Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Large-Scale Graph-Based Semi-Supervised Learning via Tree Laplacian Solver

Yan-Ming Zhang and **Xu-Yao Zhang**
National Laboratory of Pattern Recognition,
Institute of Automation,
Chinese Academy of Sciences, Beijing, China

Xiao-Tong Yuan
Jiangsu Province Key Laboratory of Big Data
Analysis Technology, Nanjing University of
Information Science and Technology, Nanjing, China

Cheng-Lin Liu
National Laboratory of Pattern Recognition, Institute of Automation,
Chinese Academy of Sciences, Beijing, China
Center of Excellence for Brain Science and Intelligence Technology,
Chinese Academy of Sciences, Beijing, China

Abstract

Graph-based Semi-Supervised learning is one of the most popular and successful semi-supervised learning methods. Typically, it predicts the labels of unlabeled data by minimizing a quadratic objective induced by the graph, which is unfortunately a procedure of polynomial complexity in the sample size n. In this paper, we address this scalability issue by proposing a method that approximately solves the quadratic objective in nearly linear time. The method consists of two steps: it first approximates a graph by a minimum spanning tree, and then solves the tree-induced quadratic objective function in $O(n)$ time which is the main contribution of this work. Extensive experiments show the significant scalability improvement over existing scalable semi-supervised learning methods.

Introduction

Over the past decades, big data phenomenon becomes more and more common as collecting unlabeled data is very easy. On the other hand, labeled data are still precious as labeling data needs expensive human labor. This motivates researchers to develop a new machine learning paradigm, called semi-supervised learning (SSL), that can learn from mixed data sets consist of a limited number of labeled data and a large amount of unlabeled data. Actually, many SSL methods have been proposed over the past decades, such as generative approach (Nigam et al. 2000), co-training (Blum and Mitchell 1998), transductive support vector machines (Joachims 1999), graph-based SSL (Blum and Chawla 2001; Zhu, Ghahramani, and Lafferty 2003; Zhou et al. 2004; Belkin, Niyogi, and Sindhwani 2006).

Among these methods, graph-based SSL is perhaps the most popular one. It is appealing because: (1) graph-based SSL provides a general and effective way for combining labeled and unlabeled data, (2) the resulting optimization problem is convex and admits a closed-form solution. Although having these advantages, graph-based SSL does not scale well with respect to the sample size n. Indeed, Graph-based SSL method has to minimize a quadratic-form objective function induced by the graph Laplacian matrix, which suffers from a time complexity $O(n^3)$ for dense graphs. Although this complexity can be significantly reduced when the graph is sparse, it is still polynomial in the number of nodes, which is intractable for large-scale problems.

To address this scalability issue, various methods have been proposed. In particular, based on the idea of graph sparsification, (Herbster, Pontil, and Galeano 2008; Cesa-Bianchi, Gentile, and Vitale 2009) propose to approximate the graph by a spanning tree and then design fast labeling methods under the online learning setting. Despite the speed, (Zhang, Huang, and Liu 2015) has shown that these methods enjoy the advantage over graph-based SSL of being very robust to the graph construction.

Inspired by these methods, our method also begins with the tree approximation of graph. With a given tree, our method infers labels of unlabeled data by minimizing the same objective function as in traditional graph-based SSL method. The main contribution is that we propose a novel tree Laplacian solver that solves the optimization problem in $O(n)$ time. Although the tree approximation idea has been explored before, both our learning model and the optimization method are totally different. We will review related works in Section 2, and show advantages of our method by extensive comparison experiments in Section 5: for given graphs, our method is at least 10 times faster than state-of-the-art scalable methods.

Related Works

Based on how to approximate the Laplacian matrix, scalable graph-based SSL methods can be divided into two categories: low-rank approximation and sparse approximation.

Low-rank approximation methods can be further divided into two sub-categories. The methods of the first subcategory (Delalleau, Bengio, and Le Roux 2005; Zhang, Kwok, and Parvin 2009; Liu, He, and Chang 2010; Lever, Diethe, and Shawe-Taylor 2012) are based on sampling a

small subset of data points as prototypes which are then used in building the prediction function, the affinity matrix or the data dependent kernel, and the main difference lies in the approaches to construct the affinity matrix. The time complexity of these methods are $O(m^2n)$, where m is the number of prototypes. However, in practice, methods of this type are still slow because: (1) the number of prototypes is proportional to the size of data sets, which means the complexity is not strictly linear with n; (2) the prototype selection stage, e.g. k-means clustering, is time consuming.

The methods of the second sub-category are based on the fast spectral decomposition of Laplacian matrix (Belkin and Niyogi 2004; Fergus, Weiss, and Torralba 2009; Talwalkar et al. 2013). Basically, these methods first extract the low dimensional manifold structure represented by a few eigenvectors of the Laplacian matrix with small eigenvalues, and then train a classifier on this presentation. However, when a matrix is highly sparse (which is the case for most graphs), its spectrum typically decays slowly, which makes the approximation of eigenvectors very inefficient.

The scalable methods of the second class are based on graph sparsification. In (Herbster, Lever, and Pontil 2008; Cesa-Bianchi et al. 2010), authors approximated a graph with a line, and used the nearest neighbor rule to make classificiation. In (Herbster, Pontil, and Galeano 2008), authors proposed to approximate a graph using a spanning tree, and performed the perceptron algorithm to make prediction in the RKHS space induced by this tree Laplacian. In (Cesa-Bianchi, Gentile, and Vitale 2009; Zhang, Huang, and Liu 2015; Vitale et al. 2011), authors also adopted the tree approximation approach, but labeled the tree by minimizing the tree cut.

Preliminaries

Problem Definition

In this work, we consider a particular setting of SSL called transductive learning. Specifically, we are given l labeled data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$ and $n-l$ unlabeled data points $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_n$, where $\mathbf{x}_i \in \mathbb{R}^d$ $(1 \le i \le n)$ is the input of a data point, $y_i \in \{-1, +1\}$ $(1 \le i \le l)$ indicates the label of $\mathbf{x}_i$. For convenience, we also give a fake label to each unlabel data: $y_i = 0$ $(l+1 \le i \le n)$, and thus have an input label vector $\mathbf{y} \in \mathbb{R}^n$. The goal is to predict the real labels of unlabeled data.

For the multiple-class problem, we simply transfer it to multiple binary-class problems by the one-vs-all strategy.

Graph Representation

To apply the graph-based SSL method, one has to represent the data set by an undirected graph, in which each node represents a data point and each weighted edge evaluates the similarity between the connected nodes. Mathematically, a graph can be expressed by an affinity matrix $W \in \mathbb{R}^{n \times n}$, in which $W_{ij} = W_{ji} \ge 0$ is the weight of the edge between node i and j. Furthermore, we can define a graph Laplacian matrix by $L_G \triangleq D - W$, where $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} W_{ij}$.

Graph construction is very crucial for the success of graph-based SSL, and is still an active field (Jebara, Wang, and Chang 2009; Daitch, Kelner, and Spielman 2009; Zhang et al. 2014). However, in practice, k-nearest neighbor (kNN) graph and ϵ-nearest neighbor (ϵNN) graph are most popular.

Graph-based Semi-Supervised Learning

Given a partially labeled graph, graph-based SSL infers the labels of unlabeled nodes by solving the following problem:

$$\min_{\mathbf{f} \in \mathbb{R}^n} (\mathbf{f} - \mathbf{y})' C (\mathbf{f} - \mathbf{y}) + \mathbf{f}' L_G \mathbf{f}, \tag{1}$$

where $\mathbf{y}$ and L_G are defined as above, and C is a constant diagonal matrix with $C_{ii} \ge 0$. The first term evaluates the fitness of $\mathbf{f}$ to the input labels $\mathbf{y}$, while the second term measures the smoothness of $\mathbf{f}$ w.r.t. the graph.

Many classical SSL methods can be viewed as special cases of this method. For example, Gaussian Random Fields (Zhu, Ghahramani, and Lafferty 2003) defines C by $C_{ii} = +\infty$ for $(1 \le i \le l)$ and $C_{ii} = 0$ for $(l+1 \le i \le n)$. Learning Local and Global Consistency (Zhou et al. 2004) uses a normalized graph Laplacian, and defines $C_{ii} = c$ for all i ($c > 0$ is a constant).

Since the objective function is convex w.r.t. $\mathbf{f}$, a global optimal solution exists. In addition, Problem (1) can be solved by setting the gradient of the objective function to zero, which leads to the following linear system:

$$(C + L_G)\mathbf{f} = C\mathbf{y}. \tag{2}$$

Traditionally, it is solved by either two types of methods (Golub and Loan 1996): direct methods based on matrix factorization and iterative methods such as conjugate gradient decent (CG). In general, direct methods are preferred when the problem is small and dense, while CG is more widely used for large sparse problems. The complexity of CG is $O(m\sqrt{\kappa})$ (Shewchuk 1994), where m is the number of edges and κ is the condition number of $C + L_G$. Since κ is typically very large for large sparse matrices, in practice, its running time is polynomial in n. Preconditioning techniques have been applied to CG for improving κ and thus accelerate CG (Spielman and Teng 2006). We will compare our method with both CG and preconditioned CG in the experimental part.

Tree-based Semi-Supervised Learning

Overview

In this section, we propose a method that approximately solves Problem (1) in nearly linear time with the number of edges.

Our motivation is that most graphs contain a large number of redundant edges which can be safely removed without sacrificing the accuracy. Based on this graph sparsification idea, our method first approximates the graph by a spanning tree and builds a Laplacian matrix L_T over the tree; second, it infers the labels by solving the following problem:

$$\begin{aligned} & \min_{\mathbf{f} \in \mathbb{R}^n} (\mathbf{f} - \mathbf{y})' C (\mathbf{f} - \mathbf{y}) + \mathbf{f}' L_T \mathbf{f} \\ \Leftrightarrow \quad & (C + L_T)\mathbf{f} = C\mathbf{y}, \end{aligned} \tag{3}$$

where L_T is the tree Laplacian matrix, $\mathbf{y}$ and C are defined as before. In the following subsections, we first review methods for building spanning trees, and then present the tree Laplacian solver that solves Problem (3) in $O(n)$ time, which is the core contribution of this work.

Generating Spanning Trees

There are a variety of methods for generating spanning trees for connected graphs. However, for the task of SSL, researchers have observed that minimum spanning tree (MST) tends to generate the best performance (Zhang, Huang, and Liu 2015). Theoretically, MST is the solution of $\min\{\sum_{(i,j)\in E(T)} -W_{ij} : T \in \mathcal{T}(G)\}$, where $\mathcal{T}(G)$ is the set of all spanning trees of graph G and $E(T)$ is the edge set of tree T. Also, MST best approximates the original graph in term of the trace norm (Herbster, Pontil, and Galeano 2008). Computationally, MST can be generated by Kruskal's algorithm in $O(|E(G)|\log n)$ time, where $|E(G)|$ is the number of edges in G.

In case that G is not connected, we generate one spanning tree for each connected component, and perform the tree Laplacian solver on each component.

Tree Laplacian Solver

In this section, we propose an efficient method for solving $(C + L_T)\mathbf{f} = C\mathbf{y}$. In essence, it is a direct method (Toledo 2005; Golub and Loan 1996)which consists of two steps: it first factorizes $T \triangleq (C + L_T)$ by computing a sparse matrix S such that $T = S'S$, and then solves $S'S\mathbf{f} = C\mathbf{y}$ by solving two simple linear systems.

Traditional direct methods adopt Cholesky decomposition $T = LL'$ to produce a lower triangular factor matrix. Unfortunately, L may be very dense even for a sparse T which leads to polynomial complexity. Unlike classic direct methods, our method generates a non-triangular factor matrix by directly exploiting the structure of trees. Most importantly, completed in several tree traversals, our method enjoys a complexity of $O(n)$.

Before detailing the method, we first summarize some properties of T as below:

- Since both C and L_T are symmetric and positive semi-definite (PSD), T is symmetric and PSD.
- Since a n-node tree has $n-1$ edges, T has exactly $2(n-1)$ strictly negative off-diagonal elements.
- Since each node has at least one edge, each row/column in T contains at least one negative off-diagonal element.

Ordering the nodes We select an arbitrary node as the root, and sort the nodes by their orders in a post-order traversal of the tree, denoted by $[i_1, i_2, \ldots, i_n]$. Thus, we can perform a bottom-up traversal by visiting nodes from i_1 to i_n, and a top-down traversal from i_n to i_1.

Tree Laplacian Factorization We describe a two-step tree Laplacian factorization method that computes S such that $T = S'S$. By exploiting the tree structure, it can be completed by one bottom-up traversal and one top-down traversal to the tree, and thus has a complexity of $O(n)$.

STEP 1: $P'TP = \Lambda$

We seek a matrix P to diagonalize T as $P'TP = \Lambda$, which is done by a procedure that iteratively eliminates non-zero off-diagonal elements in T from bottom to top. Setting $T^{(1)} = T$, we update $T^{(t)}$ by: $T^{(t+1)} = {P^{(t)}}'T^{(t)}P^{(t)}$ for $t = 1, \ldots, n-1$. $P^{(t)}$ is a matrix with only $n+1$ non-zero elements: all its diagonal elements equal to 1, and $P^{(t)}_{i_t\uparrow(i_t)} = -\frac{T^{(t)}_{i_t\uparrow(i_t)}}{T^{(t)}_{i_t i_t}}$. $\uparrow(a)$ denotes the farther node of a.

It is easy to show by induction that: (1) $T^{(t)}$ is symmetric and (2) the i_t row of $T^{(t)}$ has exactly two non-zero elements: $T^{(t)}_{i_t\uparrow(i_t)}$ and $T^{(t)}_{i_t i_t}$. Thus, $T^{(t+1)}$ is different from $T^{(t)}$ by only three elements: $T^{(t+1)}_{\uparrow(i_t)\uparrow(i_t)} = T^{(t)}_{\uparrow(i_t)\uparrow(i_t)} + P^{(t)}_{i_t\uparrow(i_t)}T^{(t)}_{i_t\uparrow(i_t)}$ and $T^{(t+1)}_{i_t\uparrow(i_t)} = T^{(t+1)}_{\uparrow(i_t)i_t} = 0$. Since $T^{(1)}$ has $2(n-1)$ non-zero off-diagonal elements and we delete two of them by each iteration, $\Lambda \triangleq T^{(n)} = {P^{(n-1)}}' \ldots {P^{(2)}}'{P^{(1)}}'TP^{(1)}P^{(2)}\ldots P^{(n-1)}$ is a diagonal matrix. We define P as $P \triangleq P^{(1)}P^{(2)}\ldots P^{(n-1)}$.

In each iteration, we only have to calculate and store two scalers: $P^{(t)}_{i_t\uparrow(i_t)}$ and $T^{(t+1)}_{\uparrow(i_t)\uparrow(i_t)}$. Thus, the time and space complexity of this procedure is $O(n)$.

STEP 2: $S'S = T$

Defining S as $S \triangleq \Lambda^{\frac{1}{2}}P^{-1} = \Lambda^{\frac{1}{2}}(P^{(n-1)})^{-1}\ldots(P^{(2)})^{-1}(P^{(1)})^{-1}$, it is easy to show that $T = S'S$. To compute S, we let $S^{(1)} = \Lambda^{\frac{1}{2}}$ and iteratively update $S^{(t)}$ by: $S^{(t+1)} = S^{(t)}(P^{(n-t)})^{-1}$ for $t = 1, \ldots, n-1$.

Recall that $P^{(t)}$ is a sparse matrix with all its diagonal elements equal to 1 and $P^{(t)}_{i_t\uparrow(i_t)} = -\frac{T^{(t)}_{i_t\uparrow(i_t)}}{T^{(t)}_{i_t i_t}}$. It is straightforward to show that $(P^{(t)})^{-1}$ is equal to $P^{(t)}$ except one element: $(P^{(t)})^{-1}_{i_t\uparrow(i_t)} = -P^{(t)}_{i_t\uparrow(i_t)}$.

We can show by induction that the i_{n-t} column of $S^{(t)}$ has only one non-zero element: $S^{(t)}_{i_{n-t}i_{n-t}}$. As a result, $S^{(t+1)}$ is different from $S^{(t)}$ by one element: $S^{(t+1)}_{i_{n-t}\uparrow(i_{n-t})} = -S^{(t)}_{i_{n-t}i_{n-t}}P^{(n-t)}_{i_{n-t}\uparrow(i_{n-t})}$ and $S^{(t)}_{i_{n-t}\uparrow(i_{n-t})} = 0$. Since $S^{(1)}$ is diagonal and we add one non-zero off-diagonal elements by one iteration, S has $n-1$ off-diagonal elements.

Since in each iteration we calculate and store one scalar $S^{(t+1)}_{i_{n-t}\uparrow(i_{n-t})}$, the time and space complexity of this procedure is $O(n)$.

Solving $S'S\mathbf{f} = C\mathbf{y}$ Denoting $\mathbf{b} \triangleq C\mathbf{y}$ and $\mathbf{g} \triangleq S\mathbf{f}$, we solve the linear equations system by: (1) solving $S'\mathbf{g} = \mathbf{b}$ by one bottom-up traversal; (2) solving $S\mathbf{f} = \mathbf{g}$ by one top-down traversal. From the previous analysis, S has the following properties:

- For $\forall i \neq i_n$, the i-th row has two non-zero elements: S_{ii} and $S_{i\uparrow(i)}$. The i_n-th row has only one non-zero element: $S_{i_n i_n}$.
- Let $\downarrow(a)$ denote the set of all children of node a. For $\forall i$ that $|\downarrow(i)| \neq 0$, the i-th column has $|\downarrow(i)|+1$ non-zero

Algorithm 1 Tree Laplacian Factorization

Input: $T, [i_1, i_2, \ldots, i_n]$
Output: S such that $S'S = T$
for $t = 1$ **to** $n-1$ **do**
  $P^{(t)}_{i_t\uparrow(i_t)} \leftarrow -\frac{T_{i_t\uparrow(i_t)}}{T_{i_t i_t}}$
  $T_{\uparrow(i_t)\uparrow(i_t)} \leftarrow T_{\uparrow(i_t)\uparrow(i_t)} + P^{(t)}_{i_t\uparrow(i_t)} T_{i_t\uparrow(i_t)}$
  $T_{i_t\uparrow(i_t)} \leftarrow 0$
  $T_{\uparrow(i_t)i_t} \leftarrow 0$
end for
$S \leftarrow T^{\frac{1}{2}}$
for $t = n-1$ **to** 1 **do**
  $S_{i_t\uparrow(i_t)} \leftarrow -S_{i_t i_t} P^{(t)}_{i_t\uparrow(i_t)}$
end for

elements: S_{ii} and $\{S_{ji}, j \in \downarrow(i)\}$. For $\forall i$ that $|\downarrow(i)| = 0$, the i-th column has only one non-zero element: S_{ii}.

STEP 1: $S'\mathbf{g} = \mathbf{b}$

For $\forall i$ that $|\downarrow(i)| = 0$, we have

$$S_{ii} * \mathbf{g}_i = \mathbf{b}_i \quad \Rightarrow \quad \mathbf{g}_i = \frac{\mathbf{b}_i}{S_{ii}}.$$

For $\forall i$ that $|\downarrow(i)| \neq 0$, we have

$$S_{ii} * \mathbf{g}_i + \sum_{j \in \downarrow(i)} S_{ji} * \mathbf{g}_j = \mathbf{b}_i$$

$$\Rightarrow \quad \mathbf{g}_i = \frac{1}{S_{ii}}(\mathbf{b}_i - \sum_{j \in \downarrow(i)} S_{ji} * \mathbf{g}_j).$$

Since computing $\mathbf{g}_i$ only needs to know $\{\mathbf{g}_j, j \in \downarrow(i)\}$, we can solve equations $S'\mathbf{g} = \mathbf{b}$ in order of $[i_1, i_2, \ldots, i_n]$. Since solving each equation needs $O(|\downarrow(i)|)$ operations, solving $S'\mathbf{g} = \mathbf{b}$ needs $O(\sum_i |\downarrow(i)|) = O(n)$ operations.

STEP 2: $S\mathbf{f} = \mathbf{g}$

For $i = i_n$, we have

$$S_{ii} * \mathbf{f}_i = \mathbf{g}_i \quad \Rightarrow \quad \mathbf{f}_i = \frac{\mathbf{g}_i}{S_{ii}}.$$

For $i \neq i_n$, we have

$$S_{ii} * \mathbf{f}_i + S_{i\uparrow(i)}\mathbf{f}_{\uparrow(i)} = \mathbf{g}_i \;\Rightarrow\; \mathbf{f}_i = \frac{1}{S_{ii}}(\mathbf{g}_i - S_{i\uparrow(i)}\mathbf{f}_{\uparrow(i)}).$$

Since computing $\mathbf{f}_i$ only needs to know $\mathbf{f}_{\uparrow(i)}$, we can solve equations in $S\mathbf{f} = \mathbf{g}$ in order of $[i_n, i_{n-1}, \ldots, i_1]$. Since solving each equation needs $O(1)$ operations, solving $S\mathbf{f} = \mathbf{g}$ needs $O(n)$ operations.

To conclude this part, we summarize the tree Laplacian factorization procedure in Alg. 1 and the linear equations system solver in Alg. 2. Note that the time and space complexity for both procedures is $O(n)$.

Algorithm 2 Linear Equations System Solver

Input: S, $\mathbf{b}$, $[i_1, i_2, \ldots, i_n]$
Output: $\mathbf{f}$ such that $S'S\mathbf{f} = \mathbf{b}$
for $t = 1$ **to** n **do**
  if $|\downarrow(i_t)| = 0$ **then**
    $\mathbf{g}_{i_t} \leftarrow \frac{\mathbf{b}_{i_t}}{S_{i_t i_t}}$ //leaf
  else
    $\mathbf{g}_{i_t} \leftarrow \frac{1}{S_{i_t i_t}}(\mathbf{b}_{i_t} - \sum_{j \in \downarrow(i_t)} S_{j i_t} * \mathbf{g}_j)$
  end if
end for
for $t = n$ **to** 1 **do**
  if $|\uparrow(i_t)| = 0$ **then**
    $\mathbf{f}_{i_t} \leftarrow \frac{\mathbf{g}_{i_t}}{S_{i_t i_t}}$ //root
  else
    $\mathbf{f}_{i_t} \leftarrow \frac{1}{S_{i_t i_t}}(\mathbf{g}_{i_t} - S_{i_t\uparrow(i_t)}\mathbf{f}_{\uparrow(i_t)})$
  end if
end for

Convergence and Complexity Analysis

Given a graph, our method first approximates it by a MST tree, and then solves $T\mathbf{f} = C\mathbf{y}$ by the tree Laplacian solver. The tree Laplacian solver completes in exact five times tree traversals: one for node ordering, two for matrix factorization and two for solving the linear systems. Thus, the convergence is clear. Since building MST requires $O(|E(G)| \log n)$ time and the tree Laplacian solver requires $O(n)$ time, its overall complexity is $O(|E(G)| \log n)$. When $O(|E(G)|) = O(n)$ as many applications in practice, our method scales nearly linear with n.

Experiment

Here we conduct experiments to evaluate our method in accuracy and speed. We denote our method as TbTL (Tree-based Transductive Learning), and compare it with two graph-based SSL methods GRF (Zhu, Ghahramani, and Lafferty 2003) and LLGC (Zhou et al. 2004), one tree-based method GPA (Herbster, Pontil, and Galeano 2008), one prototype-based method AGR (Liu, He, and Chang 2010), and one spectral method Eigen (Belkin and Niyogi 2004). The codes of TbTL are available at *www.nlpr.ia.ac.cn/pal/ymzhang/index.html* .

For GRF, conjugate gradient (CG) is applied. For LLGC, we adopt both CG and preconditioned CG whose preconditioners are constructed by spectral sparsifiers of graphs (Koutis, Miller, and Peng 2010). We denote this later method by LLGC-PCG, and use the algorithm CMG from *www.cs.cmu.edu/~jkoutis/cmg.html*. For TbTL, we define C as $C_{ii} = 100$ for $1 \leq i \leq l$, $C_{ii} = 0$ for $l+1 \leq i \leq n$. MST is used for both GPA and TbTL. For Eigen, we use a random algorithm described in (Halko, Martinsson, and Tropp 2011) to compute the m smallest eigenvectors of $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ as it performs much better than column sampling and Nyström which are more suitable for approximating dense matrix. All methods are run on a PC with a 3.10 GHz 4-core CPU and 8 GB RAM.

Small Data Sets

We use 4 image data sets (COIL20, USPS, MNIST and Letter) and 2 text data sets (RCV1 and News20) to test these methods. Their basic properties are listed in Table 1. We adopt kNN for graph construction. For image sets, $d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|^2$ is used for the NN

search and $\exp\{-\frac{d(\mathbf{u},\mathbf{v})}{\sigma_0^2}\}$ for weighting edges where $\sigma_0^2 = \sum_{(i,j)\in E(G)} d(\mathbf{x}_i,\mathbf{x}_j)/|E(G)|$. For text sets, $1-\frac{<\mathbf{u},\mathbf{v}>}{\|\mathbf{u}\|\|\mathbf{v}\|}$ is used for the NN search and $\frac{<\mathbf{u},\mathbf{v}>}{\|\mathbf{u}\|\|\mathbf{v}\|}$ for weighting edges.

Table 1: A brief description of data set.

Data Set	Size	Dim	Class
COIL20	1440	1024	20
USPS	9298	256	10
Letter	20000	16	26
MNSIT	70000	784	10
RCV1	20242	47236	2
News20	19996	1355191	2

Accuracy For GRF, LLGC and Eigen, k is chosen from $\{2,4,8,16\}$ by minimizing the test error. For GPA and TbTL, k is simply fixed to 16 since they are very robust to this parameter. For Eigen, we approximately compute the 400 smallest eigenvectors. For AGR, 1000 prototypes are selected by k-means clustering, and s is chosen from $\{2,4,8,16\}$ by minimizing the test error.

For each data set, we randomly select l data points as labeled data and run different methods to predict the unlabeled data. To control the variance, we repeat the procedure 20 times for different labeled/unlabeled splits. We let l vary from $n\times\{1\%,2\%,4\%,8\%,16\%,32\%,64\%\}$, and report the results in Figure 1. Since accuracies of LLGC and LLGC-PCG are very similar, only results of LLGC are reported.

We observe that: (1) In most cases, GRF and LLGC provide higher accuracies than scalable methods, which implies information lost during the approximation; (2) Two tree-based methods (TbTL and GPA) generate similar accuracy results, while are similar to or better than the prototype-based method AGR and the spectral method Eigen. It suggests that sparse approximation is more effective than low-rank approximation.

Speed We compare methods' running time for labeling a given graph. For GPA and TbTL, the reported time includes the time for generating MST tree and labeling the nodes. As AGR can not be directly applied to graphs, we do not provide its result.

The first experiment examines how the running time of the considered methods vary with n, it proceeds as follows. We construct a 8NN graph for each data set, randomly select $l = 2\%\times n$ data points as labeled data and use different methods to predict the unlabeled data. The average running time over 10 random labeled/unlabeled splits is reported in Table 2. As we can see, TbTL is at least 10 times faster than other methods for all cases, and the advantage becomes greater as n increases.

The second experiment evaluates how the running time vary with $|E(G)|$. We construct a kNN graph for each $k\in\{2,3,\ldots,12\}$ for MNIST. For each graph, we randomly select $l = 2\%\times n$ data points as labeled data and use different methods to predict the unlabeled data. The average running time over 10 random labeled/unlabeled splits is reported in Figure 2. We observe that the running time of GPA and TbTL grows slowly with $|E(G)|$, while the running time of GRF and LLGC grows rapidly.

Table 2: Average running time for labeling a given graph (in seconds).

	USPS	Letter	MNIST	RCV1	News20
GRF	0.16	1.48	73.65	6.39	5.37
LLGC	0.60	0.82	114.26	6.76	13.34
LLGC-PCG	0.49	2.11	4.62	0.38	0.39
GPA	0.28	6.92	14.29	0.45	0.45
Eigen	1.34	2.93	10.71	2.83	2.79
TbTL	**0.01**	**0.05**	**0.15**	**0.03**	**0.03**

The third experiment evaluates how the running time vary with l. We construct a 8NN graph for MNIST, randomly select $l = n\times\{1\%,2\%,4\%,8\%,16\%,32\%,64\%\}$ data points as labeled data and use different methods to predict the unlabeled data. The average running time over 10 random labeled/unlabeled splits is reported in Figure 3. We observe that the running time of LLGC, Eigen and TbTL is invariant with l. The running time of GPA grows linearly with l which is consistent with its time complexity $O(nl)$. The running time of GRF decrease rapidly with l since its complexity is polynomial with $n-l$.

Large Data Set

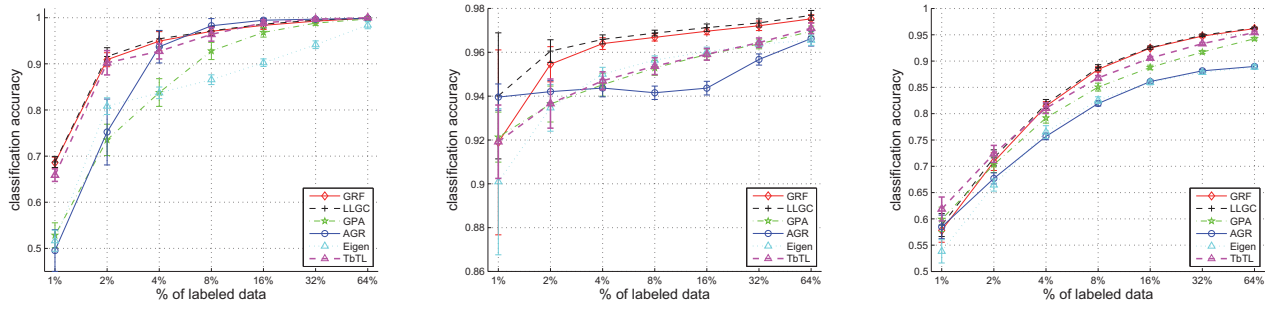
To evaluate our method on large data set, we follow the strategy used in (Liu, He, and Chang 2010). We construct extended MNIST by shifting each image by one pixel in each direction, which results in $630,000$ images. Then, PCA is performed to reduce the data to 86 dimensions.

For GPA, Eigen and TbTL, we perform KD-tree to approximately construct a 8NN graph and adopt $\exp\{-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{\sigma_0^2}\}$ for weighting edges. Since AGR reported out-of-memory error in the experiment, we directly cite the results reported by the authors. For Eigen, the 200 smallest eigenvectors are used to train the least square classifier. We additionally compare with 1NN classifier and SVM with RBF kernel as baseline methods. We list the average accuracy over 20 random labeled/unlabeled splits in Table 3, and the average running time in Table 4. For AGR, Eigen and TbTL, the running time consists of two parts: graph construction which takes 61.3 seconds and graph labeling.

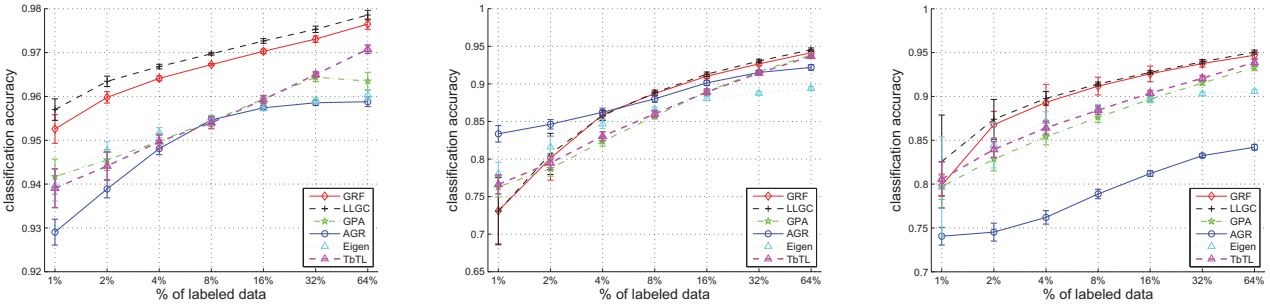
Table 3: Average classification accuracy on extended MNIST with $n = 630,000$(%).

	$l=100$	$l=1000$	$l=10000$
1NN	60.70± 1.98	82.50± 0.42	92.94±0.08
SVM+RBF	57.80± 4.51	88.41±0.26	95.79±0.06
AGR	80.25±1.83	–	–
GPA	89.63±2.13	95.09±0.32	95.78±0.86
Eigen	82.27±2.83	93.96±0.39	94.91±0.08
LLGC-PCG	**90.66±3.34**	**96.28±0.26**	**96.92±0.05**
TbTL	89.97±1.90	95.11±0.32	96.77±0.08

Clearly, TbTL performs better than baseline methods, AGR, GPA and Eigen. Although the accuracy of AGR and



(a) COIL20 (b) USPS (c) Letter



(d) MNIST (e) RCV1 (f) News20

Figure 1: Average classification accuracy for different labeled data size.

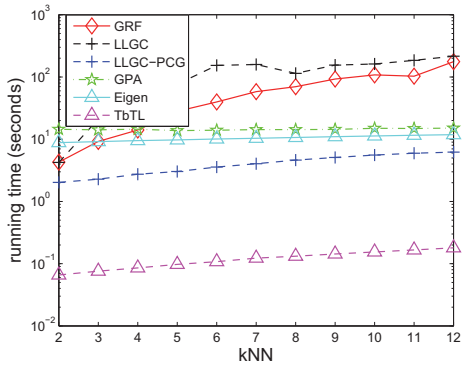


Figure 2: The dependence of running time on the number of edges on MNIST (in seconds).

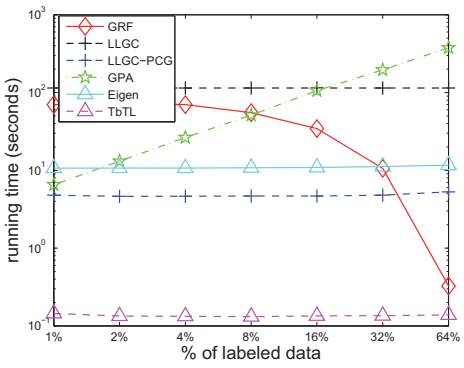


Figure 3: The dependence of running time on the number of labeled nodes on MNIST (in seconds).

Eigen can be improved by using more prototypes or eigenvectors, they will become even more slower since their time complexity scales quadratically with the number of prototypes or eigenvectors.

We also observe that LLGC-PCG's accuracy is slightly better than TbTL. However, the running time of LLGC-PCG is 50+ times of TbTL's for $l = 100$, and 130+ times of TbTL's for $l = 10000$. Thus, we conclude that for small or middle size data sets, LLGC-PCG is better because of its high accuracy; while for very large data sets, TbTL is the algorithm of choice because of its low computational cost.

Conclusion

In this work, we have proposed TbTL as a scalable approach to address the computational challenge faced by the classic graph-based SSL method. Based on the idea of graph sparsification, our method first approximates the graph by a spanning tree. Then, it labels the tree by minimizing a quadratic objective function, for which we propose a novel tree Laplacian solver that finds the solution in $O(n)$ time. Comparing with existing scalable SSL methods, our method is significantly faster while the gain of speedup comes at very little cost of accuracy.

Table 4: Average running time on extended MNIST with $n = 630,000$(in seconds). For GPA, Eigen, LLGC-PCG and TbTL, this contains two parts: the time for building the ANN graph (61.3s) and the time for inferring labels.

	$l = 100$	$l = 1000$	$l = 10000$
1NN	1.7	13.5	116.1
SVM+RBF	12.0	82.1	605.3
AGR	331.7	–	–
GPA	61.3+14.7	61.3+91.5	61.3+1814.6
Eigen	61.3+180.3	61.3+180.8	61.3+183.0
LLGC-PCG	61.3+106.8	61.3+182.2	61.3+265.2
TbTL	**61.3+2.2**	**61.3+2.2**	**61.3+2.2**

As experiments show, the running time of scalable SSL is dominated by computing nearest neighbors. Thus, efficient graph construction methods is the key to further accelerate graph-based SSL, which will be left as the future work.

Acknowledgements

The authors thank all reviewers for their valuable suggestions. This work is supported by the National Natural Science Foundation of China under Grant 61203296, 61375039 and 61403380, and Natural Science Foundation of Jiangsu Province of China under Grant BK20141003.

References

Belkin, M., and Niyogi, P. 2004. Semi-supervised learning on Riemannian manifolds. *Machine Learning* 56(1):209–239.

Belkin, M.; Niyogi, P.; and Sindhwani, V. 2006. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research* 7:2399–2434.

Blum, A., and Chawla, S. 2001. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the International Conference on Machine Learning*.

Blum, A., and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the Conference on Learning Theory*.

Cesa-Bianchi, N.; Gentile, C.; Vitale, I.; and Zappella, G. 2010. Random spanning trees and the prediction of weighted graphs. In *Proceedings of the International Conference on Machine Learning*.

Cesa-Bianchi, N.; Gentile, C.; and Vitale, F. 2009. Fast and optimal prediction of a labeled tree. In *Proceedings of the Conference on Learning Theory*.

Daitch, S.; Kelner, J.; and Spielman, D. 2009. Fitting a graph to vector data. In *Proceedings of the International Conference on Machine Learning*.

Delalleau, O.; Bengio, Y.; and Le Roux, N. 2005. Efficient non-parametric function induction in semi-supervised learning. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*.

Fergus, R.; Weiss, Y.; and Torralba, A. 2009. Semi-supervised learning in gigantic image collections. In *Advances in Neural Information Processing Systems*.

Golub, G. G., and Loan, C. F. V. 1996. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition.

Halko, N.; Martinsson, P. G.; and Tropp, J. A. 2011. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53(2):217–288.

Herbster, M.; Lever, G.; and Pontil, M. 2008. Online prediction on large diameter graph. In *Advances in Neural Information Processing Systems*.

Herbster, M.; Pontil, M.; and Galeano, S. R. 2008. Fast predciton on a tree. In *Advances in Neural Information Processing Systems*.

Jebara, T.; Wang, J.; and Chang, S. 2009. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*.

Joachims, T. 1999. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning*.

Koutis, I.; Miller, G.; and Peng, R. 2010. Approaching optimality for solving SDD systems. In *IEEE Symposium on Foundations of Computer Science*.

Lever, G.; Diethe, T.; and Shawe-Taylor, J. 2012. Data dependent kernels in nearly-linear time. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Liu, W.; He, J.; and Chang, S. 2010. Large graph construction for scalable semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*.

Nigam, K.; McCallum, A.; Thrun, S.; and Mitchell, T. 2000. Text classification from labeled and unlabeled documents using EM. *Machine learning* 39(2):103–134.

Shewchuk, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Technical report.

Spielman, D., and Teng, S. 2006. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *Arxiv preprint cs/0607105*.

Talwalkar, A.; Kumar, S.; Mohri, M.; and Rowley, H. 2013. Large-scale SVD and manifold learning. *The Journal of Machine Learning Research* 14(1):3129–3152.

Toledo, S. 2005. Computing the cholesky factorization of sparse matrices. Technical report, Tel Aviv University.

Vitale, F.; Cesa-Bianchi, N.; Gentile, C.; and Zappella, G. 2011. See the tree through the lines: the shazoo algorithm. In *Advances in Neural Information Processing Systems*.

Zhang, Y.; Huang, K.; Hou, X.; and Liu, C. 2014. Learning locality preserving graph from data. *IEEE Transactions on Cybernetics* 44(11):2088–2098.

Zhang, Y.; Huang, K.; and Liu, C. 2015. MTC: A fast and robust graph-based transductive learning method. *IEEE Transactions on Neural Networks and Learning Systems* 26(9):1979–1991.

Zhang, K.; Kwok, J.; and Parvin, B. 2009. Prototype vector machine for large scale semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*.

Zhou, D.; Bousquet, O.; Lal, T.; Weston, J.; and Scholkopf, B. 2004. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*.

Zhu, X.; Ghahramani, Z.; and Lafferty, J. 2003. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning*.