Sparsification and Sampling of Networks for Collective Classification

Tanwistha Saha, Huzefa Rangwala, and Carlotta Domeniconi

Department of Computer Science
George Mason University
Fairfax, Virginia, USA
`tsaha@gmu.edu, {rangwala,carlotta}@cs.gmu.edu`

Abstract. Network analysis has been an active area of research for the past few decades. Out of many open research questions that have been extensively studied, relational classification, community detection, link prediction are only to name a few. Collective classification is a well-known relational classification method for classifying entities (nodes) within a network which involves using both node based features and topological features of each node. It involves *collective prediction* of the unknown labels of *all* the test nodes in the network using label information of the training nodes. Even though this has been a well researched topic for years, very little has been done to address the following two challenges: (1) how to actively select the labeled nodes from the network to be used for training, and (2) how to efficiently obtain a sparse representation of the original network without losing much information, so that learning can scale to large networks. A lot of work has been done in theoretical computer science which aims towards finding the best approximation of large graphs. However, not much has been done from the perspective of finding an approximate subgraph that will help in classification of network datasets. In this paper, our contribution is in proposing an efficient graph sparsification method and a sampling technique which, along with the state-of-the-art network classifiers, can give comparable runtime and classification accuracies.

1 Introduction

Networks are usually represented as graphs where the vertices are entities and the edges represent interaction between these entities. Of numerous aspects of computational social science that are in practice, our research is primarily focused onto learning supervised models that can be used for making accurate predictions on unseen data within the network. Although major work has been done on finding efficient learning models, an important challenge arises while trying to find the right samples (i.e, instances that represent the distribution from which the original dataset was generated), using which we can learn our models.

Learning in relational datasets is different from learning in non-relational datasets, because of the presence of links between entities in networks. These links account for pairwise interactions between entities and hence, can be considered for extracting additional features about the network dataset. *Collective classification* considers link-based features along with the node specific features, while training a model. It deals with the simultaneous classification of neighboring instances in relational data, until a convergence criterion is reached. The rationale behind collective classification stems from the fact that an entity in a network (or relational data) is most likely influenced by the neighboring entities, and can be classified accordingly, based on the class assignment of the neighbors. Collective classification approaches jointly classify a set of related (linked) nodes by exploiting the underlying correlations between the labels and the attributes of the training nodes and that of the test nodes in a network [4].

Our contribution in this paper is two fold: (i) We propose an algorithm to *sample* subgraphs from the full network (a.k.a. *network sampling* [2]). (ii) We propose an algorithm to sparsify the network by removing noisy edges and find a suitable approximation that can facilitate collective classification, without compromising the accuracy. We explain the notion of sampling in detail in section 3.3. We use two benchmark network datasets: (i) Cora (directed graph representing citation network of scholarly papers)[1] and (ii) DBLP (undirected graph representing co-authorship network of computer science researchers)[2] to show that our sampling technique, along with a state-of-the-art collective classification method [8], can improve the classification accuracy. In addition, our graph sparsification method helps in reducing the number of edges in the network and reduces the execution time without compromising the classification accuracy.

2 Related Work

Given the aim of collective classification earlier in this paper, an important question that comes to mind is - how to train a collective classification model? In traditional learning theory, samples (set of instances) chosen randomly (with or without replacement) from the entire set of data points are assumed to represent the key characteristics of the original dataset [3]. In these cases, randomly chosen instances for training is sufficient for learning models which does not make use of link structure in the dataset. However, in network analysis, researchers have shown that randomly chosen nodes are not good representative samples of the network [2]. Many competitive approaches, e.g., forest fire sampling [5], node sampling, edge sampling and edge sampling with graph induction [1] have been proposed so far. Most of the proposed methods have either taken a generalized approach towards sampling or have considered networks that evolve over time. Recently, Ahmed et al. [2] have shown that, forest fire sampling has an overall good performance for classification in relational data. The goal of any sampling method is to preserve key features (e.g., sufficient statistics) of the underlying distribution. Which feature is a *key* depends on the problem we are trying to solve. Hence, instead of developing another generalized sampling algorithm for

[1] http://www.cs.umd.edu/∼sen/lbc-proj/LBC.html
[2] http://www.informatik.uni-trier.de/∼ley/db/

any learning method (supervised, semi-supervised or unsupervised), we propose an adaptive version of forest fire sampling in order to facilitate collective classification in network datasets. This is different from the *active sampling* method proposed recently [11], where the goal is to draw samples from the network that have high probability of having a specific label. Our method can be categorized as *active labeling*, where both the value of the label as well as the local structural information of the labeled instance are acquired.

Our second contribution is a sparsification method for removing noisy edges in the network. Attribute information and labels of neighboring nodes have been useful for improving accuracy in collective classification problems. However, if the collective classification algorithm fails to classify a node correctly, then the misclassification error is propagated throughout the network. Certain edges in a network do not carry much information and do not contribute towards better accuracy for classification models. These edges, or *weak ties*, between entities can be removed from the network to speed up the learning process without compromising much with the classification accuracy. There are many good sparsification algorithms [15, 16] which aim to find best approximation for large graphs in linear time. Those algorithms are focused on reducing the time complexity for finding the best approximation of large graphs. Our method is different from those approaches; our objective is to find a sparse network that will help in reducing the runtime of collective classification algorithms. Our graph sparsification approach is inspired by a local graph sparsification algorithm [13]. However, we emphasize that this approach was meant for efficient clustering in graphs (or networks), whereas, our approach is to facilitate classification in graphs.

3 Methodology

Let graph $G = (V, E)$ represent a relational dataset $D = (\mathcal{X}, \mathcal{Y})$, where $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ is the set of node attributes and $\mathcal{Y} = \{y_1, y_2, \cdots, y_n\}$ is the set of corresponding labels for the $|V| = n$ nodes in the network. Let K be the number of classes, denoted by $k = 1, 2, \cdots, K$, to which the nodes can belong ($y_i = k$, where $i = 1, 2, \cdots, n$). Let $M_{|E|\times 2}$ be an edge-vertex incidence matrix where row denotes an edge $e \in E$. The column of M indicates the two vertices with which the edge e is associated. Let $A_{n\times n}$ be the weighted adjacency matrix representing the interactions between any pair of nodes in the network.

3.1 Problem Definition

Given a sparsification ratio θ ($0 < \theta < 1$), our goal is to:

- Find a sparsified network $G_s = (V, E_s)$ (where $E_s \subset E$) using a sparsification algorithm *Adaptive Global Sparsifier* (Algorithm 1) where $|E_s| = \lceil \theta \times |E| \rceil$.
- Find a sample subgraph $G'_s = (V'_s, E'_s)$ from the network $G = (V, E)$ (where $V'_s \subset V$ and $E'_s \subset E$) using a sampling algorithm *Adaptive Forest Fire Sampling* (Algorithm 2), in order to train a classifier.

Algorithm 1 Adaptive Global Sparsifier (AGS)

```
Input: A graph $G = (V, E)$, sparsification ratio $\theta$, edge-vertex incidence matrix $M_{|E|\times 2}$
Output: A sparsified graph $G_s = (V, E_s)$ where $E_s \subset E$
 1: $E_s \leftarrow \emptyset$
 2: $c_G = components(G)$
 3: for each edge $e \in E$ do
 4:   $(i, j) \leftarrow M_e$
 5:   Compute $e.sim = Sim(i, j)$ as in equation (1)
 6: end for
 7: Sort and order the edges $e \in E$ of the graph according to decreasing order of $e.sim$
 8: Add top $\lceil \theta \times |E| \rceil$ edges of $G$ to the sparse graph $G_s$ i.e. $|E_s| = \lceil \theta \times |E| \rceil$
 9: for each edge $e_p \in E - E_s$ such that $e_p$ is the edge with the lowest similarity score ($e.sim$) in
    the sorted edge list do
10:   $E \leftarrow E - e_p$
11:   $c = components(G)$
12:   if $c > c_G$ then
13:     $E_s \leftarrow E_s \cup e_p$
14:   end if
15: end for
```

3.2 Adaptive Global Sparsifier

Given the node attributes for all the nodes in the graph G, we compute pairwise cosine similarity between any pair of nodes i and j, as follows,

$$Sim(i, j) = \frac{\mathbf{x}_i{}^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2} \tag{1}$$

We sort the edges according to descreasing order of similarity and keep the top $\lceil \theta \times |E| \rceil$ edges in the graph. For the rest of the edges, we start scanning from the bottom of the sorted list of edges. We remove an edge if this doesn't increase the number of connected components in the graph (function *components*(G_s) in Algorithm 1, checks the number of connected components in the graph G_s), otherwise we put back that edge. A graph becomes more disjoint if the number of connected components increases. This situation is not encouraged in collective classification setting, since collective classification relies on the connectivity of the graph for propagating the labels from training nodes to test nodes. Our sparsification algorithm AGS aims to remove noisy edges from the graph such that removal of such an edge *does not increase* the number of connected components in the sparsified graph, as compared to the original graph. Checking for connected components within a graph has a time complexity of $O(|V| + |E|)$. Hence, the time complexity of Algorithm 1 is $O(|E| \times (|V| + |E|))$, since the outer *for-loop* (lines $9 - 15$) can run at most $|E|$ times. Our AGS algorithm is different from the global and local sparsification algorithms proposed by Satuluri et al. [13]. The global sparsification algorithm [13] adds only the top $\lceil \theta \times |E| \rceil$ edges in the graph based on the Jaccard similarity measures of the edges. The local sparsification algorithm [13] scans through each node in the graph, sorts the edges incident on the node according to the Jaccard similarity measure and keeps only d^e edges, where d is the degree of the node under consideration, and e is any predefined value between 0 and 1. For ease of reference, we call these two methods *Global Sparsifier (GS)* and *Local Sparsifier (LS)*, respectively. We consider these two sparsification algorithms as baseline methods for comparison.

3.3 Adaptive Forest Fire Sampling

Forest fire sampling as proposed by Leskovec et al. [6, 5], starts by choosing a random seed node and burning a fraction of its outgoing links to neighboring nodes attached to it. This fraction is chosen from a geometric distribution with a predefined mean. It is similar to a random walk based approach, where the selection process is recursive and it halts when no new nodes are selected, or when the required sample size has been reached. Since collective classification propagates the label information from the training nodes to the test nodes in a network during the inference process, it would be helpful if a test node is connected to at least one training node in order to make use of the training node's characteristics for the test node during the training phase of the classifier. Our sampling algorithm is an adaptive version of forest fire sampling and is described in detail in Algorithm 2.

Algorithm 2 Adaptive Forest Fire Sampling

```
Input: A graph $G = (V, E)$, edge-vertex incidence matrix $M_{|E|\times 2}$, training ratio $\delta$
Output: A sampled subgraph $G'_s = (V'_s, E'_s)$
 1: $V'_s \leftarrow \emptyset$, $E'_s \leftarrow \emptyset$, $E_p \leftarrow \emptyset$
 2: Sample size $s = \lceil |V| \times \delta \rceil$
 3: Randomly select a seed node $p$ from the graph $G$
 4: repeat
 5:   $V'_s \leftarrow V'_s \cup p$
 6:   $d_p \leftarrow degree(p)$ /* $degree(p)$ returns degree of node $p$ */
 7:   Select $\lceil d_p \times \delta \rceil$ edges incident on node $p$ and add to $E_p$
 8:   for each edge $e \in E_p$ do
 9:     $(p, v) \leftarrow M_e$
10:     $E'_s \leftarrow E'_s \cup e$
11:     if $v \notin V'_s$ then
12:       $V'_s \leftarrow V'_s \cup v$
13:       Add node $v$ in the end of the queue $Q$
14:     end if
15:   end for
16:   if Queue $Q$ is not empty then
17:     $p \leftarrow$ first node at the head of the queue $Q$
18:   else
19:     Randomly select a seed node $p$ from the graph $G$ such that $p \notin V'_s$
20:   end if
21:   $E_p \leftarrow \emptyset$
22: until $|V'_s| = s$
```

The complexity of the adaptive forest fire sampling algorithm mostly depends on the time for adding/removing elements (edges or nodes) from the queue Q which is run by the inner *for-loop* (lines $8 - 15$) in Algorithm 2 with $O(\lceil d_p \times \delta \rceil)$ time. Hence, the time complexity is a function of the degree of the seed nodes chosen randomly by the algorithm. Algorithm 2 aims to find at least one labeled node for each test node in the graph and also ensures that the subgraph is connected. To achieve this goal, the algorithm performs a check at each step by looking forward from the current node and selecting a certain percentage (equal to the percentage of training samples) of edges to be included in the sampled subgraph. It considers the nodes connected to those selected edges as *training nodes* for the sampled subgraph. This process continues until the required training sample size (of nodes) has been reached. We refer to this approach as *adaptive forest fire*, because at each step the algorithm *adapts* towards selecting the edges incident on that particular node residing at the head of the queue that keeps track of the processed nodes. In case there are multiple connected components in the graph, then this algorithm is repeated for each connected component, until the overall sample size for the entire graph has been reached. Selecting training nodes by traversing certain percentage of edges only, ensures that the nodes (that end up being test nodes) at the other end of the unselected edges are connected to at least one labeled node. However, readers should note that, by "sampling" from a graph we mean to extract a subset of nodes whose labels would be made available to the learning algorithm during the training phase such that it helps collective classification. To prove whether this sampled subgraph retains the essential properties of the original graph, is a part of future work.

3.4 Collective Classification

Collective Classification is a combinatorial optimization problem where we are given the set of nodes $V = \{v_1, v_2, v_3, \cdots, v_n\}$ over a graph $G = (V, E, \mathcal{X}, \mathcal{Y}, K)$, such that, E is the set of edges, each $\mathbf{x}_i \in \mathcal{X}$ is an attribute vector for node $v_i \in V$, each $y_i \in \mathcal{Y}$ is a label variable for node v_i, and K is the set of possible labels [14]. We are also given a set of known values Y_l for nodes $V_l \in V$ (the nodes for which we know the correct labels) such that $Y_l = \{y_i | v_i \in V_l\}$. The task is to infer Y_u (i.e., the value of y_i for the each node $v_i \in V - V_l$) for which we do not know the labels. Many sophisticated collective classification methods have been proposed [10, 7, 9]. Here we use the state-of-the-art weighted vote relational neighbor (wvRN) classifier [8] because it is simple, fast, and yet an efficient method. We also use another simple relational classifier RankNN [12](a multi-label classification algorithm, modified to support single label classification), and a multi-class Support Vector Machine classifier [17] to compare our methods against the baseline.

3.5 Connected Components

A graph $G = (V, E)$ is said to be connected if there is a path between any pair of vertices (u, v), such that $u \in V$ and $v \in V$. A *connected component*, $G_c = (V_c, E_c)$, is a *connected* subgraph of a graph G. A graph $G = (V, E)$ is said to have multiple connected components $G_1, G_2, \cdots, G_n$ (where $G_1 = (V_1, E_1), G_2 = (V_2, G_2), \cdots, G_n = (V_n, E_n)$, $V = V_1 \cup V_2 \cup V_3 \cdots \cup V_n$, and $E = E_1 \cup E_2 \cup E_3 \cdots \cup E_n$), if, for any pair of such components (say $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$), there exists a pair of nodes (u, v) such that $u \in V_i$ and $v \in V_j$ and there is no path from u to v, or vice versa.

A graph becomes more disjoint if the number of connected components increases. This situation is not encouraged in collective classification setting since collective classification relies on the connectivity of the graph for propagating the labels from training nodes to test nodes. Our sparsification algorithm *AGS*

Table 1. Description of datasets.

Dataset	n	K	$\|E\|$	$\|E_s\|$ with (AGS)	Time (sec) (AGS)	$\|E_s\|$ with (GS)	Time (sec) (GS)	$\|E_s\|$ with (LS)	Time (sec) (LS)
Cora	2708	7	5429	3850	5.4859	3800	2.5695	2429	2.3997
DBLP	5602	6	17265	12251	53.8278	12086	28.4722	6859	19.6182

aims to remove noisy edges from the graph such that removal of such an edge *does not increase* the number of connected components in the sparsified graph, as compared to the original graph.

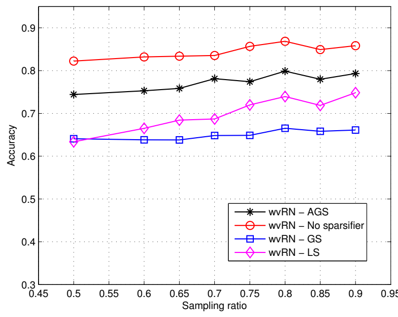
4 Experimental Results

We processed two real-world datasets, Cora and DBLP, for testing our methods. The statistics of the datasets are given in Table 1, along with the time taken by different sparsifiers to obtain a sparsified graph from those networks (sparsification ratio $\delta = 0.7$). Cora is a directed citation network where each node represents a paper and each edge represents a citation link from one paper to another. Each paper can belong to one of seven research areas: Case-Based Reasoning, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning and Theory. The DBLP network is an undirected co-authorship network between authors publishing research papers in computer science, which can belong to one of six research areas: Theory, NLP, Bioinformatics, Operating System, Distributed System and Networking. We have also reported the time taken by our proposed algorithm Adaptive Global Sparsifier (AGS) and by the baseline algorithms Global Sparsifier (GS)/Local Sparsifier (LS) in order to sparsify the original network datasets.

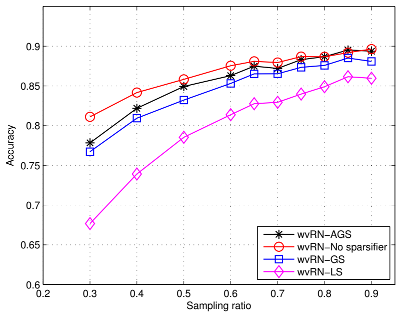
Figures 1(a) and 1(b) show the performance of our sparsification algorithm (AGS) with respect to the baseline methods (GS and LS) [13] on Cora and DBLP datasets respectively, using the wvRN classifier with iterative inference method and random sampling. On non-sparsified networks, the wvRN classifier performs best for both datasets (which is expected, since, due to sparsification we are losing some information from the network resulting into a loss of accuracy). But when networks are sparsified, our algorithm performs best compared to others (note that, GS and LS methods encourage removal of those edges that increase the number of connected components in the graph). Also, for DBLP, the classifier's accuracy is almost similar to that obtained using a non-sparsified graph.

Figures 1(c) and 1(d) show the performance of different classifiers with random node sampling and adaptive forest fire sampling on the Cora dataset. On non-sparsified networks, the wvRN classifier performs best with random sampling. Immediately following this, is the performance of the wvRN classifier used on networks sparsified by our adaptive global sparsification (AGS) algorithm. However, when adaptive forest fire sampling is used, all the three classifiers using

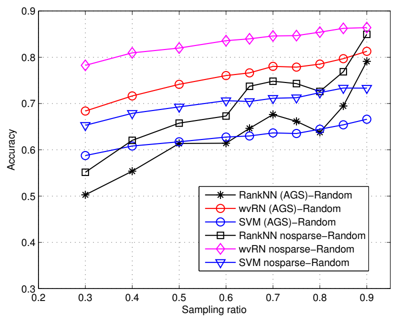
Fig. 1. Collective Classification performance using graphs sparsified by AGS, GS and LS.



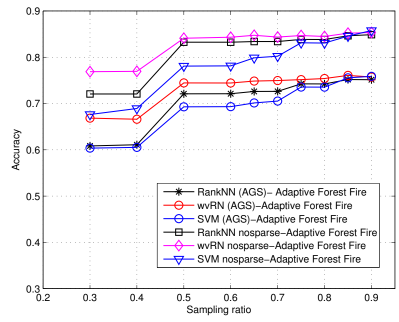
(a) Cora



(b) DBLP



(c) Cora



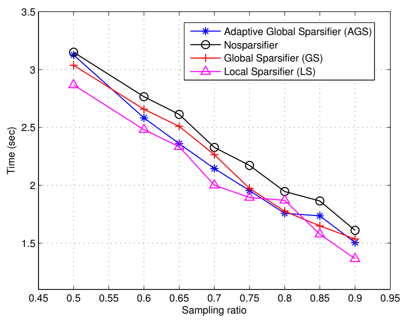
(d) Cora

non-sparsified networks, perform better in comparison with the classifiers using sparsified networks (which is natural, since sparsification of networks causes loss of information).

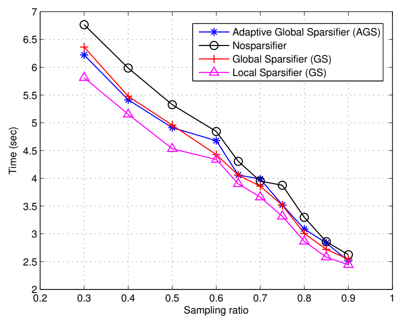
Figures 2(a) and 2(b) show the run time for the wvRN classifier using different graph sparsification algorithms on Cora and DBLP datasets respectively, while varying the training sample sizes. As expected, the wvRN classifier with sparsified networks, take less time for execution in both the datasets.

Figures 2(c) and 2(d) show the run time of the wvRN classifier with networks sparsified using our adaptive global sparsification algorithm (AGS) for different sparsification ratios. Time consumption is less for both the datasets (with different sampling algorithms) when AGS algorithm is used to sparsify the original input networks. We also observe that, the variation of time consumption for the wvRN classifier using the sparsified graphs becomes almost flat when sparsification ratio $\theta > 0.6$.

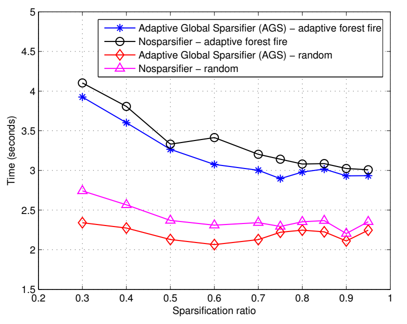
Fig. 2. Run Time of wvRN classifier using different sparsification methods.



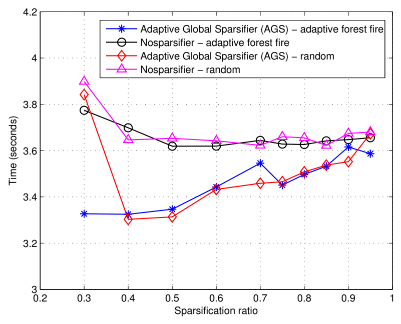
(a) Cora



(b) DBLP



(c) Cora



(d) DBLP

5 Conclusion

In this paper, we have proposed a useful and efficient technique to sparsify graphs (or networks of entities) that can help in improving the performance of a collective classification method, without compromising the accuracy. We also proposed a network sampling algorithm to extract samples from a network by actively labeling nodes in the original network. We have reported extensive experimental results on two real-world relational datasets, demonstrating that our proposed methods perform well, both with respect to accuracy and time complexity. However, these algorithms have been designed to treat single labeled network datasets only. In the future, we aim to design efficient graph sparsification algorithms for multi-labeled network datasets as well.

References

1. Ahmed, N., Berchmans, F., Neville, J., Kompella, R.: Time-based sampling of social network activity graphs. In: Proceedings of the Eighth Workshop on Mining and Learning with Graphs. pp. 1–9. ACM (2010)

2. Ahmed, N., Neville, J., Kompella, R.: Network sampling designs for relational classification. In: Sixth International AAAI Conference on Weblogs and Social Media (2012)
3. Friedman, J., Hastie, T., Tibshirani, R.: The elements of statistical learning, vol. 1. Springer Series in Statistics (2001)
4. Jensen, D., Neville, J., Gallagher, B.: Why collective inference improves relational classification. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 593–598. ACM (2004)
5. Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD. pp. 631–636. ACM (2006)
6. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceedings of the eleventh ACM SIGKDD. pp. 177–187. ACM (2005)
7. Lu, Q., Getoor, L.: Link-based classification. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). vol. 20, pp. 496–503 (2003)
8. Macskassy, S., Provost, F.: Classification in networked data: A toolkit and a univariate case study. The Journal of Machine Learning Research 8, 935–983 (2007)
9. McDowell, L., Gupta, K., Aha, D.: Cautious inference in collective classification. In: Proceedings of the National Conference on Artificial Iintelligence. vol. 22, p. 596. AAAI (2007)
10. Neville, J., Jensen, D.: Iterative classification in relational data. In: Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data. pp. 13–20 (2000)
11. Pfeiffer III, J., Neville, J., Bennett, P.: Active sampling of networks. In: 10th International Workshop on Mining and Learning with Graphs (2012)
12. Saha, T., Rangwala, H., Domeniconi, C.: Multi-label collective classification using adaptive neighborhoods. In: Machine Learning and Applications (ICMLA), 2012 11th International Conference on. vol. 1, pp. 427–432. IEEE (2012)
13. Satuluri, V., Parthasarathy, S., Ruan, Y.: Local graph sparsification for scalable clustering. In: Proceedings of the 2011 ACM SIGMOD. pp. 721–732. ACM (2011)
14. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI magazine 29(3), 93 (2008)
15. Spielman, D., Srivastava, N.: Graph sparsification by effective resistances. SIAM Journal on Computing 40(6), 1913–1926 (2011)
16. Spielman, D., Teng, S.: Spectral sparsification of graphs. arXiv preprint arXiv:0808.4134 (2008)
17. Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: Proceedings of the 21st international conference on Machine learning. p. 104. ACM (2004)