Lightweight Label Propagation for Large-Scale Network Data

Yu-Feng Li, *Member, IEEE* and De-Ming Liang

Abstract—Label propagation spreads the soft labels from few labeled data to a large amount of unlabeled data according to the intrinsic graph structure. Nonetheless, most label propagation solutions work under relatively small-scale data and fail to cope with many real applications, such as social network analysis, where graphs usually have millions of nodes. In this paper, we propose a novel algorithm named SLP to deal with large-scale data. A lightweight iterative process derived from the well-known stochastic gradient descent strategy is used to accelerate the solving process. We also give a theoretical analysis on the necessity of the warm-start technique for label propagation. Experiments show that our algorithm is several times faster than state-of-the-art methods while achieving highly competitive performance.

Index Terms—Label Propagation, Scalability, Stochastic, Network Data

1 Introduction

With the vigorous development of the Internet, many enterprises, such as Twitter, Facebook, YouTube etc., are accumulating massive data in daily operating. Information which can boost revenue are buried under raw data and need to dig out. Among them, the social network is one of worthiest data to mine and classification on a network would provide useful information on the nodes which represent users. Besides, many other areas are generating large networks. For examples, web-based encyclopedia platform like Wikipedia has a complex and large hyperlinks graphs for different articles, computer science bibliography website DBLP has large collaboration network and so on. The connections in network represent the relationship between nodes and can help find patterns or other knowledge in the network. We visualize two social network using Gephi software in Fig.1 and Fig.2. We can see that certain network truly contains some patterns. In general, the networks in real-world tasks have at least millions of nodes, which make high requirements for the scalability and efficiency of the algorithm.

Label propagation is a practical and popular algorithm for network analysis. Label propagation algorithms make prediction for nodes given a graph with a small portion of nodes with initial labels. It was first formulated as a graph cut problem [1], [2], [3], [4], in which the positive nodes and negative nodes are departed by removing some edges. Other methods [5], [6] formulated the problem as a regularized function estimation over the graph. They extend discrete predictive function to continuous one and makes the min-cut solvable. These methods optimize the trade-off between fitting a label prediction function on the labeled nodes and a regularization term which encourages smoothness of such a function over the graph. However, they suffer from the time complexity to solve the linear system and hard to run in real-world tasks with large-scale graphs.

- *Y.-F. Li and D.-M. Liang are with National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. De-Ming Liang contributes equally to this work.*
  E-mail: {liyf, liangdm}@lamda.nju.edu.cn;

Various algorithms have been proposed to mitigate the scalability problem. Most of them mainly focus on graph approximation to improve the efficiency and can be divided into two sorts: low-rank approximation and sparse approximation.

The low-rank approximation is implemented in several ways. [7], [8], [9], [10], [11] select a subset of the nodes as the prototypes and use them in the following predicting procedure. Empirically, the number of prototypes m should be proportional to the training set size n, and they are usually selected from the clustering centers, which makes the prototypes expensive and inefficient. An alternative approach is based on the fast spectral decomposition of Laplacian matrix [12], [13]. They try to find a representation of the structure in a low-dimension space formed by the eigenvectors of the Laplacian matrix with smallest eigenvalues and then train a classifier on this representation. However, the derivation relies on the assumption of dimension independence, which are not true for real-world data.

Graph sparse approximation mainly utilizes the minimum spanning tree (MST). These approaches first construct a minimum spanning tree on the given graph, and then solve the label propagation with a designed tree Laplacian solver [14], [15], [16], [17]. These approaches seem great in complexity analysis, but they need to solve each connected component respectively, and the constructing process needs meticulous coding work.

TABLE 1: Summary of state-of-the-art graph-based methods. LP? means whether to solve label propagation, GC? means whether to construct approximate graph. Approximation type and time complexity are also reported. n is the number of nodes, E is the set of edges, m is the number of prototype points or eigenvectors and d is the dimension of features.

Method	LP?	GC?	Approximation	Time Complexity
Anchor [10]	Yes	Yes	Low-rank	$O(dmn + m^2n)$
Eigen [12]	Yes	No	Low-rank	$O(dmn + m^3)$
TbTL [17]	Yes	No	Sparse	$O(\|E\| \log n)$
SLP	Yes	No	Stochastic	$O(\|E\|)$

We try to ease the burden of large-scale label propagation given the network in a way distinct from the approximation work mentioned above. We solve LP (label propagation) as a semi-

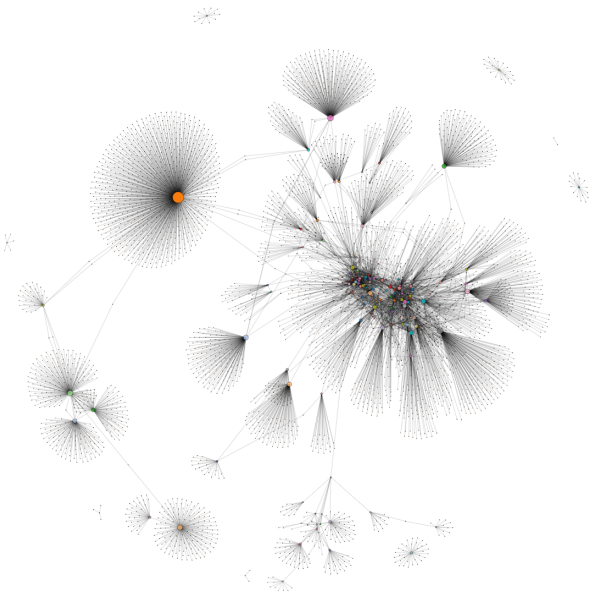

Fig. 1: Com-LiveJournal Network

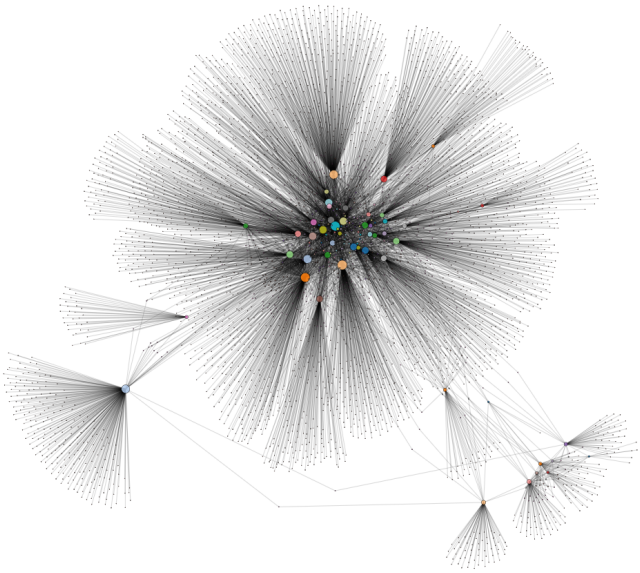

Fig. 2: Com-Orkut Network

supervised learning problem, which is equivalent to minimize a loss function from the graph and derive a lightweight iterative process from the stochastic gradient descent (SGD) algorithm. SGD has been successfully applied to large-scale machine learning problems [18], [19]. Our proposal is simple, fast and scalable, and we name the algorithm SLP (Stochastic Label Propagation). Experiments show that our algorithm can handle million-scale graphs in few seconds while achieving highly competitive performance with existing algorithms.

Table 1 compares state-of-the-art efficient graph-based methods with SLP. Our method focus on solving the scalability issue of label propagation. We assume that a graph is provided in advance. It is worth mentioning that distributed solution for label propagation have been implemented with the wide spread of distributed computing frameworks like Hadoop [20] or Apache Spark. However, our work is still valuable since we improve the unit efficiency in single machine. Our method can be easily extended to distributed version with the help of existing distributed stochastic gradient descent algorithms [21], [22].

In the following, Section 2 introduces the formulation of label propagation. Our proposed method is presented in Section 3, followed by the analyses in Section 4. We report the experimental results in Section 5 and give conclusive remark finally.

TABLE 2: Summary of Notation

Notation	Meaning
n	Number of instances
l	Number of labeled instances, $l \ll n$
$\boldsymbol{x}_i \in \mathbb{R}^d$	Feature vector of instance
$y_i \in \{+1, -1\}$	Label of instance
$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{l}$	Labeled instances
$\{\boldsymbol{x_i}\}_{i=l+1}^{n}$	Unlabeled instances
$\mathbf{y_l} = [y_1, y_2, ..., y_l]$	Label vector for given labeled instances
$f : \mathbb{R}^d \to [-1, 1]$	Predictive function
$\mathbf{f_l} = (f_1, f_2, ..., f_l)$	f's prediction on labeled instances
$\mathbf{f_u} = (f_{l+1}, ..., f_n)$	f's prediction on unlabeled instances
$\mathbf{f} = [\mathbf{f_l}, \mathbf{f_u}]$	Predictive vector
$\mathbf{F} \in \mathbb{R}^{n \times c}$	Continuous soft label matrix
diag$(\cdot)$	Diagonal matrix with diagonal vector $\cdot$
$\mathbf{1}$	Vector with all elements set to 1
$\mathbf{e_i}$	Vector with the i-th element set to 1 and others are set to 0
$\mathbf{I}$	Identity matrix
$\mathbb{I}(\cdot)$	$\mathbb{I}(\cdot) = 1$ if $\cdot$ is true, otherwise $\mathbb{I}(\cdot) = 0$
$\mathbf{W} \in \mathbb{R}^{n \times n}$	Affinity matrix of instances
$\mathbf{W}_i$	the i-th row of $\mathbf{W}$
$\mathbf{D} \in \mathbb{R}^{n \times n}$	$\mathbf{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} W_{ij}$
$\mathbf{L} \in \mathbb{R}^{n \times n}$	Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$
$\mathbf{A} \circ \mathbf{B}$	Hadamard product of matrix $\mathbf{A}$ and $\mathbf{B}$: $(\mathbf{A} \circ \mathbf{B})_{ij} = \mathbf{A}_{ij}\mathbf{B}_{ij}$

2 Problem Formulation and Notation

We first give the necessary notation of label propagation. The problem is defined on a set of instances $\mathcal{X}$, i.e., labeled instances $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_l, y_l)\}$ and unlabeled instances $\{\boldsymbol{x}_{l+1}, \boldsymbol{x}_{l+2}, ..., \boldsymbol{x}_n\}$, where $l \ll n$. To propagate labels, a graph $G = (V, E)$ with affinity matrix $\mathbf{W}$ is constructed. V is exactly the set of the feature vectors of nodes, and E is a set of edges between node pairs. Here the affinity matrix can refer to any information coming with the structure of a graph, including commonly used similarity measures, connections and so on. The weights between any node pair (i, j) are stored in the affinity matrix $\mathbf{W}$. W_{ij} can reflect the similarity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The weight matrix can be computed with Gaussian kernel function

$$W_{ij} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\sigma^2}\right),$$

where σ is the hyper-parameter [5]. The vertex degree matrix $\mathbf{D}$ is a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} W_{ij}$. We can also use 0/1 as the value of W_{ij} to represent the existence of edges or not.

The graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{W}$. There are other forms of Laplacian such as symmetric normalized version $\mathbf{L}^{\text{sym}} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ and random walk normalized version $\mathbf{L}^{\text{rw}} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$.

The goal of LP is to find a labeling which is consistent with the initial labeling and the pairwise similarity of the nodes. The former is commonly considered in supervised learning paradigm, which can be measured by loss on the labeled data. The latter one follows *smoothness assumption* [23] and is usually represented with a smoothness term. Classic methods like Gaussian fields and harmonic functions (GFHF) method [5] and Local and global consistency (LGC) method [6] involve the goals above. Both of them define a loss function $\mathcal{L}$ which combines the contributions of the loss on labeled data $\mathcal{L}_{\text{label}}$ and the global smoothness $\mathcal{L}_{\text{smooth}}$. The predictive label matrix F^* is obtained by minimizing the loss function $\mathcal{L}$,

$$\begin{aligned}\mathbf{F}^* &= \arg\min_{\mathbf{F}\in\mathbb{R}^{n\times c}} \mathcal{L}(\mathbf{F}) \\ &= \arg\min_{\mathbf{F}\in\mathbb{R}^{n\times c}} \left(\mathcal{L}_{\text{label}}(\mathbf{F}) + \mathcal{L}_{\text{smooth}}(\mathbf{F})\right).\end{aligned}$$

Loss function Eq.(1) is proposed by LGC [6],

$$\mathcal{L}_{\text{LGC}}(\mathbf{F}) = \frac{1}{2}\text{tr}(\mathbf{F}^\top \mathbf{L}^{\text{sym}}\mathbf{F}) + \frac{1}{2}\mu\|\mathbf{F}-\mathbf{Y}\|^2, \tag{1}$$

The first term $\text{tr}(\mathbf{F}^\top \mathbf{L}^{\text{sym}}\mathbf{F})$ measures label smoothness over the whole graph, and the second term $\|\mathbf{F}-\mathbf{Y}\|^2$ is the loss on labeled data, where $\mathbf{Y} \in \{0,1\}^{n\times c}$ is the ground truth label matrix. Note that when we set $\mu = \infty$ and replace the symmetric normalized Laplacian $\mathbf{L}^{\text{sym}}$ with a normal one $\mathbf{L}$ in the smoothness term, the loss function reduces to the formulation in GFHF [5]:

$$\min \ \mathcal{L}_{\text{GFHF}}(\mathbf{F}) = \frac{1}{2}\text{tr}(\mathbf{F}^\top \mathbf{L}\mathbf{F}) \tag{2}$$

$$\text{s.t.} \ \mathbf{F}_{\boldsymbol{l}} = \mathbf{Y}_{\boldsymbol{l}}. \tag{3}$$

For simplicity, we first consider binary classification in the following and set $y_i \in \{+1,-1\}$. The target is to predict the label for unlabeled instances. Like other LP methods, we try to learn a real value function $f : \mathcal{V} \to [-1,1]$ to make prediction [5]. For ease of operation, the labeled instances are placed from the beginning as the Table 2 indicates.

As stated by Chapelle et al. [23], label propagation can be cast into a common framework which minimizes a quadratic loss function as

$$\mathcal{C}(\mathbf{f}) = \mathbf{f}^\top \mathbf{L}\mathbf{f} + \mu\|\mathbf{f}_{\boldsymbol{l}} - \mathbf{y}_{\boldsymbol{l}}\|^2 + \epsilon\|\mathbf{f}\|^2. \tag{4}$$

The last term is added to prevent some degenerate situations, for instance, when the graph has a connected component with no labeled sample [23].

Eq.(4) is similar to the regularization framework for the iterative algorithm proposed by [6]. In the extreme case when $\mu \to \infty, \epsilon = 0$, minimizing Eq.(4) is equivalent to minimize $\mathbf{f}^\top \mathbf{L}\mathbf{f}$ under the constraint $\mathbf{f}_{\boldsymbol{l}} = \mathbf{y}_{\boldsymbol{l}}$, which is the setting in [5].

The loss function is convex and so minimized when the derivative is set to 0, i.e.,

$$\mathbf{f}^* = (\mathbf{S} + \frac{1}{\mu}\mathbf{L} + \frac{\epsilon}{\mu}\mathbf{I})^{-1}\mathbf{S}\mathbf{y},$$

where $\mathbf{S}$ is a diagonal matrix with $S_{ii} = \mathbb{I}(1 \le i \le l)$. We can get the labels by simple matrix inversion. However, computing the inverse takes $O(n^3)$ time and $O(n^2)$ memory in general, which makes it infeasible on large-scale datasets.

3 Stochastic Label Propagation

In this section, we describe our simple and efficient algorithm SLP concretely. First, we derivate the updating routine for the common loss function. Then, the general framework is applied to the case in [5].

3.1 Binary Case

We start from the loss function in Eq.(4) and give the stochastic iterative solution to the optimization problem,

$$\min_{\mathbf{f}} \ \mathcal{C}(\mathbf{f})$$

where

$$\begin{aligned}\mathcal{C}(\mathbf{f}) &= \mathbf{f}^\top \mathbf{L}\mathbf{f} + \mu\|\mathbf{f}_{\boldsymbol{l}} - \mathbf{y}_{\boldsymbol{l}}\|^2 + \epsilon\|\mathbf{f}\|^2 \\ &= \sum_{i,j}^{n} W_{ij}(f_i - f_j)^2 + \mu\sum_{i=1}^{l}(f_i - y_i)^2 + \epsilon\sum_{i=1}^{n} f_i^2\end{aligned}$$

Let

$$\mathcal{C}_i(\mathbf{f}) = \sum_{j=1}^{n} W_{ij}(f_i - f_j)^2 + \mathbb{I}_{[l]}(i)\cdot\mu(f_i - y_i)^2 + \epsilon f_i^2$$

where $\mathbb{I}_{[l]}(i) = \mathbb{I}(1 \le i \le l)$, $i \in \{1,2,...,n\}$ represents the index of the chosen instance. We have $\mathbb{E}(\nabla\mathcal{C}_i(\mathbf{f})) = \frac{1}{n}\nabla\mathcal{C}(\mathbf{f})$. Specifically, according to the definition, we have

$$\mathcal{C}(\mathbf{f}) = \sum_{i=1}^{n}\mathcal{C}_i(\mathbf{f})$$

which implies that

$$\nabla\mathcal{C}(\mathbf{f}) = \sum_{i=1}^{n}\nabla\mathcal{C}_i(\mathbf{f})$$

If we randomly select a node with index i, we have $\Pr[i] = \frac{1}{n}$. The expectation of $\nabla\mathcal{C}_i(\mathbf{f})$

$$\begin{aligned}\mathbb{E}[\nabla\mathcal{C}_i(\mathbf{f})] &= \sum_{i=1}^{n}\nabla\mathcal{C}_i(\mathbf{f})\cdot\Pr[i] \\ &= \frac{1}{n}\sum_{i=1}^{n}\nabla\mathcal{C}_i(\mathbf{f}) \\ &= \frac{1}{n}\nabla\mathcal{C}(\mathbf{f}) \\ \mathbb{E}[\nabla\mathcal{C}_i(\mathbf{f})] &= \frac{1}{n}\nabla\mathcal{C}(\mathbf{f})\end{aligned}$$

Therefore, $\nabla\mathcal{C}_i(\mathbf{f})$ is an unbiased estimator of $\frac{1}{n}\nabla\mathcal{C}(\mathbf{f})$ where $\frac{1}{n}$ is a constant given a graph. We can now adopt SGD strategy [24] to solve $\mathbf{f}_{\mathbf{u}}$ by following updating

$$\mathbf{f}^{(t+1)} = \mathbf{f}^{(t)} - \eta\nabla\mathcal{C}_i(\mathbf{f}),$$

where the gradient

$$\nabla\mathcal{C}_i(\mathbf{f}) = (\mathbf{f} - f_i\mathbf{1})\circ\mathbf{W}_i^\top + \left(\mathbb{I}_{[l]}(i)\cdot\mu(f_i - y_i) + \epsilon f_i\right)\boldsymbol{e_i}.$$

However, the gradient takes $O(n)$ multiplication, which terribly slows down the iteration on a large graph. We observe that most of the elements in the vector of $\nabla\mathcal{C}_i(\mathbf{f})$ are zeros. In addition, the number of non-zero elements is determined by the degree $\deg(\text{v})$ of the node v, and $\deg(\text{v})$ is usually much smaller than n. In order to prevent unnecessary multiplication of zeros, we cache the neighbors' indexes for all nodes in the graph first. Then we

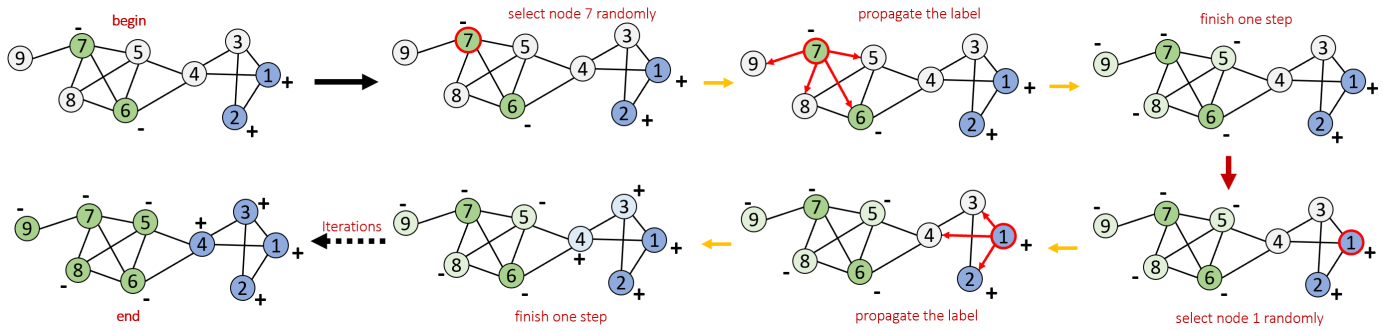

Fig. 3: Demo for the iteration of SLP.

use them to compute non-zero elements of $\nabla \mathcal{C}_i(\mathbf{f})$ and update $\mathbf{f}$. Trivially, the computation cost for each iteration is $O(\deg(\mathrm{v}))$.

Although SGD has made great success in large-scale machine learning, two more techniques are required for the effectiveness on large graphs. The first one is the warm start, which is attracting more and more interest in recent years. We manually select labeled instances and propagate their labels to neighbors before the iteration. Warm start can make more initial labels before iteration, which will improve the performance. The second one is to traverse all the shuffled nodes in each epoch as suggested by [25]. This process ensures that every node is accessed in each epoch and so as to reduce variance of $\nabla \mathcal{C}_i(\mathbf{f})$ as well as to improve the stability of the performance.

The overall description of the framework is presented in Algorithm 1. It takes affinity matrix $\mathbf{W}$, label vector $\mathbf{y}_l$, step size η, epoch T and two coefficients of regularization term as input. After the initialization in line 1, it stores neighbors' indexes in $\mathcal{I}$. Then, the warm-start process is taken from line 3 to 7, where every node with an initial label is selected to update its neighbors' labels. Finally, it takes T epochs and performs n steps per epoch to update $\mathbf{f}$ from line 8 to 17. As suggested in [25], the step size η is set to $O(\frac{1}{\sqrt{s}})$ where s is the total steps already taken. We call this framework SLP (Stochastic Label Propagation).

Algorithm 1 SLP Framework

Input: $\mathbf{W}$, $\mathbf{y_l}$, μ, ϵ, η, T
Output: $\mathbf{f_u}$

1: Initialize $\mathbf{f}_{\boldsymbol{u}}^{(1)}$ to $\mathbf{0}$, $\mathbf{f}_{\boldsymbol{l}}^{(1)}$ to $\mathbf{y_l}$
2: Find neighborhood indexes for node i and store them in $\mathcal{I}(i), i = 1, 2, ..., n$
3: **for** $i = 1, 2, ..., l$ **do**
4: **for** ind **in** $\mathcal{I}(i)$ **do**
5: $f_{ind}^{(1)} \leftarrow f_{ind}^{(1)} - \eta(f_{ind}^{(1)} - f_i^{(1)})W_{i,ind}$
6: **end for**
7: **end for**
8: **for** $t = 1, ..., \mathrm{T}$ **do**
9: $r \leftarrow$ random index from $[1, n]$
10: **for** $i = r, r+1, ..., n, 1, ..., r-1$ **do**
11: **for** ind **in** $\mathcal{I}(i)$ **do**
12: $f_{ind}^{(t+1)} \leftarrow f_{ind}^{(t)} - \eta(f_{ind}^{(t)} - f_i^{(t)})W_{i,ind}$
13: **end for**
14: $f_i^{(t+1)} \leftarrow f_i^{(t)} - \eta(\mathbb{I}_{[l]}(i) \cdot \mu(f_i^{(t)} - y_i) + \epsilon f_i^{(t)})$
15: update η
16: **end for**
17: **end for**

Algorithm 1 solves the regularization framework of label propagation with stochastic gradient and some techniques mentioned above. It gives a solution to large-scale regularization framework of label propagation and is easy to implement and light to run. Fig 3 is a demo to make the iteration more intuitive.

Similar to section 2, if we set $\mu \to \infty, \epsilon = 0$, minimize the quadratic loss function Eq.(4) is equivalent to minimize $\mathbf{f}^{\mathrm{T}}\mathbf{L}\mathbf{f}$ under the constraint of $\mathbf{f_l} = \mathbf{y_l}$, which is the setting in GRF [5]. More specifically, the problem is

$$\begin{aligned} \min_{\mathbf{f}} \ & \mathbf{f}^\top \mathbf{L}\mathbf{f} \\ s.t. \ & \mathbf{f_l} = \mathbf{y_l} \end{aligned}$$

We can derive corresponding algorithm for this problem from Algorithm 1 by removing the operations associated with μ, ϵ in line 14 as well as forcing $\boldsymbol{f_l} = \boldsymbol{y_l}$ in line 5 and line 12.

3.2 Multi-Class Case

Our methods can be extended to multi-class version easily. The label information is stored in a label matrix $\mathbf{Y} \in \{1, 0\}^{n \times c}$, where $Y_{ij} = 1$ if sample $\boldsymbol{x}_i$ has a label $j \in 1, \cdots, c$, and $Y_{ij} = 0$ otherwise. In the classification task, we have the constraints $\sum_{j=1}^{c} Y_{ij} = 1$. The soft label matrix $\mathbf{F} \in [0, 1]^{n \times c}$ saves the output values of the classifier. Specially, $\mathbf{F} = (\tilde{\boldsymbol{y}}_1; \tilde{\boldsymbol{y}}_2; ...; \tilde{\boldsymbol{y}}_n)$, $\tilde{\boldsymbol{y}}_i \in [0, 1]^{1 \times c}$ is the predictive scores for i-th instance and therefore we have predicted label $\hat{y}_i = \arg\max_{1 \le j \le c} \tilde{y}_{ij}$. For convenience, $\mathbf{F_u} = (\tilde{\boldsymbol{y}}_{l+1}; \tilde{\boldsymbol{y}}_2; ...; \tilde{\boldsymbol{y}}_n)$. We set $\mathbf{F}^{(0)} = \mathbf{Y}$.

We take the loss function in Eq. 2 as an example:

$$\begin{aligned} \mathcal{L}_{\mathrm{GFHF}}(\mathbf{F}) &= \frac{1}{2}\mathrm{tr}(\mathbf{F}^\top \mathbf{L}\mathbf{F}) && (5) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\|\tilde{\boldsymbol{y}}_i - \tilde{\boldsymbol{y}}_j\|^2 && (6) \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(n\sum_{j=1}^{n} W_{ij}\|\tilde{\boldsymbol{y}}_i - \tilde{\boldsymbol{y}}_j\|^2\right) && (7) \end{aligned}$$

Let

$$\begin{aligned} \mathcal{L}_i(\mathbf{F}) &= n\sum_{j=1}^{n} W_{ij}\|\tilde{\boldsymbol{y}}_i - \tilde{\boldsymbol{y}}_j\|^2 \\ &= n\sum_{j=1}^{n}\sum_{k=1}^{c} W_{ij}(\tilde{y}_{ik} - \tilde{y}_{jk})^2, \end{aligned}$$

we have $\mathcal{L}_{\mathrm{GFHF}}(\mathbf{F}) = \sum_{i=1}^{n} \mathcal{L}_i(\mathbf{F})$.

Similar to the process in the last section, we finally derive the updating formula

$$\mathbf{F}^{(t+1)} = \mathbf{F}^{(t)} - \eta \nabla \mathcal{L}_i(\mathbf{F})$$

the gradient

$$\nabla \mathcal{L}_i(\mathbf{F}) = C \cdot diag(\mathbf{W}_i)(\mathbf{F} - \mathbf{1} \cdot \tilde{\boldsymbol{y}}_i^\top),$$

where $\mathbf{1} = \{1\}^{n \times 1}$ and $\mathbf{W}_i$ is the i-th row of $\mathbf{W}$.

3.3 Inductive Model

The previous algorithm follows the transductive setting, which means that label predictions are only required for the given test set and cannot handle the points out of sample. However, sometimes inductive algorithm is needed when new test examples are presented one at a time and solving the entire problem turns out to be expensive. A inductive algorithm trys to infer a function on the entire space from given data samples and then evaluate the function at the test points. In this section, we incorporate an induction formula that can be computed in linear time.

Assuming that soft labels $\tilde{\boldsymbol{y}}_1, \cdots, \tilde{\boldsymbol{y}}_n$ have been computed by SLP above and we want to know the label $\tilde{\boldsymbol{y}}$ of a point. We can minimize the loss function $\mathcal{L}_{\text{GFHF}}(\mathbf{F})$ with respect to this new label $\tilde{\boldsymbol{y}}$,i.e.

$$\min \sum_{j=1}^{n} (\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{y}}_j)^2 \mathbf{M}_{\mathbf{X}}(\boldsymbol{x}, \boldsymbol{x}_j)$$

where $\mathbf{M}_{\mathbf{X}}$ is a function which generated the matrix $\mathbf{W}$ on $\mathbf{X} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n)$. We have the closed-form solution

$$\tilde{\boldsymbol{y}} = \frac{\sum_{j=1}^{n} \mathbf{W}_{\mathbf{X}}(\boldsymbol{x}, \boldsymbol{x}_j) \tilde{\boldsymbol{y}}_j}{\sum_{j=1}^{n} \mathbf{W}_{\mathbf{X}}(\boldsymbol{x}, \boldsymbol{x}_j)} \tag{8}$$

We define $\mathbf{M}_{\mathbf{X}}$ as a graph-dependent function, i.e.,

$$\mathbf{M}_{\mathbf{X}}(\boldsymbol{x}, \boldsymbol{x}_j) = \begin{cases} \text{weight of edge}, & \text{if } \boldsymbol{x} \text{ and } \boldsymbol{x}_j \text{ are connected.} \\ 0, & \text{otherwise.} \end{cases}$$

then the final prediction for point $\boldsymbol{x}$ is the weighted voting result of its neighbors in the graph. The time complexity of the inductive model is $O(d)$ for each new coming nodes, where d is the degree of the node.

4 Analyses

This section contains our analyses of SLP. In the first part, we give a convergent result and analyze the complexity of our algorithm. Then we discuss the necessity of warm start.

4.1 Convergence

We prove the following convergent rate for our algorithm:

Theorem 4.1.1. *Consider binary task. Assume we have a dataset with n instances, the constructed graph $G = (V, E)$ has m edges and* SLP *algorithm runs for T epochs. Assume that $G > 0$ is a constant. If SLP runs for T epochs with $\eta = O(\sqrt{\frac{n^2}{G^2 T}})$, Then, the expectation of the excess risk would be bounded by*

$$\mathbb{E}[\mathcal{C}(\mathbf{F})] - \mathcal{C}(\mathbf{F}^*) \leq O(\frac{nG}{\sqrt{T}})$$

Proof. SGD has been well analyzed in the past decades. By adopting the proof from Chapter 14.3 of [26], we have the convergent rate in Lemma 4.1.2,

Lemma 4.1.2. *Let $B, G > 0$. Let $\mathbf{F}^* \in \arg\min_{\|\mathbf{F}\|_2 \leq B} \mathcal{C}(\mathbf{F})$. Assume that SLP run for T epochs with $\eta \leq \sqrt{\frac{B^2}{G^2 T}}$, also assume that $\|\nabla \mathcal{C}_i(\mathbf{F})\| \leq G$ with probability 1. Then,*

$$\mathbb{E}[\mathcal{C}(\mathbf{F})] - \mathcal{C}(\mathbf{F}^*) \leq O(\frac{BG}{\sqrt{T}})$$

Since $\mathbf{F}$ is the soft label matrix, we have $\sum_{j=1}^{c} \mathbf{F}_{ij} \leq 1, i = 1, 2, \cdots, n$, so that $\|\mathbf{F}\|_2 \leq B = O(n)$. Applying them to Lemma 4.1.2, the Theorem 4.1.1 is proved. □

Therefore, for any $\epsilon > 0$, to achieve $\mathbb{E}[\mathcal{C}(\mathbf{f})] - \mathcal{C}(\mathbf{f}^*) \leq \epsilon$, it suffices to run the algorithm for a number of epochs which satisfies $T \geq \frac{B^2 \rho^2}{\epsilon^2}$. In practice, The algorithm often obtains a quite good approximate solution by passing the unlabeled instances in less than five times (five epochs). The inner two *for-loop*s actually go through all edges in E and have a time complexity in $O(|E|)$, so the proposed algorithm can reach a good solution in $O(T|E|)$ time, which can be furthermore equivalent to $O(Tn)$ w.r.t sparse graphs. SLP needs to put the affinity matrix $\mathbf{W}$, the corresponding non-zero indexes $\mathcal{I}$ and predictive vector $\mathbf{f}$ into memory, so the space cost is $O(|E|)$.

4.2 Warm Start

In the following part, we try to explain why the warm-start process is required to perform in large-scale label propagation. We denote the length of the shortest walk between two nodes u and v as $dis(u, v)$. Given the initially labeled nodes, we can divide the V into several subsets $\hat{V}_l, V_1, V_2, ..., V_k$. $\hat{V}_l$ is the set of the initially labeled nodes, and $V_i = \{u | \min_{v \in V_l} dis(u, v) = i\}, i = 1, 2, ..., k$. The predictive label of a node is regarded as **valid** if it has received the label information from initially labeled nodes, and otherwise **invalid**.

Observation 4.2.1. *Based on the fact that label propagates through one edge per epoch, the predictions of the nodes in V_i will not be valid until the propagation of epoch i is done.*

We will show that the performance is affected by the density of graph and initially labeled ratio.

Theorem 4.2.2. *Let G be a graph with n nodes and the degree of all nodes in G is d. With l initial labeled nodes, after t-epoch propagation, at most $\min\{n, \frac{(d^{t+1}-1)l}{d-1}\}$ nodes will be valid.*

Proof. According to Observation 4.2.1, after t-epoch propagation, the predictions of the nodes in $\hat{V}_l, V_1, V_2, ..., V_t$ are valid, and we have

$$|\hat{V}_l \cup V_1 \cup ... \cup V_t| \leq |\hat{V}_l| + |V_1| + ... + |V_t| \tag{9}$$

$$\leq l + ld + ... + ld^t \tag{10}$$

$$= \frac{(d^{t+1} - 1)l}{d - 1}. \tag{11}$$

The two sides of inequality (9) are equal if and only if $\hat{V}_l, V_1, V_2, ..., V_t$ are mutual disjoint, and the two sides of inequality (10) are equal if and only if the graph G consists of d-ary trees and the roots of these trees are selected as initially labeled nodes.

Also notice that $|\hat{V}_l \cup V_1 \cup ... \cup V_t| \leq n$, we have

$$|V_\ell \cup \hat{V}_l \cup ... \cup V_t| \leq \min\{n, \frac{(d^{t+1} - 1)l}{d - 1}\},$$

and the theorem is proved. □

TABLE 3: Label Propagation on Large-Scale Network Analysis

GRAPH	LABELED RATIO	ACC		AUC		TIME(S)	
		TBTL	SLP	TBTL	SLP	TBTL	SLP
WIKIPEDIA (1,791,489 NODES, 28,511,807 EDGES)	0.1%	$.607 \pm .009$	$.643 \pm .008$	$.539 \pm .006$	$.591 \pm .006$	33.1 ± 1.4	7.0 ± 0.4
	0.2%	$.655 \pm .003$	$.687 \pm .004$	$.559 \pm .004$	$.606 \pm .004$	37.2 ± 1.7	6.4 ± 0.1
	0.4%	$.669 \pm .002$	$.703 \pm .002$	$.581 \pm .003$	$.630 \pm .005$	45.6 ± 3.4	6.5 ± 0.2
	0.8%	$.684 \pm .001$	$.717 \pm .002$	$.605 \pm .002$	$.652 \pm .004$	60.2 ± 3.9	6.5 ± 0.1
	1.6%	$.698 \pm .001$	$.727 \pm .001$	$.630 \pm .001$	$.668 \pm .003$	90.6 ± 5.8	6.5 ± 0.1
	3.2%	$.712 \pm .001$	$.732 \pm .001$	$.656 \pm .001$	$.681 \pm .003$	146.0 ± 7.0	6.5 ± 0.1
LIVEJOURNAL (3,997,962 NODES, 34,681,189 EDGES)	0.1%	$.979 \pm .000$	$.950 \pm .036$	$.515 \pm .004$	$.619 \pm .020$	107.2 ± 5.4	10.3 ± 0.1
	0.2%	$.979 \pm .000$	$.960 \pm .011$	$.530 \pm .004$	$.646 \pm .014$	113.7 ± 4.4	10.2 ± 0.1
	0.4%	$.979 \pm .000$	$.964 \pm .004$	$.549 \pm .003$	$.670 \pm .011$	115.6 ± 4.2	10.4 ± 0.1
	0.8%	$.980 \pm .000$	$.966 \pm .003$	$.575 \pm .004$	$.696 \pm .011$	106.1 ± 8.6	10.4 ± 0.1
	1.6%	$.981 \pm .000$	$.969 \pm .002$	$.614 \pm .003$	$.732 \pm .005$	115.1 ± 6.3	10.5 ± 0.2
	3.2%	$.982 \pm .000$	$.973 \pm .001$	$.657 \pm .003$	$.769 \pm .004$	144.0 ± 8.7	10.5 ± 0.2
ORKUT (3,072,441 NODES, 117,185,083 EDGES)	0.1%	$.720 \pm .010$	$.594 \pm .045$	$.559 \pm .007$	$.701 \pm .012$	256.5 ± 113.9	32.4 ± 1.9
	0.2%	$.773 \pm .003$	$.623 \pm .013$	$.528 \pm .004$	$.712 \pm .010$	222.5 ± 94.6	30.7 ± 0.6
	0.4%	$.773 \pm .001$	$.635 \pm .008$	$.532 \pm .002$	$.718 \pm .007$	248.9 ± 95.3	31.1 ± 0.5
	0.8%	$.772 \pm .001$	$.636 \pm .007$	$.538 \pm .002$	$.722 \pm .003$	232.9 ± 108.6	30.9 ± 0.6
	1.6%	$.772 \pm .001$	$.645 \pm .005$	$.546 \pm .001$	$.719 \pm .002$	247.9 ± 117.0	30.9 ± 0.5
	3.2%	$.771 \pm .001$	$.657 \pm .004$	$.558 \pm .001$	$.724 \pm .002$	205.7 ± 89.4	31.3 ± 0.4

Corollary 4.2.3. *Let G be a graph with n nodes and the degree of all nodes in G is d. With l initial labeled nodes, the predictions on graph G will all be valid with t epochs, and $t \in \Omega(\log_d \frac{n(d-1)}{l})$.*

Proof. According to Theorem 4.2.2, after t-epoch propagation, at most $\min\{n, \frac{(d^{t+1}-1)l}{d-1}\}$ nodes will be valid. To make sure the predictions on the graph G are all valid, let $\frac{(d^{t+1}-1)l}{d-1} \geq n$, then we have

$$t \geq \log_d \left(\frac{n(d-1)}{l} + 1 \right).$$

□

Theorem 4.2.2 and Corollary 4.2.3 show that the sparser the graph is, and the fewer the initial labels are, the more invalid predictions will be made in the first few epochs. The warm start process spreads initial labels to neighborhoods before main iteration (line 3 to 7 in Algorithm 1), which helps reduce invalid predictions when the algorithm terminates.

5 EXPERIMENTS

In this section, we evaluate SLP in various tasks including large-scale network analysis and semi-supervised classification. Then, we analyze the influence of parameters and graphs on SLP. Finally, we show how to prevent performance degeneration in semi-supervised classification with some low-quality graphs.

5.1 Experimental Setup

The proposed approach is compared with a number of methods, including supervised method 10-NN as a baseline, ARG [9], Eigen [9] and TbTL [17]. The ARG method uses a few anchor points which cover the entire point cloud to build a smaller graph with strong representative power. ARG method needs an anchor graph the final prediction is obtained by a simple linear combination of anchor points. The Eigen method approximates the graph Laplacian matrix by spectral decomposition, and in this way, the problem is solved in a reduced dimensional space. The TbTL method first generates a minimum spanning tree for given graph and then solve the problem with a designed Laplacian solver. The work presented in [5], [6] fails to cope with the problem scale of our experiments, therefore, we do not include them in the experiments. For SLP, the implementation of the extreme case mentioned at the end of Section 3.1 is evaluated in the following three tasks.

The codes of ARG, Eigen, TbTL are all shared by their authors. The parameters for comparison methods are chosen from the recommended ones suggested by the authors. We adopt KD-tree algorithm to construct approximate nearest neighbor graphs when the dataset does not provide a graph. For ARG, anchor points are generated by k-means clustering and the released LAE version is chosen to run. For Eigen, the number of eigenvectors is set to 160. For TbTL, the tree type is set to the minimum spanning tree, the number of trees is 16 as suggested by the authors and the regularization factor is selected from $0.01, 1, 100$ by performing 5-fold cross-validation on the training set. We use default hyper-parameters for SLP. The step size η is set to $\frac{1}{10\sqrt{t}}$ if not specify where t is the next step to take, and epoch T is set to 6. $0.1\%, 0.2\%, 0.4\%, 0.8\%, 1.6\%, 3.2\%$ of instances are randomly selected as labeled data. Results are averaged over 20 independent repetitions. The performance is measured in terms of both accuracy and AUC. We run these evaluations on a PC with 3.2GHz AMD Ryzen 1400 CPU and 16GB RAM.

5.2 Application of SLP

In this part, we evaluate SLP in different tasks compared with state-of-the-art methods. The tasks includes large-scale network analysis, forest covertype categorization, handwritten character recognition and some benchmark datasets.

5.2.1 Tasks on Large-Scale Network Analysis

We collect three large-scale network datasets and perform label propagation on them. These graphs [27], [28] include graphs from Wikipedia[1], Livejournal[2] and Orkut[3]. Specifically, Wikipedia is a web graph Wikipedia hyperlinks collected in September 2011 with $1,791,489$ nodes and $28,511,807$ edges. LiveJournal is a free on-line blogging community where users declare friendship each

1. http://snap.stanford.edu/data/wiki-topcats.html
2. http://snap.stanford.edu/data/com-LiveJournal.html
3. http://snap.stanford.edu/data/com-Orkut.html

TABLE 4: Performance on Forest Covertype Categorization

	METHOD	0.1%	0.2%	0.4%	0.8%	1.6%	3.2%
ACC	10NN	.684 ± .007	**.709 ± .006**	.732 ± .004	.758 ± .003	.789 ± .002	.822 ± .001
	ARG	**.688 ± .008**	.703 ± .005	.717 ± .004	.728 ± .002	.737 ± .001	.741 ± .001
	EIGEN	.513 ± .003	.530 ± .017	.572 ± .028	.629 ± .004	.636 ± .001	.646 ± .004
	TBTL	.652 ± .014	.698 ± .007	.739 ± .005	.777 ± .003	.811 ± .002	.840 ± .001
	SLP	.633 ± .011	.691 ± .006	**.742 ± .004**	**.787 ± .003**	**.824 ± .002**	**.857 ± .001**
AUC	10NN	.687 ± .007	.711 ± .006	.734 ± .004	.759 ± .002	.790 ± .002	.823 ± .001
	ARG	**.755 ± .007**	**.771 ± .005**	.796 ± .002	.812 ± .002	.822 ± .001	.828 ± .001
	EIGEN	.718 ± .023	.724 ± .023	.734 ± .022	.734 ± .009	.741 ± .007	.752 ± .003
	TBTL	.653 ± .015	.695 ± .008	.738 ± .005	.778 ± .003	.813 ± .002	.845 ± .001
	SLP	.731 ± .011	.768 ± .007	**.801 ± .004**	**.837 ± .003**	**.868 ± .002**	**.899 ± .001**
TIME(S)	10NN	457.2 ± 35.3	615.7 ± 70.4	913.2 ± 12.1	1544.0 ± 55.5	2767.7 ± 64.5	5270.0 ± 745.8
	ARG	786.1 ± 0.0	786.1 ± 0.0	786.1 ± 0.0	786.1 ± 0.0	786.1 ± 0.0	786.1 ± 0.0
	EIGEN	8.3 ± 0.3	8.2 ± 0.2	9.1 ± 1.7	8.9 ± 1.5	8.2 ± 0.1	8.3 ± 0.3
	TBTL	20.1 ± 1.2	25.0 ± 0.8	31.1 ± 1.0	38.3 ± 1.1	46.4 ± 1.4	55.2 ± 1.2
	SLP	**1.0 ± 0.0**	**0.9 ± 0.0**	**0.9 ± 0.0**	**0.9 ± 0.0**	**0.9 ± 0.0**	**1.0 ± 0.0**

other and the graph contains $3,997,962$ nodes and $34,681,189$ edges. Orkut is a free on-line social network where users form friendship each other and the graph contains $3,072,441$ nodes and $117,185,083$ edges.

We can not access the profiles of nodes in these graphs and the labels of these nodes are generated by the ground-truth communities information provided by [27]. We adopt the one-vs-rest strategy by regarding the most frequent label as positive one. Because Anchor and Eigen methods take features of instances as input and fail to run in these tasks, we can only compare SLP with TbTL. We will evaluate Anchor and Eigen in the next part.

The performance is shown in Table 3. SLP achieves highly competitive performance with TbTL with much shorter time. Although TbTL seems to have higher accuracy, its poor AUC performance indicates that it benefits from the unbalance of the classes. We observe that most of TbTL's predictions are the label with the larger proportion in the training set. The superior of SLP over TbTL owes to the fact that SLP does not modify the given graph and maintain origin edges for label propagation. In addition, SLP is generally at least five times faster than TbTL.

5.2.2 Task on Forest Covertype Categorization

The second task is collected from UCI[4] and the target is to predict forest cover type from cartographic variables only. On this dataset, more state-of-the-art methods can be conducted for comparison. The dataset has $581,012$ instances and we construct a 10NN graph based on the features. Following the procedure in previous part, we transform the task into a binary task by treating the most frequent label as the positive label and the rest as the negative one.

Average accuracy, AUC and running time with standard deviation are shown in Table 4. On Covertype dataset, EIGEN, TbTL, and SLP are all inferior to 10NN and ARG with 0.1% and 0.2% labeled data. ARG may achieve better performance with more anchor points, but it's restricted by the memory cost. SLP performs better with more initial labeled data. Besides, given a graph, SLP spends much shorter time than competitors and achieve the highest AUC at most time.

5.2.3 Tasks on Benchmark Data

To evaluate our proposal, we conduct experiments on several medium scale datasets collected from UCI[5] and MNIST[6]. We generate four binary datasets: Adults, Ijcnn, Mnist$3vs8$ [29] and Devanagari. Adults dataset predict whether income exceeds 50K/yr based on census data. Devanagari is a dataset of Devanagari handwritten character images. There are 46 classes of characters with 2000 examples each and resolution of 32×32. We select 9 similar characters $(ka, cha, taamatar, thaa, tabala, da, pa, yaw, la\ )$ and extend the subset by shifting each image by one or two pixels in each direction inspired by [9], [17]. So each character has total 18000 examples and we want to distinguish ka from other 8 characters. Similarly, we select one of most challenging binary classification task $3vs8$ in handwritten digits set MNIST and extend them five times to original. In order to accelerate the running speed, we perform PCA to reduce dimensions. Experiments repeat for 20 different initial labels and AUCs are reported.

Figure 4 displays average AUC with standard deviation on the four datasets with 10NN graph. Tested methods perform differently on six datasets. Specifically, SLP and Anchor outperform other comparisons much except for Adults dataset. The performance of Eigen is stable but less competitive to SLP in four datasets. Although TbTL method achieves high accuracy, it gets low AUC in most time, which means the prediction of Anchor is highly imbalanced. In short, our algorithm makes competitive performance on benchmark datasets. We'll further evaluate the influence of the graph's density in Section 5.3.

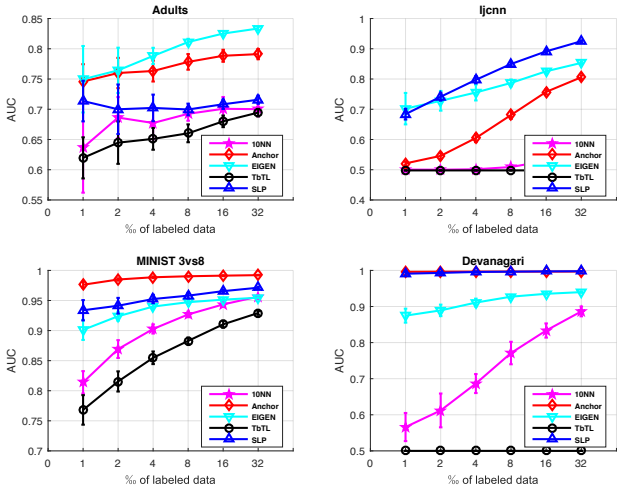

Fig. 4: AUC on benchmark datasets.

4. https://archive.ics.uci.edu/ml/datasets/Covertype
5. archive.ics.uci.edu/ml/index.php
6. http://yann.lecun.com/exdb/mnist/

TABLE 5: Performance on Handwritten Devanagari Character Recognition (Multi-Class Classification)

		Method	0.1%	0.2%	0.4%	0.8%	1.6%	3.2%
Devanagari_9c (162,000 instances)	Acc(%)	10NN	49.14 ± 1.81	58.41 ± 1.53	66.88 ± 1.55	74.40 ± 0.97	80.86 ± 0.54	86.02 ± 0.37
		Anchor	72.80 ± 0.99	76.69 ± 0.79	80.72 ± 0.58	84.22 ± 0.34	86.84 ± 0.32	88.48 ± 0.13
		TbTL	51.16 ± 2.51	61.13 ± 1.45	67.94 ± 1.27	74.65 ± 0.48	80.13 ± 0.43	84.40 ± 0.30
		SLP	$\mathbf{79.06 \pm 1.17}$	$\mathbf{82.35 \pm 1.01}$	$\mathbf{85.72 \pm 0.41}$	$\mathbf{88.21 \pm 0.44}$	$\mathbf{90.42 \pm 0.29}$	$\mathbf{92.02 \pm 0.18}$
	Time(s)	10NN	1.95	3.37	5.67	10.74	20.87	42.28
		Anchor	221.51	221.51	221.52	221.51	221.51	221.51
		TbTL	2.77	2.73	2.74	2.76	2.76	2.82
		SLP	0.91	0.92	0.91	0.93	0.99	0.97
Devanagari_46c (828,000 instances)	Acc(%)	10NN	34.33 ± 0.88	44.21 ± 0.74	53.55 ± 0.56	62.62 ± 0.38	70.45 ± 0.25	77.22 ± 0.18
		Anchor	45.77 ± 0.80	50.34 ± 0.40	53.16 ± 0.30	54.74 ± 0.13	55.62 ± 0.09	56.19 ± 0.08
		TbTL	44.50 ± 0.57	55.43 ± 0.41	53.28 ± 0.27	71.56 ± 0.17	77.72 ± 0.16	83.02 ± 0.08
		SLP	$\mathbf{62.98 \pm 0.95}$	$\mathbf{69.69 \pm 0.69}$	$\mathbf{74.66 \pm 0.38}$	$\mathbf{79.16 \pm 0.23}$	$\mathbf{82.94 \pm 0.13}$	$\mathbf{86.36 \pm 0.11}$
	Time(s)	10NN	274.73	531.36	1033.25	2042.58	4057.88	8280.25
		Anchor	1113.16	1113.15	1113.15	1113.15	1113.19	1113.15
		TbTL	51.97	51.59	51.85	52.00	51.53	51.28
		SLP	26.61	26.85	28.92	29.33	30.79	33.78

5.2.4 Tasks on Handwritten Character Recognition

We evaluate the **multi-class** version algorithm on the handwritten character recognition tasks. The origin dataset has 46 classes of characters with 2000 examples for each and resolution of 32×32. To extend the dataset, we shift each image by one or two pixels in each direction inspired by [9], [17]. So each character has total 18000 examples and we finally make a dataset with $828,000$ instances. We use 10NN graphs in this part. Experiments repeat for 20 different initial labels and both accuracy and time are reported. The running time does not include the time for data pretreatment such as graph construction or clustering. Since Eigen method only handle binary task, we don't include it in this part.

The result is shown in Table 5. In the task with 9 classes, SLP is the most efficient and effective one. All of the evaluated methods takes seconds to solve the problem. Anchor is efficient in inference phase, but it takes some time to construct the anchor graph with given cluster points. In the task with 46 classes, SLP maintains the effectiveness as in former tasks. It converges within half a minute. In the part, however, SLP no longer runs five times faster than the comparisons because of the sorting operation after label propagation.

5.3 Behavior of SLP

In this part, we further explore the behavior of SLP including the influence of the coefficient and the graph. We also evaluate the convergence of SLP on benchmark datasets with various initial labeled data by recording the performance in first few epochs. Finally, we evaluate the performance of inductive SLP.

5.3.1 Benchmark Tasks with Different Graphs

In order to evaluate how the density of the graph influences the performance of SLP, we further conduct experiments on several benchmark datasets collected from UCI[7] and MINIST[8] [29]. Six binary datasets, i.e., Adults, Ijcnn, Minist$3vs8$ and Devanagari are conducted. We construct 4NN, 6NN, 8NN and 10NN graphs for these datasets and compare the predictive accuracy of SLP with different graphs.

Results in Figure 5 show that our proposal can have a better performance on a denser graph. Commonly, 8NN and 10NN graphs give competitive results, while 4NN and 6NN graphs performs worse. This is consistent with Theorem 4.2.2. Particularly, the denser the graph is, the more predictions will be valid in the first few epochs.

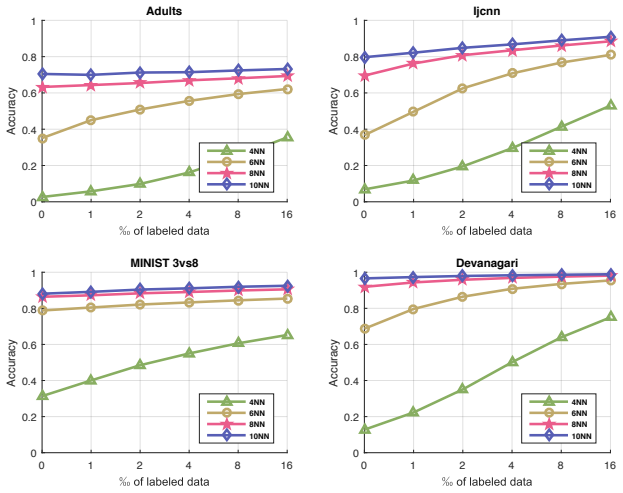

Fig. 5: Performance of SLP with different graphs.

5.3.2 Influence of the Coefficient

In Eq. 4, there are two coefficients μ and ϵ for regularizers. In previous empirical studies, they are set as default values 0s and our algorithm SLP performs well. In Figure 6, we further studies the influence of μ and ϵ on six medium datasets with 10NN graphs and 0.1% initial labels. The values of μ and ϵ are both selected from $\{0.01, 0.1, 1, 10, 100\}$. We present the average accuracy for 20 different initial labels for each setting.

It can be seen that, the performance of SLP is quite insensitive to the setting of μ and ϵ. One possible reason is that, μ only controls the propagation from initial labeled nodes, and it influence is rather low-weighted if the labeled ratio is small. ϵ actually tends to minimize the norm of the predictive vector, which have little impact on final prediction.

5.3.3 Convergence Evaluation

We analysis the convergence of SLP on benchmark datasets. Figure 7 shows the performance of SLP in the first few epochs

7. https://archive.ics.uci.edu/ml/index.php
8. http://yann.lecun.com/exdb/mnist/

with different labeled ratio. For each datasets, we construct 10NN graph and set η, μ, ϵ as $\frac{1}{100\sqrt{T}}, 1, 0$. We can see that SLP can converge in less than 5 epochs. Further more, SLP would converge faster with more initial labels, sometimes even in 2 epochs.

5.3.4 Performance of Inductive Model

We evaluate the inductive model of SLP on three large-scale networks used in Section 5.2.1. We randomly sample 0.1% and 1% of the nodes to the testing set, and another 1% of the nodes to have initial labels. We first run label propagation on a network whose nodes in the testing set are removed, and then make prediction for the nodes in the testing set. For each network, we sample 20 distinct test sets and the results are averaged over 20 independent initializations. We compare the performance of inductive model and transductive model. The results in Table 6 show that inductive model makes identical performance in orkut and wiki, but degenerate in livej.

The reason for the performance degradation on livej is that deleting testing nodes from the graph has caused damage to the connectivity of the graph, making it impossible for some nodes to be labeled in label propagation, which further prevents some of the testing nodes to receive supervision information. In order to further evaluate the inductive model, for livej, we sort the nodes based on the degrees of them and then sample $p\%$ testing nodes from the first $1.5p\%$ ascend nodes in order to prevent damage to the connectivity. The performance (denoted as *or_ind*) is much better than the case where testing nodes are randomly sampled from all nodes.

TABLE 6: Accuracy of Inductive/Transductive Model for Large-Scale Network Analysis.

Data	livej			orkut		wiki	
Ratio	trans.	ra_ind.	or_ind.	trans.	ind.	trans.	ind.
0.1%	96.77	53.38	89.71	64.12	63.45	71.83	73.64
1.0%	96.69	53.31	83.83	64.29	63.48	72.06	73.39

TABLE 7: AUC of Inductive Model Compared with Baselines.

DATA	SLP_IND	ANCHOR_IND	TBTL_IND
ADULTS	70.13 ± 3.17	78.67 ± 3.01	67.68 ± 2.85
IJCNN	84.97 ± 2.22	71.98 ± 3.16	67.16 ± 2.74
MINIST 3VS8	96.49 ± 0.81	99.58 ± 0.20	95.28 ± 0.99
DEVANAGARI	98.65 ± 0.68	99.95 ± 0.10	97.94 ± 0.81

We then further compare the inductive model of SLP (SLP_IND) with other baselines including inductive extension of Anchor (Anchor_IND) and TbTL (TbTL_IND) on semi-supervised learning tasks. Anchor_IND and TbTL_IND are both extended from transductive ones by adding a nearest-neighbor predictor for testing data. Semi-supervised learning algorithms require features of instances, which is not provided in orkut and wiki and livej, so we have to conduct the experiments on benchmark data. We construct 10NN graphs for Anchor_IND and TbTL_IND. The results are shown in Table 7. SLP_IND always has the performance above average and does better than TbTL_IND. Anchor_IND does the best in handwritten character recognition tasks such as MINIST and Devanagari, followed by SLP_IND closely.

5.4 Robust GSSL with Multiple Graphs

In the Figure 5, we can observe that our proposal is somehow more sensitive to the quality of graph. Further more, it is widely

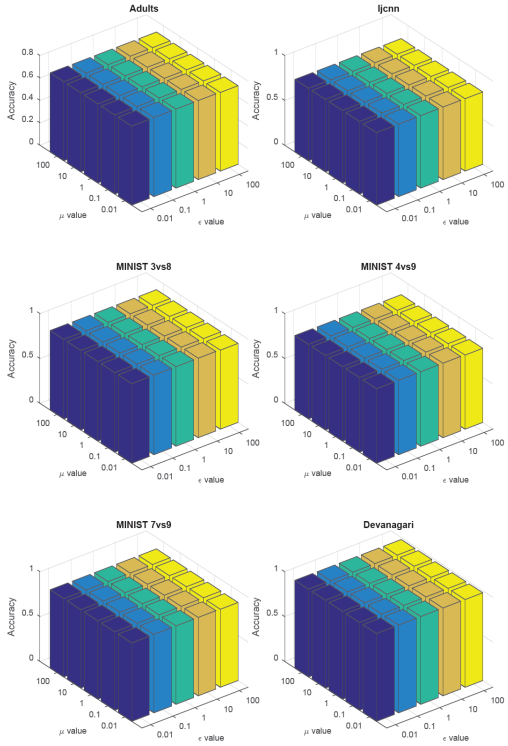

Fig. 6: Influence of μ and ϵ on six datasets.

accepted that the key for the success of graph-based semi-supervised learning (GSSL) is to construct an excellent graph for given training data, rather than designing various learning or optimization algorithms [30], [31], [32], [33]. For this reason, many efforts have been devoted to graph construction during the decades, e.g., [31], [34], [35]. Generally, excellent graph construction remains challenging or an open problem, especially when domain knowledge is scarce or insufficient to afford a reliable graph construction.

Beyond constructing excellent graphs, one another or more serious issue is that constructing graph improperly may even deteriorate performance, which means its performance is worse than that of its supervised counterpart (supervised nearest neighbor method) with only labeled data [6], [30], [31], [33]. These phenomena clearly conflicts with the original intention of GSSL. They will also encumber the deployment of GSSL in reality, because the GSSL users typically expect that employing GSSL methods should not be worse than direct supervised nearest neighbor methods. Therefore, it is highly desirable to derive performance-safe GSSL method, which would not be outperformed by its supervised counterpart.

We try incorporating a robustness technique named LEAD [36] to improve the performance of our algorithm. The basic assumption of LEAD is that the predictive results on high-quality graphs may have a large margin separation. Unlabeled instances lying within the margin is regarded as risky to used and their labels are assigned with the direct supervised learning method. Given several candidate graphs, assuming that we don't have an idea which graph is better, LEAD can judge the quality of graphs and make safe prediction. We incorporate LEAD to SLP and test its performance on benchmark datasets with four candidate graphs.

The results are shown in Table 8. SLP-Lead can make full use of the high-quality graphs. The performance in Adults and Ijcnn

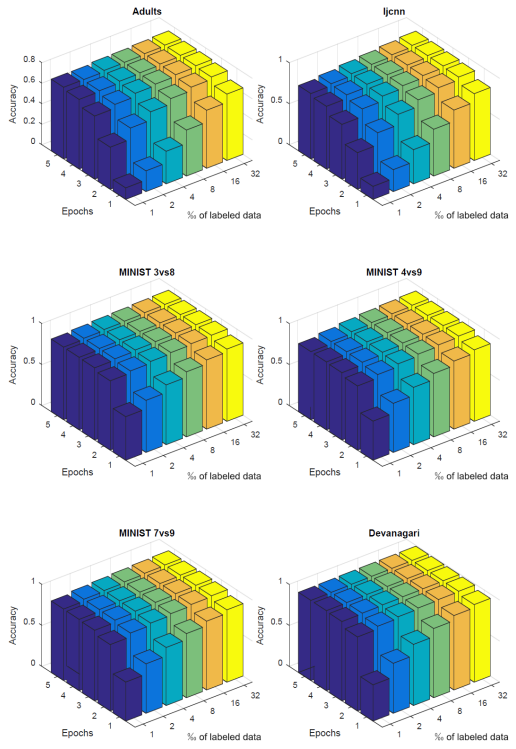

Fig. 7: The performance of SLP in different epochs.

are not displayed in this table because SLP loses to the nearest-neighbor method with all the candidate graphs, which indicates that label propagation algorithm may not be the right choice for these two datasets and the performance degeneration has nothing to do with the quality of graphs. In MNIST4vs9, MNIST7vs9, Devanagari datasets, SLP-Lead seldom loses to supervised learning method given candidate graphs, even though we have no idea which graph is better.

6 Conclusion

In this paper, we propose a fast and scalable label propagation method named SLP. Its procedure is derived from stochastic gradient descent and it is simple to implement. Since the method inherits the framework of SGD well, it can incorporate the recent process of SGD such as SVRG [37] and AsySVRG [22], combined with practical needs to reduce variance, distributed execution, etc. Experiments on million-scale graphs demonstrate that our method is much faster than leading approaches and achieves highly competitive performance. The contribution of this work is that we firstly incorporate the stochastic method to label propagation and provides a different inspiration to handle large-scale label propagation tasks with clear practical advantages. Besides, we incorporate a safe technique to reduce performance degeneration of GSSL with multiple graphs with uncertain quality.

Acknowledgments

This research was supported by the National Key R&D Program of China (2017YFB1001903) and the National Natural Science Foundation of China (61772262).

References

[1] B. Avrim and C. Shuchi, "Learning from labeled and unlabeled data using graph mincuts," in *Proceedings of the twentieth International Conference on Machine Learning*, Williamstown, MA, 2001, pp. 19–26.

[2] T. Joachims, "Transductive learning via spectral graph partitioning," in *Proceedings of the twentieth International Conference on Machine Learning*, Washington, DC, 2003, pp. 290–297.

[3] A. Blum, J. D. Lafferty, M. R. Rwebangira, and R. Reddy, "Semi-supervised learning using randomized mincuts," in *Proceedings of the twenty-first International Conference on Machine Learning*, Banff, Canada, 2004.

[4] X. Zhu and J. D. Lafferty, "Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning," in *Proceedings of the Twenty-Second International Conference on Machine Learning, Bonn, Germany*, 2005, pp. 1052–1059.

[5] X.-J. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the twentieth International Conference on Machine Learning*, Washington, DC, 2003, pp. 912–919.

[6] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2004, pp. 321–328.

[7] F. Wang and C. Zhang, "Label propagation through linear neighborhoods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp. 55–67, 2008.

[8] K. Zhang, J. T. Kwok, and B. Parvin, "Prototype vector machine for large scale semi-supervised learning," in *Proceedings of the twenty-sixth Annual International Conference on Machine Learning*, Montreal, Canada, 2009, pp. 1233–1240.

[9] W. Liu, J.-F. He, and S.-F. Chang, "Large graph construction for scalable semi-supervised learning," in *Proceedings of the twenty-seventh International Conference on Machine Learning*, Haifa, Israel, 2010, pp. 679–686.

[10] W. Liu, J. Wang, and S.-F. Chang, "Robust and scalable graph-based semisupervised learning," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2624–2638, 2012.

[11] G. Lever, T. Diethe, and J. Shawe-Taylor, "Data dependent kernels in nearly-linear time," in *Artificial Intelligence and Statistics*, La Palma, 2012, pp. 685–693.

[12] R. Fergus, Y. Weiss, and A. Torralba, "Semi-supervised learning in gigantic image collections," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2009, pp. 522–530.

[13] A. Talwalkar, S. Kumar, M. Mohri, and H. Rowley, "Large-scale svd and manifold learning," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 3129–3152, 2013.

[14] M. Herbster, M. Pontil, and S. R. Galeano, "Fast prediction on a tree," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2009, pp. 657–664.

[15] N. Cesa-Bianchi, C. Gentile, and F. Vitale, "Fast and optimal prediction on a labeled tree," in *Annual Conference on Learning Theory*, Montreal, Canada, 2009, pp. 145–156.

[16] F. Vitale, N. Cesa-Bianchi, C. Gentile, and G. Zappella, "See the tree through the lines: The shazoo algorithm," in *Advances in Neural Information Processing Systems*, Granada, Spain, 2011, pp. 1584–1592.

[17] Y.-M. Zhang, X.-Y. Zhang, X.-T. Yuan, and C.-L. Liu, "Large-scale graph-based semi-supervised learning via tree laplacian solver." in *Proceedings of the thirtieth AAAI Conference on Artificial Intelligence*, New Orleans, LA, 2016, pp. 2344–2350.

[18] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of International Conference on Computational Statistics*, Paris, France, 2010, pp. 177–186.

[19] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the seventeenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, 2011, pp. 69–77.

[20] T. White, *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[21] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2010, pp. 2595–2603.

[22] S. Zhao and W. Li, "Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, Arizona, 2016, pp. 2379–2385.

[23] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-supervised learning*. MIT Press, 2006.

TABLE 8: The predictive accuracy when running with different graph. The win/tie/loss counts of paired t-test at 95 percent significance level against 10NN on benchmark datasets are listed at the bottom. The win/tie/loss counts against 10NN are summarized and the method with the smallest number of losses is bolded.

DATA SETS	LABELS	10NN	SLP				SLP-LEAD
			4NN GRAPH	6NN GRAPH	8NN GRAPH	10NNGRAPH	
MNIST3VS8 (69830 INSTANCES)	0.1%	81.34 ± 1.85	31.35 ± 1.71	78.80 ± 1.44	86.39 ± 1.67	88.10 ± 1.69	83.32 ± 1.29
	0.2%	86.92 ± 1.51	40.05 ± 1.10	80.45 ± 1.27	87.22 ± 1.33	89.13 ± 1.39	87.08 ± 0.88
	0.4%	90.29 ± 0.77	48.48 ± 0.69	82.10 ± 0.78	88.32 ± 0.73	90.39 ± 0.71	90.01 ± 0.41
	0.8%	92.72 ± 0.45	55.00 ± 0.53	83.27 ± 0.48	89.04 ± 0.49	91.08 ± 0.44	92.35 ± 0.35
	1.6%	94.39 ± 0.37	60.61 ± 0.39	84.45 ± 0.29	89.88 ± 0.25	91.94 ± 0.30	94.08 ± 0.21
	3.2%	95.68 ± 0.17	65.23 ± 0.44	85.41 ± 0.20	90.53 ± 0.20	92.51 ± 0.20	95.42 ± 0.13
MNIST4VS9 (68910 INSTANCES)	0.1%	70.79 ± 2.65	29.69 ± 0.84	73.03 ± 1.78	80.40 ± 2.27	82.44 ± 2.15	73.35 ± 1.77
	0.2%	75.50 ± 2.76	39.05 ± 0.91	76.66 ± 1.23	83.43 ± 1.31	85.27 ± 1.40	78.06 ± 1.50
	0.4%	82.26 ± 1.26	48.07 ± 0.88	79.47 ± 0.66	85.82 ± 0.54	87.82 ± 0.55	83.28 ± 0.76
	0.8%	86.88 ± 1.04	54.98 ± 0.63	81.54 ± 0.37	87.44 ± 0.45	89.39 ± 0.40	87.47 ± 0.58
	1.6%	92.45 ± 0.39	60.71 ± 0.52	83.10 ± 0.34	88.57 ± 0.31	90.51 ± 0.26	90.97 ± 0.28
	3.2%	93.52 ± 0.32	65.32 ± 0.29	84.51 ± 0.23	89.51 ± 0.21	91.51 ± 0.20	93.30 ± 0.18
MNIST7VS9 (71255 INSTANCES)	0.1%	76.14 ± 3.57	28.52 ± 1.14	76.17 ± 1.40	84.74 ± 1.74	86.74 ± 1.48	80.92 ± 1.90
	0.2%	82.05 ± 1.78	38.97 ± 1.22	79.64 ± 1.15	87.29 ± 1.15	89.22 ± 1.13	85.28 ± 1.05
	0.4%	86.71 ± 1.30	48.25 ± 0.96	82.08 ± 0.69	88.59 ± 0.56	90.36 ± 0.67	89.22 ± 0.73
	0.8%	90.89 ± 0.86	55.96 ± 0.63	83.75 ± 0.42	89.57 ± 0.47	91.39 ± 0.44	91.78 ± 0.41
	1.6%	93.51 ± 0.34	61.93 ± 0.51	85.35 ± 0.31	90.60 ± 0.30	92.42 ± 0.30	94.07 ± 0.27
	3.2%	95.87 ± 0.17	66.88 ± 0.50	86.39 ± 0.26	91.25 ± 0.22	93.04 ± 0.21	95.65 ± 0.11
DEVANAGARI (162000 INSTANCES)	0.1%	90.34 ± 0.85	12.74 ± 0.66	68.70 ± 1.50	91.91 ± 0.83	96.59 ± 0.70	92.76 ± 2.04
	0.2%	91.37 ± 1.03	22.30 ± 0.48	79.66 ± 0.91	94.35 ± 0.63	97.32 ± 0.52	93.31 ± 2.57
	0.4%	93.02 ± 0.58	35.09 ± 0.41	86.45 ± 0.53	95.83 ± 0.29	97.92 ± 0.25	96.15 ± 1.56
	0.8%	94.90 ± 0.68	50.14 ± 0.50	90.82 ± 0.26	96.85 ± 0.19	98.32 ± 0.16	97.70 ± 0.14
	1.6%	96.29 ± 0.43	63.92 ± 0.34	93.53 ± 0.13	97.55 ± 0.08	98.61 ± 0.07	98.49 ± 0.08
	3.2%	97.46 ± 0.30	75.00 ± 0.35	95.46 ± 0.10	98.13 ± 0.07	98.88 ± 0.07	98.76 ± 0.74

[24] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[25] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[26] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[27] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[28] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, "Local higher-order graph clustering," in *Proceedings of the twenty-third ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, Canada, 2017, pp. 555–564.

[29] L. Yann, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[30] B. Mikhail and N. Partha, "Towards a theoretical foundation for laplacian-based manifold methods," *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1289–1308, 2008.

[31] J. Tony, W. Jun, and C. Shih-Fu, "Graph construction and b-matching for semi-supervised learning," in *In Proceedings of the twenty-sixth International Conference on Machine Learning*, Montreal, Canada, 2009, pp. 441–448.

[32] W. Fei and Z. Chang-shui, "Label propagation through linear neighborhoods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp. 55–67, 2008.

[33] X.-J. Zhu, "Semi-supervised learning literature survey," *Computer Science, University of Wisconsin-Madison, 2006, 2(3): 4*, 2006.

[34] C.-P. Miguel Á. and Z. Richard S., "Proximity graphs for clustering and manifold learning," in *Advances in Neural Information Processing Systems*, Cambridge, MA, 2005, pp. 225–232.

[35] W. Fei and Z. Chang-Shui, "Label propagation through linear neighborhoods," in *In Proceedings of the twenty-third International Conference on Machine Learning*, Pittsburgh, PA, 2006, pp. 985–992.

[36] Y.-F. Li, S.-B. Wang, and Z.-H. Zhou, "Graph quality judgement: A large margin expedition," in *Proceedings of the twenty-fifth International Joint Conference on Artificial Intelligence*, New York City, NY, 2016, pp. 1725–1731.

[37] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, Lake Tahoe, NV, 2013, pp. 315–323.

Yu-Feng Li received the BSc and PhD degrees in computer science from Nanjing University, China, in 2006 and 2013, respectively. He joined the Department of Computer Science & Technology at Nanjing University as an Assistant Researcher in 2013, and is currently Associate Professor of the National Key Laboratory for Novel Software Technology. He is a member of the LAMDA group. His research interests are mainly in machine learning. Particularly, he is interested in semi-supervised learning, statistical learning and optimization. He has published over 30 papers in top-tier journal and conferences such as JMLR, TPAMI, AIJ, ICML, NIPS, AAAI, etc. He is/was served as a senior program committee member of top-tier AI conferences such as IJCAI'15, IJCAI'17, AAAI'19, and an editorial board member of machine learning journal special issues. He has received outstanding doctoral dissertation award from China Computer Federation (CCF), outstanding doctoral dissertation award from Jiangsu Province and Microsoft Fellowship Award.

De-Ming Liang is a master student at Department of Computer Science and Technology in Nanjing University, China. He is currently a member of the LAMDA Group. His main research interest is machine learning.