Robust Label Propagation on Multiple Networks

Tsuyoshi Kato
Center for Informational Biology, Ochanomizu University
2-1-1, Otsuka, Bunkyo, Tokyo 112-8610, Japan
kato-tsuyoshi@aist.go.jp

Hisashi Kashima
IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan
hkashima@jp.ibm.com

Masashi Sugiyama
Department of Computer Science, Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku, Tokyo, 152-8552, Japan
sugi@cs.titech.ac.jp

Abstract

Transductive inference on graphs such as label propagation algorithms is receiving a lot of attention. In this paper, we address a label propagation problem on multiple networks and present a new algorithm that automatically integrates structure information brought in by multiple networks. The proposed method is robust in that irrelevant networks are automatically deemphasized, which is an advantage over Tsuda et al.'s approach [1]. We also show that the proposed algorithm can be interpreted as an EM algorithm with a Student-t prior. Finally, we demonstrate the usefulness of our method in protein function prediction and digit classification, and show analytically and experimentally that our algorithm is much more efficient than existing algorithms.

Keywords

label propagation, multiple networks, EM algorithm

1 Introduction

Recently, network-structured data is becoming increasingly popular in the field of machine learning [2, 3, 1, 4, 5, 6, 7]. Network-structured data is usually represented as an undirected graph, each of whose nodes represents an example, and each of whose edges represents a relationship between two examples. For example, in protein networks, proteins

are represented as nodes, and relationships among proteins such as physical interactions and expression similarities are represented as edges. If we want to predict the functions of the proteins in a network, the task is essentially formalized as a classification problem on a network. Since we are usually given both labeled examples and unlabeled examples prior to the training stage, the task can be handled as a semi-supervised problem or a transduction problem ([8, 2, 9, 3, 10, 1]). Learning in such a setting is called *graph-based learning* (Section 2).

A generally accepted approach to graph-based learning is *label propagation* [11, 3, 12, 13] (Section 3). Label propagation assumes that a pair of nodes connected by an edge should have similar predictions, and the resultant optimization problem is easily solved in a closed form.

Let us consider the situations where we are given multiple data sources, or multiple networks. In the case of protein function prediction, a wide variety of data sources are available, such as gene expression data, amino acid sequences, phylogenetic profiles, and subcellular locations. For label propagation, we have multiple networks corresponding to those multiple data sources. Since different data sources are likely to contain different information, we expect that effective integration of the complementary pieces of information will enhance the predictive performance.

To integrate multiple networks, a natural choice is to take a weighted sum of the graph Laplacians [5]. Tsuda et al. [1] proposed a method which determines the weights automatically. However, as expected from the characteristics of support vector machines (SVMs) [14, 15, 16], their algorithm assigns large weights to networks *irrelevant* to the task, where a network is *irrelevant* if it is not concordant with the classification result of the scores of nodes in a network. Therefore, Tsuda et al.'s method is not robust in a situation where noisy networks are included in the set of given networks (See also Section 6).

In Section 4, we present a new robust transductive learning method that makes predictions by integrating different networks. Similar to Tsuda et al.'s method, our algorithm optimizes the weights in a linear combination of graph Laplacians. The advantage of our algorithm is that the weights are chosen so that informative data sources for prediction are automaticaly emphasized even in the presence of irrelevant networks. Section 5 justifies our algorithm by using a probabilistic framework for emphasizing the weights of relevant networks. Our probabilistic model employs the Student-t distribution [17], which provides robust predictions. The probabilistic model based on the Student-t distribution can be interpreted as a latent variable model in which the expectations of the latent variables are naturally derived as the weights of the graph Laplacians. Accordingly, our algorithm automatically yields an integrated network with a statistically estimated linear combination. Our algorithm is very fast. We show the fact analytically in Section 7 and experimentally in Section 6. For evaluation of the prediction performance, we conduct experiments on label propagation for protein function classification. The experimental results reveal that the proposed method is promising in Section 7. The final section summarizes this study and discusses future work.

Notation: In this paper, vectors are denoted by bold-faced lower-case letters and matrices by bold-faced upper-case letters. Elements of vectors and matrices are not bold-faced. The transposition of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^\top$, and the inverse of $\boldsymbol{A}$ is by $\boldsymbol{A}^{-1}$. The $n \times n$ identity matrix is denoted by $\boldsymbol{I}_n$. The n-dimensional vector all of whose elements are one is denoted by $\mathbf{1}_n$. We use $\mathbb{R}$ to denote the set of real numbers, $\mathbb{R}^n$ to denote the set of n-dimensional real vectors, and $\mathbb{R}^{m\times n}$ to denote the set of $m \times n$ real matrices. The set of real nonnegative numbers is denoted by $\mathbb{R}_+$, and the set of n-dimensional real nonnegative vectors is by $\mathbb{R}^n_+$. We use $\mathbb{S}^n$ to denote the set of symmetric $n \times n$ matrices, $\mathbb{S}^n_+$ to denote the set of symmetric positive semi-definite $n \times n$ matrices, and $\mathbb{S}^n_{++}$ to denote the set of symmetric strictly positive definite $n \times n$ matrices. The symbols $\leq$ and $\geq$ are used to denote not only the standard inequalities between scalars, but also the componentwise inequalities between vectors.

2 Graph-based Learning Problem

Let us start with the definition of the graph-based learning problem, including two problem settings, *learning from a single network* and *learning from multiple networks*. Suppose that we are given n examples: the first $\ell(< n)$ examples are labeled by $y_1, \ldots, y_\ell$, where $y_i \in \{\pm 1\}$. The remaining $n-\ell$ examples are unlabeled and we wish to predict the labels of these unlabeled examples. For *learning from a single network*, we use an undirected network over n nodes. Each node represents an example. The network is described by a normalized symmetric adjacency matrix $\boldsymbol{A} \in \mathbb{S}^n$ satisfying $\boldsymbol{A}\mathbf{1}_n = \mathbf{1}_n$. Every edge has a positive weight and no edge is a self-loop (i.e. $A_{ii} = 0$ for $\forall i$). The edge set is expressed as

$$\mathcal{E} \equiv \{(i,j) \,|\, A_{ij} > 0,\ i,j = 1,\ldots,n\}.$$

In this paper, we wish to perform *learning from multiple networks*. If the number of networks is K, we have K adjacency matrices corresponding to the K different networks, denoted by $\boldsymbol{A}_k$ $(k = 1,\ldots,K)$. Our setting is often said to be transductive. Typical transductive setting has a large number of unlabeled examples and a small portion of examples are labeled. However, we assume many examples are labeled as usual learning settings assume. The assumption is needed for robust label propagation, as detailed later.

Graph-based learning determines the score vector $\boldsymbol{f} \in \mathbb{R}^n$ from the given labels and the link information ($\boldsymbol{A}$ in the case of a single network, and $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_K$ in the case of multiple networks). For calculating the score vector $\boldsymbol{f} \in \mathbb{R}^n$, a typical formulation is the regularized least-squares problem.

Networks are usually constructed in two stages [2]. First, we compute the distances among all the pairs of examples, and then determine the edges based on the distances. From the distances, the edge set of the network is determined by finding the k-nearest neighbors or by picking pairs with distances smaller than a threshold.

3 Existing Label Propagation Algorithm with Single Network

In this section, we review a graph-based learning approach [3] for a single network. The task here is to determine the score vector $\boldsymbol{f} \in \mathbb{R}^n$ from the link information $\boldsymbol{A}$ and the labels $\boldsymbol{y}$. For calculating the score vector $\boldsymbol{f} \in \mathbb{R}^n$, the typical algorithm we consider here solves the regularized least-squares problem defined by

$$\hat{\boldsymbol{f}} \equiv \underset{\boldsymbol{f}}{\operatorname{argmin}} \left(\beta_{\mathrm{y}} \sum_{i=1}^{\ell} (y_i - f_i)^2 + \beta_{\mathrm{bias}} \sum_{i=1}^{n} f_i^2 + \beta_{\mathrm{net}} \sum_{(i,j)\in\mathcal{E}} A_{ij} (f_i - f_j)^2 \right)$$

where β_{y}, β_{bias} and β_{net} are constant. If the score is greater than a threshold, then it is classified as positive, and otherwise it is classified as negative. Using the graph Laplacian defined by

$$\boldsymbol{L} \equiv \mathrm{diag}\,(\boldsymbol{A}\boldsymbol{1}_n) - \boldsymbol{A},$$

the minimization problem can be re-written as

$$\min_{\boldsymbol{f}} \beta_{\mathrm{y}} (\boldsymbol{f} - \boldsymbol{y})^\top \boldsymbol{G} (\boldsymbol{f} - \boldsymbol{y}) + \boldsymbol{f}^\top (\beta_{\mathrm{bias}} \boldsymbol{I}_n + \beta_{\mathrm{net}} \boldsymbol{L}) \boldsymbol{f} \tag{1}$$

where $\boldsymbol{G} \in \mathbb{S}_+^n$ is a diagonal matrix with

$$G_{ii} \equiv \begin{cases} 1 & \text{if } 1 \le i \le \ell, \\ 0 & \text{if } \ell + 1 \le i \le n, \end{cases}$$

i.e. the i-th diagonal element of $\boldsymbol{G}$ is one if the i-th node is labeled, or otherwise zero. Therein, we have defined

$$\boldsymbol{y} \equiv [y_1, \ldots, y_\ell, 0, \ldots, 0]^\top \in \mathbb{R}^n.$$

4 Robust Label Propagation Algorithm

This section presents a new algorithm for robust label propagation on multiple networks. The task is to predict the score vector $\boldsymbol{f} \in \mathbb{R}^n$ from the labels $\boldsymbol{y}$ and the network structures $\{\boldsymbol{A}_k\}_{k=1}^K$. A choice is integration of multiple networks followed by applying the algorithm for a single network described in the previous section. A way for integration of networks is superimposition. In order to emphasize the informative networks, we consider weighted linear combination of K networks. Let $\bar{\boldsymbol{u}} \in \mathbb{R}_+^K$ be the weights. The integrated adjacency matrix and the integrated graph Laplacian are then respectively given by

$$\boldsymbol{A}_{\mathrm{int}}(\bar{\boldsymbol{u}}) = \sum_{k=1}^{K} \bar{u}_k \boldsymbol{A}_k$$

and

$$\boldsymbol{L}_{\mathrm{int}}(\bar{\boldsymbol{u}}) = \sum_{k=1}^{K} \bar{u}_k \boldsymbol{L}_k.$$

Once the weights are determined, we obtain scores in the same way as for the standard label propagation:

$$\begin{aligned}\hat{\boldsymbol{f}} &= \operatorname*{argmin}_{\boldsymbol{f}} \left(\beta_{\mathrm{y}} \left(\boldsymbol{f} - \boldsymbol{y} \right)^{\top} \boldsymbol{G} \left(\boldsymbol{f} - \boldsymbol{y} \right) + \boldsymbol{f}^{\top} \left(\beta_{\mathrm{bias}} \boldsymbol{I}_n + \beta_{\mathrm{net}} \boldsymbol{L}_{\mathrm{int}}(\bar{\boldsymbol{u}}) \right) \boldsymbol{f} \right) \\ &= \left(\boldsymbol{G} + \frac{\beta_{\mathrm{bias}}}{\beta_{\mathrm{y}}} \boldsymbol{I}_n + \frac{\beta_{\mathrm{net}}}{\beta_{\mathrm{y}}} \boldsymbol{L}_{\mathrm{int}}(\bar{\boldsymbol{u}}) \right)^{-1} \boldsymbol{G}\boldsymbol{y}. \end{aligned} \tag{2}$$

How do we compute the weights? We determine the weights iteratively with the following update rule using the current $\boldsymbol{f}$:

$$\bar{u}_k = \frac{\nu + n}{\nu + \beta_{\mathrm{net}} \boldsymbol{f}^{\top} \boldsymbol{L}_k \boldsymbol{f}}, \tag{3}$$

where ν is a positive constant. In Section 5.4, we will show that the update rule is naturally derived from an EM algorithm. Note that

$$\boldsymbol{f}^{\top} \boldsymbol{L}_k \boldsymbol{f} = \sum_{(i,j) \in \mathcal{E}_k} A_{ij}^{(k)} \left(f_i - f_j \right)^2,$$

where $A_{ij}^{(k)}$ is the (i,j)th element in $\boldsymbol{A}_k$. If a node is likely to belong to the same class as an adjacent node, label propagation algorithms work well. In other words, an informative network should have the property that adjacent nodes tend to have similar predictions. For that reason, the value of $\boldsymbol{f}^{\top} \boldsymbol{L}_k \boldsymbol{f}$ is large for a network irrelevant to the task, whereas $\boldsymbol{f}^{\top} \boldsymbol{L}_k \boldsymbol{f}$ is small for a relevant network. Since the term $\boldsymbol{f}^{\top} \boldsymbol{L}_k \boldsymbol{f}$ is in the denominator of Eq. (3), the weights of relevant networks become large and the weights of irrelevant networks become small. Our algorithm is summarized as:

1: Set the initial weights $\bar{\boldsymbol{u}} = \boldsymbol{1}_K / K$.
2: **repeat**
3: Update the scores $\boldsymbol{f}$ using Eq. (2).
4: Update the weights $\bar{\boldsymbol{u}}$ using Eq. (3).
5: **until** convergence.

5 Probabilistic Interpretation

In this section, we give a probabilistic interpretation of our algorithm. We begin by presenting a probabilistic model for label propagation with a single network, and show that the MAP estimation coincides with the solution of the label propagation approach. We next extend the probabilistic model to the case of integration of multiple networks

with fixed weights. We then introduce a prior distribution of the weights. Finally we derive an EM algorithm for MAP estimation according to that probabilistic model, and show the equivalence between the EM algorithm and the iterative algorithm described in Section 4.

5.1 Label Progapagation with Single Network

We here give a probabilistic interpretation of label propagation with a single network. The label propagation method can be viewed as performing MAP estimation of the score vector $\boldsymbol{f}$ in the probabilistic model described below. The score vector $\boldsymbol{f} \in \mathbb{R}^n$ is in the set of model parameters. The observations $y_1, \ldots, y_\ell$ are drawn according to the Gaussian distribution

$$p(y_i|\boldsymbol{f}) = \mathcal{N}\left(y_i\,;\, f_i, \frac{1}{\beta_{\mathrm{y}}}\right), \tag{4}$$

where $\mathcal{N}(\boldsymbol{x}\,;\,\boldsymbol{m},\boldsymbol{S})$ is a Gaussian density function of the observation $\boldsymbol{x} \in \mathbb{R}^n$ with mean $\boldsymbol{m} \in \mathbb{R}^n$ and covariance $\boldsymbol{S} \in \mathbb{S}^n_{++}$ defined by

$$\mathcal{N}(\boldsymbol{x}\,;\,\boldsymbol{m},\boldsymbol{S}) \equiv \frac{1}{(2\pi)^{n/2}|\boldsymbol{S}|^{1/2}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x}-\boldsymbol{m}\right)^\top \boldsymbol{S}^{-1}\left(\boldsymbol{x}-\boldsymbol{m}\right)\right). \tag{5}$$

The prior distribution of the model parameters is defined by the multivariate Gaussian distribution

$$p(\boldsymbol{f}) = \mathcal{N}\left(\boldsymbol{f}\,;\,\boldsymbol{0}_n,\ (\beta_{\mathrm{bias}}\boldsymbol{I}_n + \beta_{\mathrm{net}}\boldsymbol{L})^{-1}\right).$$

Zhu et al. [11] also use a similar prior distribution. MAP estimation finds the value of the model parameters $\boldsymbol{f}$ which maximizes the posterior probability

$$p(\boldsymbol{f}|\boldsymbol{y}) = \frac{p(\boldsymbol{f})\prod_{i=1}^{\ell} p(y_i|\boldsymbol{f})}{p(\boldsymbol{y})}. \tag{6}$$

Since the denominator of Eq. (6) is constant for maximization, MAP estimation is equivalent to maximizing the following objective function

$$\log p(\boldsymbol{f}) + \sum_{i=1}^{\ell} \log p(y_i|f_i) = -\frac{1}{2}\left(\beta_{\mathrm{y}}(\boldsymbol{f}-\boldsymbol{y})^\top \boldsymbol{G}(\boldsymbol{f}-\boldsymbol{y}) + \boldsymbol{f}^\top\left(\beta_{\mathrm{bias}}\boldsymbol{I}_n + \beta_{\mathrm{net}}\boldsymbol{L}\right)\boldsymbol{f}\right) + \mathrm{const.}$$

We can see that the values of $\boldsymbol{f}$ at the maximum of the posterior probability function are equal to the solution of Eq. (1). Thus the probabilistic interpretation is established.

5.2 Label Progapagation with Fixed-Weight Network Integration

We next extend the probabilistic model introduced above to the integration of multiple networks with fixed weight $\bar{\boldsymbol{u}} \in \mathbb{R}^K_+$. A probabilistic model associated with this weighted

integration is given by the conditional probabilities in Eq. (4) and the prior

$$p(\boldsymbol{f}) = \mathcal{N}\left(\boldsymbol{f}\,;\,\mathbf{0}_n,\ (\beta_{\text{bias}}\boldsymbol{I}_n + \beta_{\text{net}}\boldsymbol{L}_{\text{int}}(\bar{\boldsymbol{u}}))^{-1}\right). \tag{7}$$

Using a small constant ϵ such that $\epsilon > 0$, we add a small positive value to the diagonal elements of the graph Laplacian

$$\tilde{\boldsymbol{L}}_k \equiv \boldsymbol{L}_k + \epsilon \boldsymbol{I}_n.$$

Since any graph Laplacian is positive semi-definite, the matrices $\tilde{\boldsymbol{L}}_k$ are strictly positive definite. Exploiting this feature, we change the prior as follows:

$$p(\boldsymbol{f}) = \frac{1}{Z'}\mathcal{N}\left(\boldsymbol{f}\,;\,\mathbf{0}_n,\ \frac{1}{\beta_{\text{bias}}}\boldsymbol{I}_n\right)\prod_{k=1}^{K}\mathcal{N}\left(\boldsymbol{f}\,;\,\mathbf{0}_n,\ \frac{1}{\beta_{\text{net}}\bar{u}_k}\tilde{\boldsymbol{L}}_k^{-1}\right), \tag{8}$$

where Z' is a normalizing constant. Taking the limit $\epsilon \to 0$, we can confirm Eq. (8) is reduced to Eq. (7) (See Appendix A.). Thus we see that the prior of the model parameters for the fixed weight integration of multiple networks is expressed as the product-of-Gaussians [18]. This rearrangement facilitates development of the probabilistic version of our robust label propagation algorithm on multiple networks.

5.3 Prior Distribution over Network Weights

We have seen the probabilistic model for label propagation with given weights. Here let us consider the situation where the weights are unknown. We introduce a prior distribution of the weights and marginalize out the random variables of the weights from the expressions. The weights are marginalized in the probabilistic model described below, but finally obtained as the expected values defined later in Eq. (16). We employ the Gamma distribution for the prior of the weights. The Gamma distribution is defined by

$$\text{Gamma}(u;\alpha,\beta) = \frac{\beta^\alpha}{\Gamma(\alpha)}u^{\alpha-1}\exp(-\beta u)$$

for $u \geq 0$, $\alpha \geq 0$, $\beta \geq 0$. In the probabilistic model described here, each component of a network in Eq. (8),

$$\mathcal{N}\left(\boldsymbol{f}\,;\,\mathbf{0}_n,\ \frac{1}{\beta_{\text{net}}\bar{u}_k}\tilde{\boldsymbol{L}}_k^{-1}\right),$$

is replaced with an infinite mixture of Gaussians:

$$\int_0^\infty du_k\,\text{Gamma}\left(u_k;\frac{1}{2}\nu,\frac{1}{2}\nu\right)\mathcal{N}\left(\boldsymbol{f}\,;\,\mathbf{0}_n,\ \frac{1}{\beta_{\text{net}}}\left(u_k\tilde{\boldsymbol{L}}_k\right)^{-1}\right) \tag{9}$$

where ν is a positive constant. In this model, the mixture coefficients are expressed by the Gamma distribution, and the weights $\boldsymbol{u} \equiv [u_1,\ldots,u_K]^\top$ can be viewed as latent variables. If ν is chosen to be smaller, the prior of the weights is flatter.

Our probabilistic model can be seen to employ a robust prior for the score vector. The function in Eq. (9) is equal to the Student-t density function [17] with mean zero, covariance $\beta_{\text{net}}^{-1}\tilde{\boldsymbol{L}}_k^{-1}$, and degree-of-freedom ν, where the density function of the Student-t distribution is generally defined by

$$\mathcal{T}(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma},\nu) = \frac{\Gamma\left(\frac{\nu+n}{2}\right)\sqrt{\det\boldsymbol{\Sigma}}}{\left(\sqrt{\pi\nu}\right)^n\Gamma\left(\frac{\nu}{2}\right)\left(\sqrt{1+\|\boldsymbol{x}-\boldsymbol{\mu}\|^2_{\boldsymbol{\Sigma}^{-1}}/\nu}\right)^{\nu+n}},$$

with mean $\boldsymbol{\mu}\in\mathbb{R}^n$, covariance $\boldsymbol{\Sigma}\in\mathbb{S}^n_{++}$, and degree-of-freedom $\nu\in\mathbb{R}_+$. The Student-t distribution has a heavier tail than Gaussian. As degree-of-freedom ν increases, the Student-t distribution approaches Gaussian. Thus, the prior can be viewed as the product of Student-t distributions [19] given by

$$p(\boldsymbol{f}) = \frac{1}{Z}\mathcal{N}\left(\boldsymbol{f}\,;\,\boldsymbol{0}_n,\ \beta_{\text{bias}}^{-1}\boldsymbol{I}_n\right)\prod_{k=1}^{K}\mathcal{T}\left(\boldsymbol{f}\,;\,\boldsymbol{0}_n,\,\beta_{\text{net}}^{-1}\tilde{\boldsymbol{L}}_k^{-1},\nu\right) \tag{10}$$

where Z is a normalizing constant. Student-t distributions are often used for modeling noisy data robustly [20], since heavy-tailed distributions such as Student-t distributions are robust against outliers. The algorithm described below offers a robust tool for label propagation, implicitly exploiting the heavy-tailed property of Student-t distributions.

5.4 EM Algorithm for MAP Estimation

We devised an EM algorithm for MAP estimation of $\boldsymbol{f}$ according to the model with the prior in Eq. (10). We here describe the EM algorithm and show that the EM algorithm is reduced to the iterative algorithm presented in Section 4. Given that the constants ν, β_{y}, β_{bias} and β_{net} are determined in advance, MAP estimation finds the model parameters which maximize the objective function:

$$J_{\text{lp}}(\boldsymbol{f}) \equiv \log p(\boldsymbol{f}) + \sum_{i=1}^{\ell}\log p(y_i|f_i). \tag{11}$$

From Eqs. (9) and (10), the logarithm of the prior is re-written as

$$\log p(\boldsymbol{f}) = \log Z + \log\mathcal{N}\left(\boldsymbol{f}\,;\,\boldsymbol{0}_n,\ \beta_{\text{bias}}^{-1}\boldsymbol{I}_n\right) + \sum_{k=1}^{K}\log\int_0^\infty du_k\, h_k(\boldsymbol{f},u_k), \tag{12}$$

where Z is a normalizing constant, and the function $h_k(\cdot,\cdot)$ is defined by

$$h_k(\boldsymbol{f},u_k) \equiv \text{Gamma}\left(u_k;\frac{1}{2}\nu,\frac{1}{2}\nu\right)\mathcal{N}\left(\boldsymbol{f}\,;\,\boldsymbol{0}_n,\frac{1}{u_k\beta_{\text{net}}}\tilde{\boldsymbol{L}}_k^{-1}\right). \tag{13}$$

We introduce an arbitrary distribution $q(u_k)$ such that

$$q(u_k)\geq 0 \text{ and } \int_0^\infty du_k\, q(u_k) = 1.$$

Using Jensen's inequality, each component in the second term in Eq. (12) is bounded from below as

$$\begin{aligned}\log \int_0^\infty du_k\, h_k(\boldsymbol{f}, u_k) &= \log \int_0^\infty du_k\, q(u_k) \frac{h_k(\boldsymbol{f}, u_k)}{q(u_k)} \\ &\geq \int_0^\infty du_k\, q(u_k) \log \frac{h_k(\boldsymbol{f}, u_k)}{q(u_k)} \\ &= \int_0^\infty du_k\, q(u_k) \log h_k(\boldsymbol{f}, u_k) + \mathcal{H}[q(u_k)], \end{aligned} \tag{14}$$

where $\mathcal{H}[q(u_k)]$ denotes the entropy of the distribution $q(u_k)$. The inequality holds with equality when the density function $q(u_k)$ maximizes the lower-bound [21]. Specifically,

$$\log \int_0^\infty du_k\, h_k(\boldsymbol{f}, u_k) = \int_0^\infty du_k\, \hat{q}(u_k) \log h_k(\boldsymbol{f}, u_k) + \mathcal{H}[\hat{q}(u_k)], \tag{15}$$

where $\hat{q}(u_k)$ is the optimal distribution. Our EM algorithm attempts to maximize the lower-bound in Eq. (14). The EM algorithm consists in E-step and M-step: E-step computes the optimal distribution $\hat{q}(u_k)$, and M-step maximizes the logarithm of the posterior probability with respect to the model parameters $\boldsymbol{f}$. Note that Eq. (15) implies that the logarithm of the posterior probability is equal to the lower-bound in M-step. Variational analysis derives the optimal distribution $\hat{q}(u_k)$ to be computed in E-step as

$$\begin{aligned}\log \hat{q}(u_k) &= \log h_k(\boldsymbol{f}, u_k) + \text{const.} \\ &= -\frac{u_k \beta_{\text{net}}}{2} \boldsymbol{f}^\top \tilde{\boldsymbol{L}}_k \boldsymbol{f} + \frac{n}{2} \log u_k + \left(\frac{\nu}{2} - 1\right) \log u_k - \frac{\nu}{2} u_k + \text{const.} \\ &= \log \text{Gamma}\left(u_k\,;\, \frac{\nu + n}{2}\,,\, \frac{\nu}{2} + \frac{\beta_{\text{net}}}{2} \boldsymbol{f}^\top \tilde{\boldsymbol{L}}_k \boldsymbol{f}\right), \end{aligned}$$

where 'const' denotes the terms independent of u_k (See Appendix B for derivation.). Let us denote the expectation of u_k over $\hat{q}(u_k)$ by

$$\bar{u}_k \equiv \int_0^\infty du_k \hat{q}(u_k) u_k = \frac{\nu + n}{\nu + \beta_{\text{net}} \boldsymbol{f}^\top \tilde{\boldsymbol{L}}_k \boldsymbol{f}}. \tag{16}$$

We now derive the update rule of M-step. With the help of Eqs. (11), (12), (15), and (16), the logarithm of the posterior probability can be re-written as

$$J_{\text{lp}}(\boldsymbol{f}) = -\frac{\beta_{\text{y}}}{2} (\boldsymbol{f} - \boldsymbol{y})^\top \boldsymbol{G} (\boldsymbol{f} - \boldsymbol{y}) - \frac{\beta_{\text{bias}}}{2} \|\boldsymbol{f}\|^2 - \frac{\beta_{\text{net}}}{2} \boldsymbol{f}^\top \left(\sum_{k=1}^K \bar{u}_k \tilde{\boldsymbol{L}}_k\right) \boldsymbol{f} + \text{const.} \tag{17}$$

where 'const' here denotes the terms independent of $\boldsymbol{f}$. From this expression, we can see that $\bar{u}_k$ play the role of the weights for the linear combination of networks. Hence $\bar{u}_k$ will now be called *weights*. The derivative with respect to $\boldsymbol{f}$ is expressed as

$$\frac{\partial J_{\text{lp}}(\boldsymbol{f})}{\partial \boldsymbol{f}} = -\frac{1}{2} \beta_{\text{y}} \boldsymbol{G} (\boldsymbol{f} - \boldsymbol{y}) - \frac{1}{2} \left(\beta_{\text{bias}} \boldsymbol{I}_n + \beta_{\text{net}} \sum_{k=1}^K \bar{u}_k \tilde{\boldsymbol{L}}_k\right) \boldsymbol{f},$$

which leads to representing the update rule for the score vector $\boldsymbol{f}$ as

$$\boldsymbol{f} = \left(\boldsymbol{G} + \frac{\beta_{\mathrm{bias}}}{\beta_{\mathrm{y}}} \boldsymbol{I}_n + \frac{\beta_{\mathrm{net}}}{\beta_{\mathrm{y}}} \sum_{k=1}^{K} \bar{u}_k \tilde{\boldsymbol{L}}_k \right)^{-1} \boldsymbol{G}\boldsymbol{y}. \tag{18}$$

If we take the limit $\epsilon \to 0$, Eqs. (16) and (18) become Eqs. (3) and (2), respectively.

The EM algorithm is then summarized as follows:

E-step: Update $\bar{u}_k$ using Eq. (3) for $k = 1, \ldots, K$.
M-step: Update $\boldsymbol{f}$ using Eq. (2).

The two steps are repeated until convergence. Thus the equivalence between the iterative algorithm given in the previous section and the EM algorithm is established.

EM algorithms are guaranteed to converge to a local optimum [22], so is our algorithm. Currently we just choose the equal weights as the initial point; we may use a multi-point search strategy to further improve the performance, although this increases the computational cost.

6 Related Work

Besides our algorithm, there exist several studies [2, 5, 1] on learning with multiple networks. Tsuda et al. [1] have proposed a label propagation algorithm. The task in their paper is exactly the same as ours: to predict the labels of nodes from multiple networks. We refer to their algorithm as TSS. They pose the following optimization problem:

$$\begin{aligned} \min \quad & (\boldsymbol{f} - \boldsymbol{y})^\top (\boldsymbol{f} - \boldsymbol{y}) + c\gamma + c_0 \|\boldsymbol{\xi}\|_1 \\ \text{wrt.} \quad & \boldsymbol{f} \in \mathbb{R}^n, \quad \gamma \in \mathbb{R}, \quad \boldsymbol{\xi} \in \mathbb{R}_+^K \\ \text{subj. to} \quad & \boldsymbol{f}^\top \boldsymbol{L}_k \boldsymbol{f} \leq \gamma + \xi_k, \qquad k = 1, \ldots, K. \end{aligned}$$

The dual problem is given by

$$\begin{aligned} \min \quad & \boldsymbol{y}^\top \left(\boldsymbol{I}_n + \sum_k \bar{u}_k \boldsymbol{L}_k \right)^{-1} \boldsymbol{y} \\ \text{wrt.} \quad & \bar{\boldsymbol{u}} \in \mathbb{R}_+^K \\ \text{subj. to} \quad & \bar{\boldsymbol{u}} \leq c_0 \mathbf{1}_k, \qquad \bar{\boldsymbol{u}}^\top \mathbf{1}_K \leq c, \end{aligned}$$

where $\bar{u}_k$ is the dual variable corresponding to the inequality $\boldsymbol{f}^\top \boldsymbol{L}_k \boldsymbol{f} \leq \gamma + \xi_k$. Once the optimal values of the dual variables are obtained, the scores are recovered by minimizing

$$(\boldsymbol{f} - \boldsymbol{y})^\top (\boldsymbol{f} - \boldsymbol{y}) + \boldsymbol{f}^\top \boldsymbol{L}_{\mathrm{int}}(\bar{\boldsymbol{u}}) \boldsymbol{f} \tag{19}$$

with respect to the score vector $\boldsymbol{f}$, where

$$\boldsymbol{L}_{\mathrm{int}}(\bar{\boldsymbol{u}}) = \sum_{k=1}^{K} \bar{u}_k \boldsymbol{L}_k.$$

Equation (19) implies that TSS also integrates networks and performs label propagation. The dual variables are the weights of integration. In terms of this point, TSS is similar to our label propagation algorithm. However, TSS does not assume the existence of irrelevant networks. If we used TSS in such a situation, TSS would be confronted with a difficulty. At the optimum, if $\boldsymbol{f}^\top \boldsymbol{L}_k \boldsymbol{f} < \gamma$, we have $\bar{u}_k = 0$, since the optimal $\bar{\boldsymbol{u}}$ satisfies the Karush-Kuhn-Tucker (KKT) condition

$$\bar{u}_k(\boldsymbol{f}^\top \boldsymbol{L}_k \boldsymbol{f} - \gamma - \xi_k) = 0,$$

as described in Tsuda et al.'s paper [1]. If $\boldsymbol{f}^\top \boldsymbol{L}_k \boldsymbol{f} = \gamma$, then ξ_k is zero and the corresponding dual variable can take $0 \leq \bar{u}_k \leq c_0$ because of the similar reason. If $\boldsymbol{f}^\top \boldsymbol{L}_k \boldsymbol{f} > \gamma$, then ξ_k must be positive, and thereby the corresponding dual variable becomes $\bar{u}_k = c_0$. As mentioned in Section 4, the value of $\boldsymbol{f}^\top \boldsymbol{L}_k \boldsymbol{f}$ of irrelevant networks tends to be large. Therefore, TSS gives large weights $\bar{u}_k$ to irrelevant networks and deprives relevant networks of their weights. Hence, TSS cannot be expected to achieve good performance when noisy networks are included.

Argyriou et al. [2] have proposed a kernel-based approach for combining different networks. Their method is based on SDP/SVM [23], which is formulated as a semidefinite programming problem, and which performs training of SVM and optimization of the weights of the kernel matrices simultaneously. First, they convert each graph Laplacian into the Laplacian kernel matrix, and then efficiently solve the optimization problem by exploiting the sparsity of the graph Laplacians. However, their method yields an integrated kernel matrix, not the integrated graph Laplacians. For recovering a network from the integrated kernel matrix, we would consider a reverse procedure which computes the inverse of the integrated kernel matrix. However, the inverse matrix is not sparse anymore, and generally produces a fully connected network. Hence, if one wanted to obtain the predictions as well as the integrated network, their approach could not meet that requirement directly.

Let us discuss the computational complexity of the proposed algorithm. A Laplacian matrix is typically sparse and the number of non-zero elements is $|\mathcal{E}| + n = O(|\mathcal{E}|)$ if the network is connected. According to Spielman and Teng's work [24], our M-step in Eq. (2) can be computed in $O(|\mathcal{E}|)$, while E-step in Eq. (3) takes in time nearly linear in $|\mathcal{E}|$. Thus the total complexity is nearly linear in $|\mathcal{E}|R$ where R is the number of iterations. This is the same order as TSS. Thus our method is as computationally efficient as TSS, while ours is more robust against noisy networks. SDP/SVM takes $O(n^3 R)$ [25] so it is generally much slower than ours. Argyriou et al.'s method becomes faster if the least square loss is chosen and some techniques for sparse matrices are combined [26].

7 Experiments

We first illustrate the utility of our label propagation algorithm with a synthetic dataset. We then show the performance of our algorithm with real biological data and handwritten digits.

7.1 A Synthetic Example

The three artificial networks we used for the demonstration are depicted in Fig. 1(a),(b),(c). Note that they have ten common nodes. We assume that the first five nodes are the positive nodes, and the last five nodes are in the negative class. Nodes 1, 3, 6 and 8 are assumed to be labeled, and the remaining nodes are unlabeled. Most of the edges in $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ connect nodes with the same class labels, whereas the edges in $\boldsymbol{A}_3$ connect nodes with opposite class labels. Hence, $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are relevant to the task, but $\boldsymbol{A}_3$ is irrelevant.

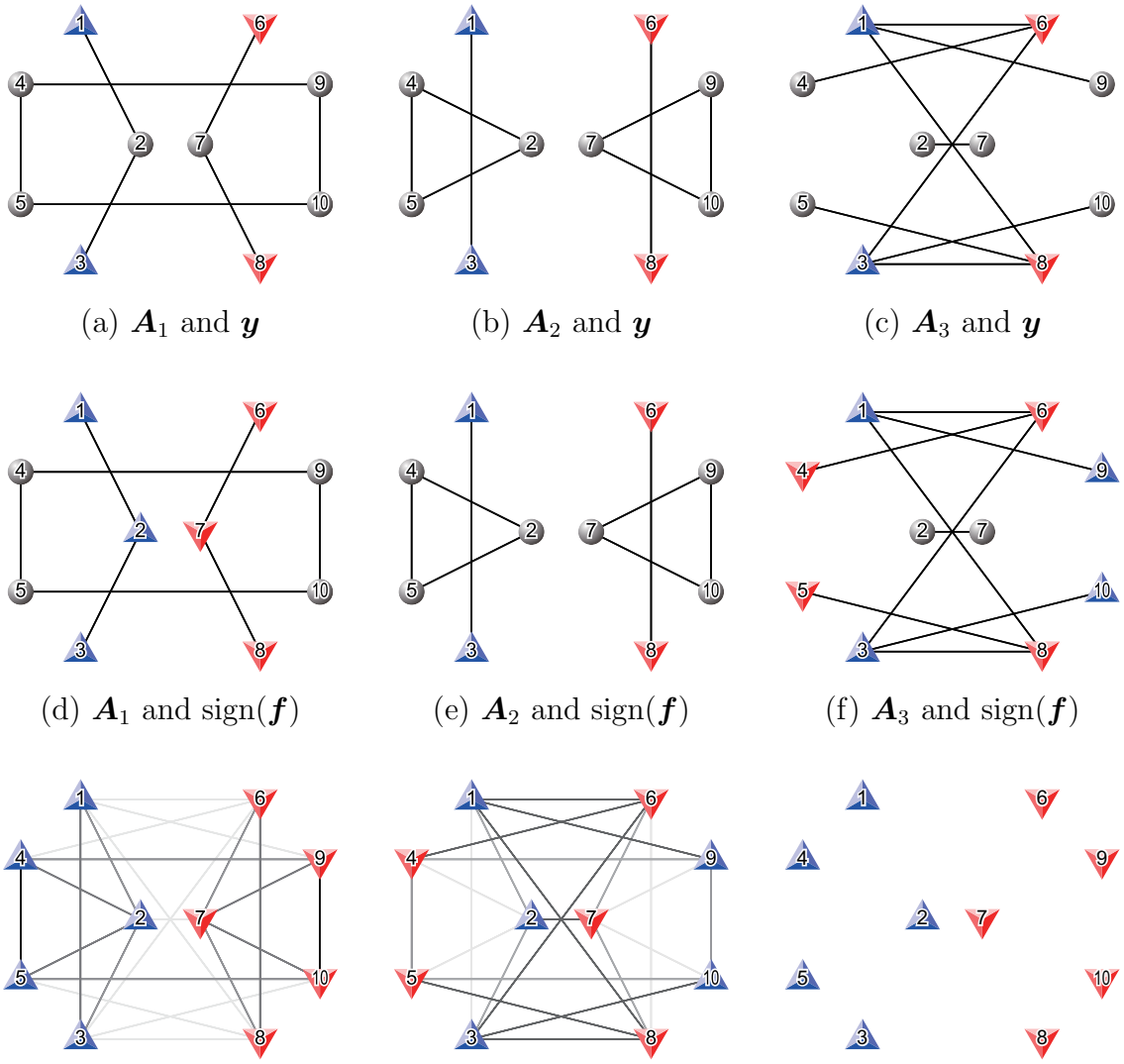
We begin with a demonstration of label propagation individually on a single network. The prediction results are shown in Fig. 1(d),(e),(f). For $\boldsymbol{A}_1$, nodes 2 and 7 are correctly predicted, but nodes 4, 5, 9, 10 can not be predicted because these four have no paths to any labeled nodes. For $\boldsymbol{A}_2$, no unlabeled nodes can be classified because of the same reason. For $\boldsymbol{A}_3$, nodes 4, 5, 9, 10 are classified, but the predictions are wrong due to the irrelevant edges. The results suggest that correct predictions can not be made individually from any single network. We have to at least combine these networks for better predictions.

Now let us see the results of label propagation on the integration of networks. Figure. 1(g) depicts the prediction results of our algorithm. All the nodes are correctly classified. The resultant weights are $\bar{\boldsymbol{u}} = [2.98, 3.78, 0.53]^\top$. The shading of the edges in the figures represents the weights $\bar{u}_k$. The weight of the third network $\bar{u}_3$ is automatically determined to be small. As a result, the labels of all of the nodes are correctly predicted. Figure. 1(h) is the results of TSS. The values of weights produced by TSS were $\bar{\boldsymbol{u}} = [4.00, 2.00, 4.00]^\top$. The algorithm thus assigned a large weight to $\boldsymbol{A}_3$, which caused poor predictions.

7.2 Protein Function Prediction

Next we performed an experiment on protein function prediction [23]. (The data set is available from http://noble.gs.washington.edu/proj/sdp-svm/.) The task is a binary classification problem to predict whether or not each protein is ribosomal. Out of 760 proteins in total, 92 proteins are ribosomal. We used two types of input data consisting of the protein interaction network [27] and the gene expression data [23]. The gene expression data was converted to a 5-nearest-neighbor network[1]. The two networks are denoted by $\boldsymbol{A}_{\mathrm{vm}}$ and $\boldsymbol{A}_{\mathrm{e}}$, respectively. In order to test the robustness to the presence of noise, we added two decoy networks. The decoy network $\boldsymbol{A}_{\mathrm{r1}}$ was made from $\boldsymbol{A}_{\mathrm{vm}}$ by randomly shuffling the node indices. The decoy network $\boldsymbol{A}_{\mathrm{r2}}$ was made in the same way as $\boldsymbol{A}_{\mathrm{e}}$. We tested two cases: in the first case only the two relevant networks $\boldsymbol{A}_{\mathrm{vm}}$ and $\boldsymbol{A}_{\mathrm{e}}$ were used for the primary input, and, in the second case all four networks were used. All the graph Laplacians derived from these networks were normalized so that the diagonal elements were one, and divided by the number of edges. We randomly choose 20% of the nodes as unlabeled nodes. We performed five-fold cross validation over the labeled nodes

[1]We also varied the number of nearest neighbors, but similar results were obtained.



(a) $\boldsymbol{A}_1$ and $\boldsymbol{y}$ (b) $\boldsymbol{A}_2$ and $\boldsymbol{y}$ (c) $\boldsymbol{A}_3$ and $\boldsymbol{y}$

(d) $\boldsymbol{A}_1$ and sign($\boldsymbol{f}$) (e) $\boldsymbol{A}_2$ and sign($\boldsymbol{f}$) (f) $\boldsymbol{A}_3$ and sign($\boldsymbol{f}$)

(g) Proposed (h) TSS (i) True Labels

Figure 1: Here we are given three networks $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ and $\boldsymbol{A}_3$, as depicted in (a),(b),(c) where up-pointing triangles, down-pointing triangles, and circles denote nodes with $y_i = +1$, $y_i = -1$, and unknown label, respectively. The true class labels of nodes are depicted in (i). (d),(e),(f) are the prediction results of label propagation on an individual network. The results of our algorithm and TSS are shown in (g),(h). The shade of the edges represents the network weights.

to choose β_{y}, β_{bias}, β_{net}, and ν for the proposed algorithm. For performance evaluation, we performed ROC analysis while changing the threshold of classification, and computed the ROC scores (i.e. the areas under the ROC curves).

We compared our method with two existing methods, Tsuda et al.'s method (TSS) [1],

Table 1: Protein function prediction using graph-learning algorithms. The left table reports the ROC scores (areas under the ROC curves) and the weights using $\boldsymbol{A}_{\mathrm{vm}}$ and $\boldsymbol{A}_{\mathrm{e}}$. The right table is with the two decoy networks, $\boldsymbol{A}_{\mathrm{r1}}$, $\boldsymbol{A}_{\mathrm{r2}}$ added. The shown weights are normalized so that the sum is one.

Method	ROC score	$\bar{u}_{\mathrm{vm}}$	$\bar{u}_{\mathrm{e}}$
Proposed	1.000	0.377	0.623
TSS	0.999	0.500	0.500
SDP/SVM	0.999	-	-

Method	ROC score	$\bar{u}_{\mathrm{vm}}$	$\bar{u}_{\mathrm{e}}$	$\bar{u}_{\mathrm{r1}}$	$\bar{u}_{\mathrm{r2}}$
Proposed	0.998	0.279	0.408	0.16	0.154
TSS	0.721	0.200	0.000	0.400	0.400
SDP/SVM	0.999	-	-	-	-

and Argyriou et al.'s kernel-based method (SDP/SVM) [2]. The hyper-parameters of TSS and SDP/SVM are also chosen using five-fold cross-validation. Table 1 summarizes the results evaluated with the ROC scores averaged over ten trials. Each row represents the predictive performance as an ROC score and the weights for each of the networks. The weights for networks are denoted by $\bar{u}_{\mathrm{vm}}, \bar{u}_{\mathrm{e}}, \bar{u}_{\mathrm{r1}},$ and $\bar{u}_{\mathrm{r2}}$.

We used the Wilcoxon test for the statistical significance of the difference among the ROC scores. Without decoy networks, no significant differences can be observed among the three methods. Next we statistically tested whether or not each algorithm lost significant performance due to the presence of noisy networks. Adding the decoy networks, the performance of TSS degrades significantly (P-value=0.006, respectively). In particular, TSS assigns large weights to noisy networks, and thereby damages the prediction performance severely. In contrast, no significant performance loss was seen for Proposed and SDP/SVM (P-value=0.100 and 0.181, respectively).

In order to further investigate the robustness of the proposed method, we increased the number of decoy networks. Figure 2 shows the ROC scores of Proposed, TSS, and SDP/SVM methods. TSS performs considerably less well as the number of decoy networks increases. However, Proposed and SDP/SVM successfully maintain their performance.

In summary, the Proposed algorithm achieves comparable classification accuracy to SDP/SVM, yet provides the optimally integrated network $\boldsymbol{L}_{\mathrm{int}}(\bar{\boldsymbol{u}})$.

7.3 Digit Classification

We also tested our algorithm on handwritten digit recoginition. We used 200 images randomly chosen from the MNIST handwritten digit dataset for each digit. We gave class labels to 160 of the 200 images, and the remaining 40 images were unlabeled. We posed a binary classification problem which is to classify the odd digits from the even digits. Therefore, the number of nodes is $n = 2,000$ and the number of labeled nodes is $\ell = 1,600$ for the binary classification task.

The networks we used were constructed as follows. We first computed the k-nearest-neighbor graphs for $k = 1, \ldots, 5$ and obtained $\boldsymbol{A}'_{\mathrm{n1}}, \boldsymbol{A}'_{\mathrm{n2}}, \ldots, \boldsymbol{A}'_{\mathrm{n5}}$. The first network $\boldsymbol{A}'_{\mathrm{n1}}$ is the 1-nearest-neighbor graph, i.e. $\boldsymbol{A}_{\mathrm{n1}} = \boldsymbol{A}'_{\mathrm{n1}}$. The second network $\boldsymbol{A}_{\mathrm{n2}}$ was constructed by removing the edges included in the 1-nearest-neighbor graph from the 2-nearest-neighbor graph, i.e. $\boldsymbol{A}_{\mathrm{n2}} = \boldsymbol{A}'_{\mathrm{n2}} - \boldsymbol{A}'_{\mathrm{n1}}$. Similarly we obtained five networks for

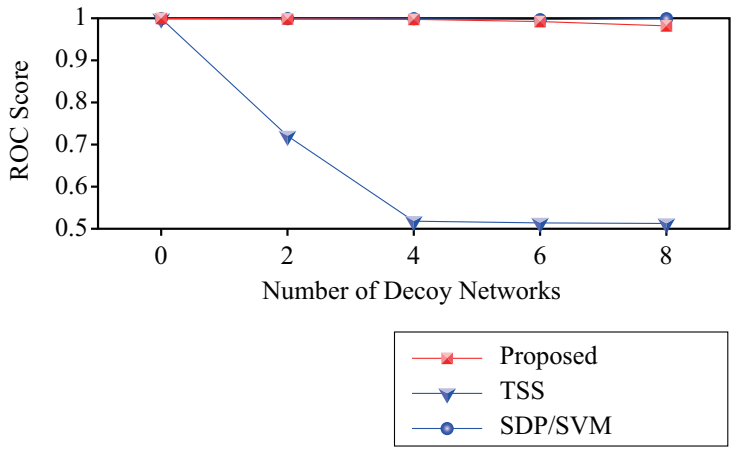


Figure 2: Accuracy of prediction against the number of decoy networks.

Table 2: Digit classification using graph-learning algorithms. The upper table reports the ROC scores (areas under the ROC curves) and the weights using the five networks $\boldsymbol{A}_{\mathrm{n}1},\ldots,\boldsymbol{A}_{\mathrm{n}5}$. The lower table is with the five decoy networks $\boldsymbol{A}_{\mathrm{r}1},\ldots,\boldsymbol{A}_{\mathrm{r}5}$ added. The shown weights are normalized so that the sum is one.

Method	Accuracy	$\bar{u}_{\mathrm{n}1}$	$\bar{u}_{\mathrm{n}2}$	$\bar{u}_{\mathrm{n}3}$	$\bar{u}_{\mathrm{n}4}$	$\bar{u}_{\mathrm{n}5}$
Proposed	0.945	0.249	0.212	0.190	0.181	0.168
TSS	0.900	0.000	0.026	0.279	0.295	0.400
SDP/SVM	0.910	-	-	-	-	-

Method	Accuracy	$\bar{u}_{\mathrm{n}1}$	$\bar{u}_{\mathrm{n}2}$	$\bar{u}_{\mathrm{n}3}$	$\bar{u}_{\mathrm{n}4}$	$\bar{u}_{\mathrm{n}5}$	$\bar{u}_{\mathrm{r}1}$	$\bar{u}_{\mathrm{r}2}$	$\bar{u}_{\mathrm{r}3}$	$\bar{u}_{\mathrm{r}4}$	$\bar{u}_{\mathrm{r}5}$
Proposed	0.944	0.193	0.168	0.152	0.146	0.137	0.038	0.040	0.042	0.042	0.043
TSS	0.493	0.000	0.000	0.000	0.000	0.000	0.400	0.299	0.184	0.064	0.053
SDP/SVM	0.922	-	-	-	-	-	-	-	-	-	-

$\boldsymbol{A}_{\mathrm{n}k} = \boldsymbol{A}'_{\mathrm{n}k} - \boldsymbol{A}'_{\mathrm{n}(k-1)}$. In addition, in order to illustrate the robustness of the proposed method, we added five decoy networks, denoted by $\boldsymbol{A}_{\mathrm{r}1},\ldots,\boldsymbol{A}_{\mathrm{r}5}$. Each decoy network was constructed by shuffling the order of the node indices of one of $\boldsymbol{A}_{\mathrm{n}k}$, $k = 1,\ldots,5$. In total, we obtained ten networks. All of the graph Laplacians derived from these networks were normalized such that the diagonal elements are one. We performed cross validation over the labeled nodes to choose β_{y}, β_{bias}, β_{net}, and ν for the proposed algorithm. For performance evaluation, we computed the classification accuracy on the test set.

Table 2 summarizes the results evaluated with accuracies averaged over five trials. Similar to the results on protein function prediction, TSS assigned large weights to noisy networks, and thereby which degrades the prediction performance of TSS severely (P-value=0.006). In contrast, Proposed and SDP/SVM did not lose performance seriously (P-value=0.833 and 0.114, respectively).

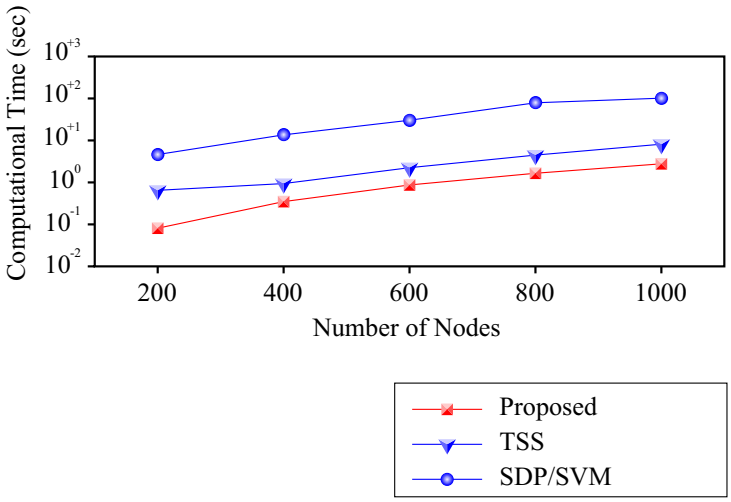


Figure 3: Computational Time.

7.4 Computational Time

We analyzed the running time of our label propagation algorithm. We constructed five networks in the similar way for building $\boldsymbol{A}_{\mathrm{n1}},\ldots,\boldsymbol{A}_{\mathrm{n5}}$ from the digit images. We randomly chose digit images from the MNIST dataset and varied the number of nodes. The average running time on five trials are plotted in Fig. 3. SDP/SVM was about 30 times slower than our algorithm in all cases. Our algorithm was also much faster than TSS. It is because TSS resorts a MATLAB function `fmincon`. The function is an implementation of a gradient-based numerical algorithm which is not known to be very efficient.

7.5 Generalization Performance Against EM Iterations

Finally we investigated the generalization performance against EM iteration of our label propagation algorithm. We used the two protein networks $\boldsymbol{A}_{\mathrm{vm}}$ and $\boldsymbol{A}_{\mathrm{e}}$ and eight decoy networks for this investigation. The results are plotted in Fig. 4. Initially, our implementation starts with equal weights, so the generalization performance was low due to the existence of decoy networks. At the second iteration, the ROC score almost reached the finally obtained ROC score. Therefore, if one stoped the algorithm at second iteration, our algorithm could become even faster and yet achieve a satisfactory generalization performance.

8 Conclusions

This paper proposed using Student-t distributions to perform robust label propagation with different networks including the presence of irrelevant networks. Our algorithm consists of two phases: weighted network integration and label propagation. The network

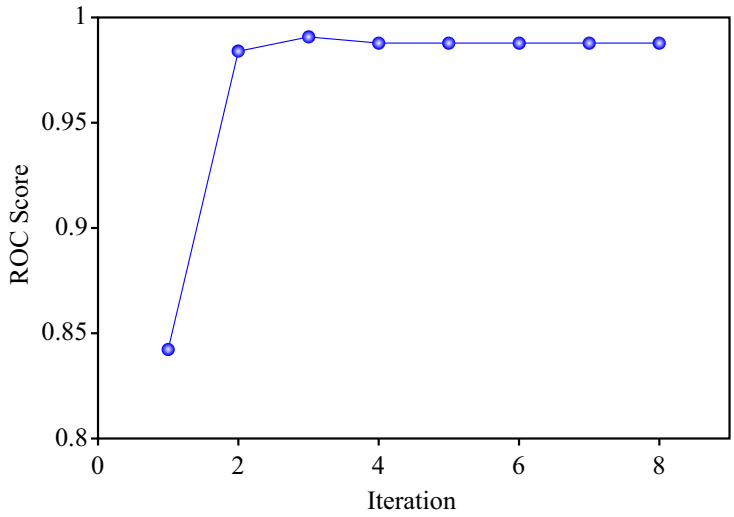


Figure 4: Generalization Performance Against EM Iterations.

integration is done so that the weights reflect how relevant each network is to the task. The label propagation is performed on the integrated network. We described the two operations as the E-step and M-step operations, respectively, of the EM algorithm. Thus our algorithm is intuitively understandable and statistically well supported.

Although our algorithm already has the compelling property of functioning with the integrated networks, we are considering how to modify our algorithm so as to produce a sparse solution. That will be future work for this area.

A Relation between Eqs. (8) and (7)

The rhs of Eq. (8) can be rearranged as

$$
\begin{aligned}
&\frac{1}{Z'}\mathcal{N}\left(\boldsymbol{f}\,;\,\mathbf{0}_n,\ \frac{1}{\beta_{\mathrm{bias}}}\boldsymbol{I}_n\right)\prod_{k=1}^{K}\mathcal{N}\left(\boldsymbol{f}\,;\,\mathbf{0}_n,\ \frac{1}{\beta_{\mathrm{net}}\bar{u}_k}\tilde{\boldsymbol{L}}_k^{-1}\right)\\
&\quad\propto \exp\left(-\frac{1}{2}\boldsymbol{f}^\top\left(\beta_{\mathrm{bias}}\boldsymbol{I}_n\right)\boldsymbol{f}\right)\prod_{k=1}^{K}\exp\left(-\frac{1}{2}\boldsymbol{f}^\top\left(\beta_{\mathrm{net}}\bar{u}_k\tilde{\boldsymbol{L}}_k\right)\boldsymbol{f}\right)\\
&\quad= \exp\left(-\frac{1}{2}\boldsymbol{f}^\top\left(\beta_{\mathrm{bias}}\boldsymbol{I}_n\right)\boldsymbol{f}-\frac{1}{2}\sum_{k=1}^{K}\boldsymbol{f}^\top\left(\beta_{\mathrm{net}}\bar{u}_k\tilde{\boldsymbol{L}}_k\right)\boldsymbol{f}\right)\\
&\quad= \exp\left(-\frac{1}{2}\boldsymbol{f}^\top\left(\beta_{\mathrm{bias}}\boldsymbol{I}_n+\beta_{\mathrm{net}}\sum_{k=1}^{K}\bar{u}_k\tilde{\boldsymbol{L}}_k\right)\boldsymbol{f}\right)\\
&\quad= \exp\left(-\frac{1}{2}\boldsymbol{f}^\top\left(\beta_{\mathrm{bias}}\boldsymbol{I}_n+\beta_{\mathrm{net}}\sum_{k=1}^{K}\bar{u}_k\boldsymbol{L}_k\right)\boldsymbol{f}\right)\exp\left(-\frac{\epsilon}{2}\beta_{\mathrm{net}}\mathbf{1}^\top\bar{\boldsymbol{u}}\|\boldsymbol{f}\|^2\right).
\end{aligned}
$$

Since

$$\lim_{\epsilon \to 0} \exp(-\epsilon c) = 1 \qquad \text{for } 0 \leq \forall c < \infty,$$

we get

$$\begin{aligned}\lim_{\epsilon \to 0} \frac{1}{Z'} \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n,\ \frac{1}{\beta_{\text{bias}}}\boldsymbol{I}_n\right) \prod_{k=1}^{K} \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n,\ \frac{1}{\beta_{\text{net}}\bar{u}_k}\tilde{\boldsymbol{L}}_k^{-1}\right) \\ = \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n,\ (\beta_{\text{bias}}\boldsymbol{I}_n + \beta_{\text{net}}\boldsymbol{L}_{\text{int}}(\bar{\boldsymbol{u}}))^{-1}\right).\end{aligned}$$

Therefore, the two priors in Eqs. (7) and (8) are equivalent at the limit $\epsilon \to 0$.

B Derivation of Eq. (17)

We here derive Eq. (17). Substituting Eq. (12) into Eq. (11), we get

$$\begin{aligned}J_{\text{lp}}(\boldsymbol{f}) = \sum_{i=1}^{\ell} \log p(y_i|f_i) + \log Z + \log \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n,\ \frac{1}{\beta_{\text{bias}}}\boldsymbol{I}_n\right) \\ + \sum_{k=1}^{K} \log \int_0^{\infty} du_k\, h_k(\boldsymbol{f}, u_k)\end{aligned}$$

Equation (15) holds immediately after E-step. Therefore,

$$\begin{aligned}J_{\text{lp}}(\boldsymbol{f}) = \sum_{i=1}^{\ell} \log p(y_i|f_i) + \log Z + \log \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n,\ \frac{1}{\beta_{\text{bias}}}\boldsymbol{I}_n\right) \\ + \sum_{k=1}^{K} \int_0^{\infty} du_k\, \hat{q}(u_k) \log h_k(\boldsymbol{f}, u_k) + \sum_{k=1}^{K} \mathcal{H}[\hat{q}(u_k)].\end{aligned}$$

From the definition of the function $h(\cdot, \cdot)$ in Eq. (13),

$$\begin{aligned}J_{\text{lp}}(\boldsymbol{f}) = \sum_{i=1}^{\ell} \log p(y_i|f_i) + \log Z + \log \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n,\ \frac{1}{\beta_{\text{bias}}}\boldsymbol{I}_n\right) \\ + \sum_{k=1}^{K} \int_0^{\infty} du_k\, \hat{q}(u_k) \log \text{Gamma}\left(u_k; \frac{1}{2}\nu, \frac{1}{2}\nu\right) \\ + \sum_{k=1}^{K} \int_0^{\infty} du_k\, \hat{q}(u_k) \log \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n, \frac{1}{u_k\beta_{\text{net}}}\tilde{\boldsymbol{L}}_k^{-1}\right) + \sum_{k=1}^{K} \mathcal{H}[\hat{q}(u_k)].\end{aligned}$$

The second term, the forth term and the last term does not depend on $\boldsymbol{f}$. If we denote the terms independent on $\boldsymbol{f}$ by 'const', we obtain

$$\begin{aligned}J_{\text{lp}}(\boldsymbol{f}) = \sum_{i=1}^{\ell} \log p(y_i|f_i) + \log \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n,\ \frac{1}{\beta_{\text{bias}}}\boldsymbol{I}_n\right) \\ + \sum_{k=1}^{K} \int_0^{\infty} du_k\, \hat{q}(u_k) \log \mathcal{N}\left(\boldsymbol{f}\,;\, \mathbf{0}_n, \frac{1}{u_k\beta_{\text{net}}}\tilde{\boldsymbol{L}}_k^{-1}\right) + \text{const.}\end{aligned}$$

By substituting the defintion of the multivariate Gaussian in Eq. (5), we obtain

$$\begin{aligned}
J_{\mathrm{lp}}(\boldsymbol{f}) &= -\frac{\beta_{\mathrm{y}}}{2}(\boldsymbol{f}-\boldsymbol{y})^{\top}\boldsymbol{G}(\boldsymbol{f}-\boldsymbol{y}) + \log Z - \frac{\beta_{\mathrm{bias}}}{2}\|\boldsymbol{f}\|^2 \\
&\quad + \sum_{k=1}^{K}\int_0^{\infty} du_k\, \hat{q}(u_k)\left(-\frac{u_k\beta_{\mathrm{net}}}{2}\boldsymbol{f}^{\top}\tilde{\boldsymbol{L}}_k\boldsymbol{f}\right) + \mathrm{const.} \\
&= -\frac{\beta_{\mathrm{y}}}{2}(\boldsymbol{f}-\boldsymbol{y})^{\top}\boldsymbol{G}(\boldsymbol{f}-\boldsymbol{y}) - \frac{\beta_{\mathrm{bias}}}{2}\|\boldsymbol{f}\|^2 + \sum_{k=1}^{K}\left(-\frac{\bar{u}_k\beta_{\mathrm{net}}}{2}\boldsymbol{f}^{\top}\tilde{\boldsymbol{L}}_k\boldsymbol{f}\right) + \mathrm{const.} \\
&= -\frac{\beta_{\mathrm{y}}}{2}\left(\boldsymbol{f}-\boldsymbol{y}\right)^{\top}\boldsymbol{G}\left(\boldsymbol{f}-\boldsymbol{y}\right) - \frac{\beta_{\mathrm{bias}}}{2}\|\boldsymbol{f}\|^2 - \frac{\beta_{\mathrm{net}}}{2}\boldsymbol{f}^{\top}\left(\sum_{k=1}^{K}\bar{u}_k\tilde{\boldsymbol{L}}_k\right)\boldsymbol{f} + \mathrm{const.}
\end{aligned}$$

Equation (17) is thus derived.

Acknowledgment

The quality of our manuscript was improved greatly by the suggestions from four anonymous reviewers. Their advices helped us to notice new advantages of our algorithm and some mistakes of our formulation. We sincerely acknowledge the reviewers. We thank W. Noble and K. Tsuda for making their software and data public. The authors would like to thank T-PRIMAL participants for their valuable comments. In particular, TK appreciates helpful discussions with Ryohei Fujimaki, Kenichi Kurihara, Shinichi Nakajima, Rikiya Takahashi and Koji Tsuda. This work was supported by a Grant-in-Aid for Young Scientists (B), number 18700287, from the Ministry of Education, Culture, Sports, Science and Technology, Japan and BIRD of Japan Science and Technology Agency (JST).

References

[1] K. Tsuda, H. Shin, and B. Schölkopf, "Fast protein classification with multiple networks," *Bioinformatics*, vol. 21, no. 2, pp. i59–i65, 2005.

[2] A. Argyriou, M. Herbster, and M. Pontil, "Combining graph Laplacians for semi–supervised learning," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 67–74.

[3] 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 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[4] H. Shin, J. Hill, and G. Rätsch, "Graph based semi-supervised learning with sharper edges," in *ECML 2006*. Berlin, Germany: Springer-Verlag, 2006, pp. 1008–1015.

[5] T. Zhang, A. Popescul, and B. Dom, "Linear prediction models with graph regularization for web-page categorization," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, Philadelphia, PA, USA, 2006, pp. 821–826.

[6] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.

[7] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete structures," in *Proc. 19th International Conference on Machine Learning (ICML2002)*, C. Sammut and A. G. Hoffmann, Eds. San Francisco, Morgan Kaufmann, 2002, pp. 315–322.

[8] J. Weston, C. Leslie, D. Zhou, A. Elisseeff, and W. . Noble, "Semi-supervised protein classification using cluster kernels," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[9] A. Kapoor, Y. A. Qi, H. Ahn, and R. Picard, "Hyperparameter and kernel learning for graph-based semi-supervised classification," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 627–634.

[10] T. Zhang and R. Ando, "Analysis of spectral kernel design based semi-supervised learning," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 1601–1608.

[11] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions." in *Proceedings of the Twentieth International Conference on Machine Learning*, T. Fawcett and N. Mishra, Eds. San Francisco, CA: AAAI Press, 2003.

[12] O. Bousquet, O. Chapelle, and M. Hein, "Measure based regularization," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[13] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf, "Ranking on data manifolds," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[14] B. Schölkopf and A. J. Smola, *Learning with kernels*. Cambridge, MA: MIT Press, 2002.

[15] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, Sept 1999.

[16] C. C. Chang, C. W. Hsu, and C. J. Lin, "The analysis of decomposition methods for support vector machines," *IEEE Transactions on Neural Networks*, vol. 11, no. 4, pp. 1003–1008, July 2000.

[17] G. J. McLachlan and T. Krishnan, *The EM algorithm and extensions*. John Wiley & Sons, 1997.

[18] F. V. Agakov and C. K. I. Williams, "Products of Gaussians and probabilistic minor component analysis," *Neural Computation*, vol. 14, no. 5, pp. 1169–1182, May 2002.

[19] M. Welling, G. Hinton, and S. Osindero, "Learning sparse topographic representations with products of Student-t distributions," in *Advances in Neural Information Processing Systems 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003, pp. 1359–1366.

[20] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, USA: Springer Science+Business Media, LLC, 2006.

[21] R. M. Neal and G. E. Hinton, *Learning in Graphical Models*. Kluwer Academic Publishers, 1998, ch. A view of the EM algorithm that justifies incremental, sparse, and other variants, pp. 355–368.

[22] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Statistical Society, Series B*, vol. 39, pp. 1–38, 1977.

[23] G. R. G. Lanckriet, T. D. Bie, N. Cristianini, M. Jordan, and W. Noble, "A statistical framework for genomic data fusion," *Bioinformatics*, vol. 20, pp. 2626–2635, 2004.

[24] D. A. Spielman and S. H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *Proceedings of the 26th annual ACM symposium on Theory of computing*. ACM Press, 2004, pp. 81–90.

[25] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf, "Large scale multiple kernel learning," *Journal of Machine Learning Research*, vol. 7, July 2006.

[26] A. Argyriou, C. A. Micchelli, and M. Pontil, "Learning convex combinations of continuously parameterized basic kernels," in *Proc. of the 18th Conference on Learning Theory*, 2005, pp. 338–352.

[27] C. von Mering, R. Krause, B. Snel, M. Cornell, S. G. Oliver, S. Fields, and P. Bork, "Comparative assessment of large-scale data sets of protein-protein interactions," *Nature*, vol. 417, pp. 399–403, 2002.