 2005.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008. ISSN 1532-4435.

S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, 2002.

V. Krebs. New political patterns, 2004. URL `http://www.orgnet.com/divided.html`.

A. Lancichinetti and S. Fortunato. Community detection algorithms: A comparative analysis. *Phys. Rev. E*, 80:056117, Nov 2009.

A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78(4), 2008.

S. Muff, F. Rao, and A. Caflisch. Local modularity measure for network clusterizations. *Phys. Rev. E*, 72:056107, 2005.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.

F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004.

Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences of the United States of America*, 105(4): 1118–1123, January 2008.

V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 721–732, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4.

M. Schmidt. minfunc, 2005. URL `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`.

W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.