Improved Semi-Supervised Learning with Multiple Graphs

Krishnamurthy Viswanathan[1]
Google Research

Sushant Sachdeva[1]
University of Toronto

Sujith Ravi **Andrew Tomkins**
Google Research

Abstract

We present a new approach for graph based semi-supervised learning based on a multi-component extension to the Gaussian MRF model. This approach models the observations on the vertices as jointly Gaussian with an inverse covariance matrix that is a weighted linear combination of multiple matrices. Building on randomized matrix trace estimation and fast Laplacian solvers, we develop fast and efficient algorithms for computing the best-fit (maximum likelihood) model and the predicted labels using gradient descent. Our model is considerably simpler, with just tens of parameters, and a single hyper-parameter, in contrast with state-of-the-art approaches using deep learning techniques. Our experiments on benchmark citation networks show that the best-fit model estimated by our algorithm leads to significant improvements on all datasets compared to baseline models. Further, our performance compares favorably with several state-of-the-art methods on these datasets, and is comparable with the best performances.

1 Introduction

In many learning scenarios, we have access to a large number of instances, relatively few of which are labeled. This situation is common, for instance, in image search, genomics, natural language processing, and speech recognition. Semi-supervised learning (SSL) seeks to leverage the unlabeled instances for improved learning performance.

Graph based SSL is an influential approach that was proposed in the work of Zhu, Ghahramani, Lafferty [1], and Zhou *et al.* [2]. It exploits a similarity function between the instances that exists naturally, or can be derived, allowing both labeled and unlabeled instances to be placed as vertices in a graph whose edges denote similarity. This approach, as proposed in [1], models the labels on the vertices according to a Gaussian Markov random field (MRF) model specified by the graph (see Section 2). Predicted labels, for all the unlabeled vertices in the graph, are computed using the Maximum Likelihood estimate (MLE) for the vertex labels, which can be estimated by solving a system of linear equations in the graph Laplacian. Or equivalently, one can view the label information as propagating through the edges of the graph [2], allowing one set of unlabeled instances to contribute to the inferred labels of another set. If the similarity functions are of high quality, the learning performance can often be improved by such techniques. In particular, if the underlying data exhibits a natural graph structure (e.g. social network, citation network) it is likely suited for label propagation.

In the last couple of years, there has been a surge of methods with the objective of repeating the success of techniques from neural networks to graph based SSL. These include learned graph embeddings (Planetoid) [3], graph convolution networks (GCN) [4], attention mechanisms in graphs (GAT) [5], (AGNN) [6] and GANs [7, 8]. These neural network-based models perform very well on benchmark datasets; GCN and GAT in particular beat all approaches based on the Gaussian random field assumption, including ours. However, the margin is modest, and models such as ours have several offsetting advantages, suggesting that high-performing techniques of both flavors are important:

1. Our model is considerably simpler, with just tens of parameters, in contrast with several thousand parameters in some of the DNN-based approaches. In the case of very limited supervision, such parsimonious models are desirable.
2. Specifically, models with a large number of parameters, and recurrence / attention mechanisms can

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89. Copyright 2019 by the author(s).

[1]Equal contribution

be hard to train, and require handpicked architectural choices (see [5] for a fascinating description of the amount of complex tuning required to attain top performance). In contrast, our approach requires just a single hyperparameter, and no hand tuning of the network architectures to obtain optimal performance in all the experiments we show.

3. Given the small optimization space and our algorithms, the end to end training is very efficient in comparison, especially when accounting for the necessary hyperparameter optimization.
4. Finally, the simplicity and linear structure of our model allows for easy interpretability, both for the importance of the various components of the Laplacian (see below), and to trace how the predicted label at each vertex is influenced by its neighbors, and in fact, by each of the original labeled examples. As interpretability of models becomes increasingly important in many user-facing situations, this property of our approach is a significant advantage.

1.1 Our Contributions

In this paper, we propose an approach to extract considerably more power from the Gaussian MRF model. We present an approach for graph based SSL based on a multi-component extension to the Gaussian MRF model, along with efficient algorithms to optimize model fit. We demonstrate experimentally that our conceptually simple approach achieves performance that is comparable to the state of the art, and in some cases beats them on standard benchmark datasets.

The standard Gaussian MRF model posits that the precision (inverse covariance) matrix of the labels on the examples is specified by the similarity graph. Our multi-component extension assumes that the precision matrix is a weighted linear combination of a collection of fixed precision matrices. The combining weights are our model parameters that need to be estimated from the seeds.

Multiple precision matrices arise naturally in several different scenarios. Here are a few seen commonly: 1. For example, vertices could be videos, with one precision matrix to represent co-watch statistics, another for visual/textual similarity between videos, and a third to represent shared tags. While we often have good approaches for tuning each precision matrix, it is not a priori clear how to weight their combination. Depending on the classification task, co-watch statistics may be highly important or almost irrelevant. 2. Settings with important similarity information plus features on the objects are also naturally modeled as multiple precision matrices. Consider the simple motivating case in which each vertex has a vector of binary features. For each feature, we introduce a special vertex, and a separate precision matrix with entries connecting the special vertex to each object with the feature. The combined precision matrix then scores each original vertex according to the summed weights of its features, computing an arbitrary learned linear function of the features. Thus, multiple precision matrices allow vertex features and graph similarity to be combined naturally in a single framework. The empirical validation for this model is evident from our results. Learning the optimal combination of precision matrices results in significant accuracy increase over the baseline.

Our main contribution is an efficient algorithm for computing the best-fit multi-component model based on the MLE principle, and the induced labels. In contrast to the usual Gaussian MRF model, an efficient estimation of the MLE is computationally daunting here since computing the likelihood requires computing the determinant of the graph Laplacian. It is prohibitively expensive to even approximate these determinants for moderately sized graphs.

One could avoid computing the likelihood by using a gradient ascent approach, which does not require evaluating the likelihood. However, computing likelihood gradients involves computing matrix inverses, and hence is also expensive. Nevertheless, building on fast solvers for Laplacian matrices and randomized trace estimation procedures, we bypass these obstacles to present efficient algorithms for approximating these gradients. These stochastic gradient approximations can then be used in any gradient based method for estimating the MLE weights.

We perform experiments on real-world citation networks that are de-facto benchmarks for graph-based SSL (PubMed, Citeseer, CORA) (Section 5.2), combining the underlying citation graphs with bi-partite feature components. The best-fit model estimated by our algorithm leads to significant improvements on all datasets as compared to a simple combination of the components, and just using the underlying citation graphs. Our performance is better than that obtained by state-of-the-art graph embedding methods [3], and comparable to graph neural networks with attention mechanisms [5, 6].

In our experiments above, we model the problem as follows: we have one component corresponding to the citation graph (this is the only component in the standard Gaussian MRF model for graph based SSL), and for a selected set of features, we introduce a simple precision component each that adds a correlation among all vertices that share this feature. Even this simple approach of modeling the influence of a feature results in significant boosts in performance – demonstrating the power of our framework. This is but one possible simple way of incorporating features, and our framework and algorithms can easily accommodate arbitrary feature components.

Since the loss function is not concave, a gradient based algorithm can get stuck in local maxima, and be unable to estimate the MLE. However, our synthetic experiments on small graphs (see Section 5.1) show that the problem is well-behaved, and gradient ascent using gradients estimated by our algorithm converges to MLE estimates that are consistent with a grid search.

1.2 Related Works

There has been a great deal of work done on semi-supervised learning in general and graph-based semi-supervised learning in particular (e.g. [1, 2, 9, 10, 11, 12, 13, 14]). Due to space constraints, we direct the reader to [5] for a thorough review of several works (e.g. [3, 4, 5, 6]) aiming to repeat the success of neural-network based techniques on graph-based semi-supervised learning.

Our method can be viewed as an approach to SSL using multiple graphs, a direction that has seen several previous works [15, 16, 17, 18, 19, 20, 21]. However, these works have two major shortcomings. Firstly, many of them (e.g. [15, 16, 20, 21]) are computationally inefficient, requiring running time at least roughly cubic in the number of examples. Secondly, the remaining works [17, 18, 19] use an approximation of the Gaussian MRF model, ignoring the normalizing determinant term. This approximation results in best-fit models consisting of a single component. These degenerate solutions are then avoided by introducing regularizations on the model parameters that are not always well motivated. Our paper presents a complete generalization of the Gaussian MRF model to multiple graphs, and crucially, provides efficient algorithms for computing the best-fit model and predicted labels. It isn't a priori clear that moving fully into the GRF framework will help empirically, but our experiments show that our estimator significantly outperforms other GRF-style approaches. Hence, we argue that our approach represents the correct version of multi-component graph SSL in the GRF setting, and is both the first unbiased estimator and the best-performing in this framework.The power of our approach is evidenced by improved performance without introducing regularization or new hyperparameters.

2 Semi-Supervised Learning

Preliminaries: Graphs and Matrices. For any graph $G(V, E)$ with a function w specifying the edge weights, the (weighted) Laplacian of G, L_G, is an $V \times V$ matrix such that for $i \neq j$, $L_{ij} = -w_{ij}$ where w_{ij} is the weight of edge (i, j), and $L_{ii} = \sum_{j:j\neq i} w_{ij}$. For any vector $x \in \mathbb{R}^V$, the Laplacian satisfies $x^\top L_G x = \sum_{i,j} w_{ij}(x_i - x_j)^2$. The edge-vertex incidence matrix of G, denoted B_G is an $V \times E$ matrix, where $(B_G)_{v,e}$ is 1 if v is e's head, -1 if v is e's tail, and 0 otherwise (the orientation of the edges can be picked arbitrarily). Let W_G denote the diagonal $E \times E$ matrix with diagonal entries $w(e)$. We have $L_G = B_G W_G B_G^\top$.

We now present the underlying model that the data is assumed to be generated from.

Gaussian Random Field Model. Let V denote the set of all examples (labeled and unlabeled), and $n = |V|$. We assume we are given a graph $G(V, E)$ that encodes similarities between the all the points. A subset of these examples $S \subseteq V$ is labeled with real valued observations, and are referred to as seeds. Let $\widehat{\mathbf{y}}_{\mathbf{S}} \in \mathbb{R}^S$ denote the vector of these observations. A binary classification problem can be encoded using $\widehat{\mathbf{y}}_{\mathbf{S}} \in \{0, 1\}^S$, whereas a multi-class classification problem over $[\ell]$ can be encoded as ℓ binary classification problems.

The Gaussian random field model introduced in [1] modeled the true underlying labels $\mathbf{y} \in \mathbb{R}^V$ as a multivariate Gaussian distribution with density

$$f(\mathbf{y}) = \frac{\sqrt{\det(L_G)}}{(2\pi)^{n/2}} \exp\left(-\tfrac{1}{2}\mathbf{y}^\top L_G \mathbf{y}\right),$$

where L_G denotes the weighted Laplacian of graph G. Note that the Laplacian of G is a singular matrix, and hence the above distribution is not well-defined. In the rest of the paper, we assume that a small multiple of the identity has been added to the Laplacian to make it positive-definite.

For any label vector, $\mathbf{y}$, let $\mathbf{y}_{\mathbf{S}}$ denote the vector restricted to the seed set S. Conditioned on $\mathbf{y}_{\mathbf{S}}$, we model the observed labels on S, $\widehat{\mathbf{y}}_{\mathbf{S}}$ to be distributed as independent Gaussians with mean $\mathbf{y}_{\mathbf{S}}$ and variance $\frac{1}{\mu_0}$, resulting in the joint probability density,

$$\begin{aligned} f(\mathbf{y}, \widehat{\mathbf{y}}_{\mathbf{S}}) = & \frac{\sqrt{\det(L_G)}}{(2\pi)^{n/2}} \left(\tfrac{\mu_0}{2\pi}\right)^{\frac{|S|}{2}} \\ & \exp\left(-\tfrac{1}{2}\mathbf{y}^\top L_G \mathbf{y} - \tfrac{\mu_0}{2}\textstyle\sum_{i\in S}(y_i - \widehat{y}_i)^2\right). \end{aligned}$$

Given the observations $\widehat{\mathbf{y}}_{\mathbf{S}}$ on the labeled examples, we can easily compute the maximum a posteriori estimate (MAP) for all the labels by solving the system $(L_G + \mu_0 I_S)\mathbf{y} = [\mu_0 \widehat{\mathbf{y}}_{\mathbf{S}}^\top, \mathbf{0}]^\top$, where I_S is a 0/1 diagonal matrix with $(I_S)_{ii} = 1$ iff $i \in S$, and we pad the vector on the right using zeros to be n-dimensional. This is essentially the estimate $\mathbf{E}[\mathbf{y}_S | \widehat{\mathbf{y}}_{\mathbf{S}}]$ and follows from standard lemmas on multivariate Gaussian distributions.

3 Multiple Components in the Gaussian Random Field Model

Given several precision matrices L_i for $i = 1, \ldots, k$, for observations on the same set of examples, we consider the Gaussian MRF model defined by linear combinations of L_i, weighed by weights μ_i. Equivalently, we have, $L_G = \sum_{i=1}^{k} \mu_i L_i$. We assume that our observations $\widehat{\mathbf{y}}_{\mathbf{S}}$ arise from the Gaussian Random field model with a precision matrix specified by L_G. In this section, we formulate the estimation problem for hyperparameters μ_i.

Given the observations $\widehat{\mathbf{y}}_{\mathbf{S}}$, we estimate $\{\mu_i\}$ using the Maximum-Likelihood principle. We denote the maximum likelihood estimate (MLE) of $\{\mu_i\}$ as $\{\widehat{\mu}_i\}$. To compute $\{\widehat{\mu}_i\}$, we need to express the marginal probability of observing $\widehat{\mathbf{y}}_{\mathbf{S}}$. The following key lemma allows us to write this marginal explicitly, and follows from standard results on Gaussians.

Lemma 3.1. *The marginal distribution of* $\widehat{\mathbf{y}}_{\mathbf{S}}$ *is a multivariate Gaussian distribution with mean* $\mathbf{0}$ *and covariance* $\frac{1}{\mu_0}\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}$*, where,* $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}^{-1} = I - \Pi_S(\frac{1}{\mu_0}L_G + I_S)^{-1}\Pi_S^\top$*, where* Π_S *is an* $S \times V$ *matrix such that* $(\Pi_S)_{ij} = 1$ *iff* $i = j$*. We can express* $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}$ *equivalently as follows,* $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}} = I + \Pi_S L^{-1} \Pi_S^\top$.

For optimizing the marginal distribution of $\widehat{\mathbf{y}}_{\mathbf{S}}$, it is more useful to define $\alpha_i = \frac{\mu_i}{\mu_0}$ for $i \geq 1$. With this re-parametrization, we can express $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}^{-1} = I - \Pi_S(I_S + \sum_{i=1}^{n} \alpha_i L_i)^{-1}\Pi_S^\top$. Treating α_i as variables independent of μ_0, $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}$ is independent of μ_0. The above formulation allows us to analytically express $\widehat{\mu}_0$, as given by the following lemma.

Lemma 3.2. *The MLE of* μ_0*, denoted* $\widehat{\mu}_0$ *can be expressed analytically as* $\widehat{\mu}_0 = \frac{|S|}{\widehat{\mathbf{y}}_{\mathbf{S}}^\top \mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}^{-1} \widehat{\mathbf{y}}_{\mathbf{S}}}$.

However, even in our synthetic experiments, this analytic estimate of μ_0 turned out to be significantly inaccurate. Since μ_0 controls the noise in the observations, we assume for the rest of the paper that μ_0 is fixed to be some relatively large number, *i.e.*, we assume that the noise in observations is tiny. Thus, ignoring the terms constant in μ_0, we obtain the following theorem characterizing $\{\widehat{\mu}_i\}_{i \geq 1}$.

Theorem 3.3. *For* $i \geq 0$, $\widehat{\mu}_i = \mu_0 \widehat{\alpha}_i$ *where* $\{\widehat{\alpha}_i\}_{i\geq 1} = \arg\max_{\alpha_i \geq 0} \mathcal{L}(\{\alpha_i\})$*, where*

$$\mathcal{L}(\{\alpha_i\}) = \log\det(\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}^{-1}) - \mu_0 \widehat{\mathbf{y}}_{\mathbf{S}}^\top \mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}^{-1} \widehat{\mathbf{y}}_{\mathbf{S}}, \qquad (1)$$

and $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}^{-1} = I - \Pi_S(I_S + \sum_{i=1}^{n} \alpha_i L_i)^{-1}\Pi_S^\top$.

4 Efficient Estimation of the MLE

Maximizing $\mathcal{L}$ for estimating $\{\widehat{\alpha}_i\}_i$ is computationally challenging for several reasons: 1. There is no analytical solution for $\{\widehat{\alpha}_i\}_i$. 2. The loss $\mathcal{L}$ is non-concave. 3. In fact, even computing $\mathcal{L}$ is costly. Specifically, the covariance matrix $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}$ and its determinant are costly to compute (or reliably estimate), and evaluating $\mathcal{L}$ for $n = 1000$ is prohibitively expensive.

Our approach for maximizing $\mathcal{L}$ is based on (stochastic) gradient descent. It has the distinct advantage that we only need to compute (estimate) the gradients, and do not need to compute the objective $\mathcal{L}$ explicitly. As we show in the next lemma, we can express the required gradients analytically. Later in the section, we present an efficient algorithm for computing reliable estimates for the gradients.

Lemma 4.1. *Let* $L = \sum_{i=1}^{n} \alpha_i L_i$*, and* $L_S = L + I_S$*. For any* $i \geq 1$*, we have,*

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \alpha_i} &= Tr(\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}} \Pi_S L_S^{-1} L_i L_S^{-1} \Pi_S^\top) \\ &\quad - \mu_0 \widehat{\mathbf{y}}_{\mathbf{S}}^\top \Pi_S L_S^{-1} L_i L_S^{-1} \Pi_S^\top \widehat{\mathbf{y}}_{\mathbf{S}}.\end{aligned}$$

The expression $\widehat{\mathbf{y}}_{\mathbf{S}}^\top \Pi_S (I_S + L)^{-1} L_i (I_S + L)^{-1} \Pi_S^\top \widehat{\mathbf{y}}_{\mathbf{S}}$ in the gradient can be interpreted as the energy contributed by the inferred labels to G_i. Exactly computing the above gradients requires computing a matrix inverse, which, is prohibitively expensive. In the next section, we develop a fast algorithm that allows us to estimate the gradients without computing the matrix inverse explicitly.

4.1 Approximating the Gradients

We estimate the gradients by two key techniques. The first involves estimating the trace term in the gradients via Hutchinston's method. This reduces trace estimation to matrix-vector multiplications. The second technique involves estimating consequently generated matrix-inverse-vector products (and those in the second term in the gradient), by using fast Laplacian solvers.

Approximating the Trace. The trace of a positive semi-definite (PSD) matrix A can be estimated by computing $g^\top A g$, where g is a vector of independent Rademacher random variables ($+1, -1$ with probability $1/2$ each). The following result shows that just $\Theta(\log n)$ samples are enough for estimating the trace of an $n \times n$ matrix with high probability.

Theorem 4.2 ([22, 23])**.** *Given a PSD* $n \times n$ *matrix* A*, and* $\epsilon, \delta > 0$*. Let* $g_1, \ldots, g_k \in \mathbb{R}^n$ *be random vectors where each coordinate is picked as an independent uniform random variable from* ± 1*. If* $k \geq \frac{6}{\epsilon^2} \ln \frac{2}{\delta}$*, then* $\mathbf{E}_{g_i}\, g_i^\top A g_i = Tr(A)$*, and with probability at least* $1 - \delta$*, we have* $|\frac{1}{k}\sum_{i=1}^{n} g_i^\top A g_i - Tr(A)| \leq \epsilon Tr(A)$.

Algorithm 1 Estimating the gradients $\left\{\frac{\partial \mathcal{L}}{\partial \alpha_i}\right\}$

```
input Seed set $S$, observed values $\widehat{\mathbf{y}}_{\mathbf{S}}$, precision matrices $L_i$, weights $\alpha_i$, number of estimates $r$

  Construct $L \leftarrow \sum_i \alpha_i L_i$ as a sparse matrix.
  Compute $\Pi_S^\top \widehat{\mathbf{y}}_{\mathbf{S}}$ by padding $\widehat{\mathbf{y}}_{\mathbf{S}}$ with zeros on non-seed vertices.
  Estimate $(I_S + L)^{-1}(\Pi_S^\top \widehat{\mathbf{y}}_{\mathbf{S}})$ using fast Laplacian solvers (Theorem 4.4).
  Compute $\widehat{\mathbf{y}}_{\mathbf{S}}^\top \mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}^{-1} \widehat{\mathbf{y}}_{\mathbf{S}} \leftarrow \widehat{\mathbf{y}}_{\mathbf{S}}^\top \widehat{\mathbf{y}}_{\mathbf{S}} - (\widehat{\mathbf{y}}_{\mathbf{S}} \Pi_S^\top)^\top ((I_S + L)^{-1}(\Pi_S \widehat{\mathbf{y}}_{\mathbf{S}}))$.
  for $i \leftarrow 1$ to $k$ do
    $\left(\frac{\partial \mathcal{L}}{\partial \alpha_i}\right)_1 \leftarrow 0$, $\left(\frac{\partial \mathcal{L}}{\partial \alpha_i}\right)_2 \leftarrow \mu_0((I_S + L)^{-1}(\Pi_S^\top \widehat{\mathbf{y}}_{\mathbf{S}}))^\top L_i (I_S + L)^{-1}(\Pi_S^\top \widehat{\mathbf{y}}_{\mathbf{S}})$
  for $j \leftarrow 1$ to $r$ do
    Let $b_j$ be a random $\pm 1$ vector of $(|S| + \sum_i m_i)$-dimensions.
    Compute $M b_j$, for $M$ given by Lem 4.3, using Laplacian solvers for applying $L^{-1}$.
    Compute $z_j \leftarrow (I_S + L)^{-1} \Pi_S^\top M b_j$.
    for $i \leftarrow 1$ to $k$ do $\left(\frac{\partial \mathcal{L}}{\partial \alpha_i}\right)_1 \leftarrow \left(\frac{\partial \mathcal{L}}{\partial \alpha_i}\right)_1 + \frac{1}{r} z_j^\top L_i z_j$.
  return $\left\{\left(\frac{\partial \mathcal{L}}{\partial \alpha_i}\right)_1 - \left(\frac{\partial \mathcal{L}}{\partial \alpha_i}\right)_2\right\}_{i=1\ldots k}$
```

An obstacle to applying this theorem is that $\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}} \Pi_S (I_S + L)^{-1} L_i (I_S + L)^{-1} \Pi_S^\top$ is not a PSD matrix. However, in case each of the L_i is a Laplacian, the following lemma allows us to estimate its trace as the trace of a PSD matrix.

Lemma 4.3. *If L is a graph Laplacian, let B and W denote the edge-vertex incidence matrix, and the diagonal weight matrix of the corresponding graph. Let M be the matrix $[I, \Pi_S L^{-1} B W^{\frac{1}{2}}]$. Then $MM^\top = \mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}}$, and thus,*

$$\begin{aligned} &Tr(\mathbf{\Sigma}_{\widehat{\mathbf{y}}_{\mathbf{S}}} \Pi_S (I_S + L)^{-1} L_i (I_S + L)^{-1} \Pi_S^\top) \\ &\quad = Tr(((I_S + L)^{-1} \Pi_S^\top M)^\top L_i ((I_S + L)^{-1} \Pi_S^\top M)). \end{aligned}$$

The matrix $((I_S + L)^{-1} \Pi_S^\top M)^\top L_i ((I_S + L)^{-1} \Pi_S^\top M)$ is PSD, and hence Theorem 4.2 applies.

Approximating matrix-inverse-vector products. The trace estimator above reduces the estimation to computing matrix vector products. In order to approximate the gradients, we need to compute matrix-vector products with matrices $\Pi_S, \Pi_S^\top, (I_S + L)^{-1}, L_i$ and the matrix M given by Lemma 4.3. Multiplication by $\Pi_S, \Pi_S^\top$, and L_i is cheap and straightforward since these matrices are sparse.

Multiplication with $(I_S + L)^{-1}$ is challenging. Given a vector v, a straightforward approach to approximating $(I_S + L)^{-1} v$ is to solve the system $(I_S + L)x = v$ approximately by using an iterative method such as Conjugate Gradient. However, the running time of CG scales as the square-root of the condition number of $(I_S + L)$, which can be arbitrarily large if α_i are large.

Since L is a symmetric diagonally dominant (SDD) matrix, i.e. for all i, $L_{ii} \geq \sum_{j \neq i} |L_{ij}|$, we can use the seminal work of [24] on Laplacian solvers (and a long line of improvements, see [25] for more references) to solve this system in time nearly-linear in the sparsity of L.

Theorem 4.4 ([24, 26])**.** *Given an $n \times n$ SDD matrix L with m non-zeros, $\epsilon > 0$, and a vector $b = L\bar{x}$, there is an algorithm that outputs with high probability a vector x such that $\|x - \bar{x}\|_L \leq \epsilon \|\bar{x}\|_L$, in expected time $O(m\sqrt{\log n} \log {1/\epsilon})$ (up to $\log\log n$ factors), where for any v, $\|v\|_L = \sqrt{v^\top L v}$.*

Multiplication by the matrix M given by Lemma 4.3 reduces to multiplication by L^{-1}, B, and $W^{1/2}$. B and $W^{1/2}$ are sparse matrices with sparsity $\sum_i |E(G_i)|$, that can be computed efficiently. We can efficiently multiply with L^{-1} for a given vector v by using fast Laplacian solvers discussed above. The following theorem summarizes the running time guarantees of our algorithm (Algorithm 1).

Theorem 4.5. *Algorithm 1 returns unbiased estimates for gradients $\frac{\partial \mathcal{L}}{\partial \alpha_i}$ and runs in expected time $O((\sum_i m_i) r \log^{1/2} n \log {1/\varepsilon})$, where $m_i = |E_i|$, ε is the machine accuracy, and r is the number of vectors used for trace estimation. We pick $r = \Theta(\epsilon^{-2} \log n)$ in order to get accurate unbiased estimates of all gradients in a total run time of $O((\sum_i m_i)\epsilon^{-2} \log^{3/2} n \log {1/\varepsilon})$.*

5 Experiments

We present the implementation details necessary for stable and rapid convergence of our algorithm, followed by our experimental results on synthetic and real datasets. We experimentally compare our performance to the state-of-the-art methods for graph based SSL [3, 6, 11, 5, 4] on standard benchmark datasets.

[S: This para is new] For methods using multiple graphs, we compare our performance with the most relevant work of Kato *et al.* [19]. The experiments by Kato *et al.* demonstrate the superior experimental performance of their method relative to the works of Tsuda *et al.* [17] and of Argyriou *et al.* [16].

Implementation Details. As mentioned in Section 3, using the analytical expression for $\widehat{\mu}_0$ resulted in inaccurate estimates even in synthetic experiments. Thus, we treat μ_0 as a hyperparameter in our method. We specify μ_0 for our synthetic experiments, and optimized

μ_0 for the datasets.

The form of the gradient ascent (since we're maximizing $\mathcal{L}$) update is important for a stable implementation. Since the weights α_i need to be positive, the usual answer would be to use Projected Gradient Descent ($\alpha_i \leftarrow (\alpha_i + \nu \frac{\partial \mathcal{L}}{\partial \alpha_i})^+$, where ν is the learning rate, and $x^+ = \max\{x, 0\}$). Projected gradient descent results in unstable iterations since the gradients are large and less reliable close to zero. Another option would be to use a mirror descent style update ($\alpha_i \leftarrow \alpha_i \exp(\nu \frac{\partial \mathcal{L}}{\partial \alpha_i})$). However, it often results in large jumps in case of a noisy positive gradient (due to the exponential). Instead, we used the following hybrid update rule $\alpha_i \leftarrow \alpha_i(1 + \nu \frac{\partial \mathcal{L}}{\partial \alpha_i})$, if $\frac{\partial \mathcal{L}}{\partial \alpha_i} \geq 0$, and $\alpha_i \leftarrow \alpha_i \exp(\nu \frac{\partial \mathcal{L}}{\partial \alpha_i})$, otherwise. where ν is the learning rate. This hybrid update rule mitigates the above shortcomings, and leads to significantly more stable updates. Additionally, if some component weights become too small, we remove them from the optimization.

Finally, for solving Laplacian systems, since the work of Spielman and Teng [24] which was very involved, there has been much progress towards simpler and faster solvers (see [25]). Though these algorithms are not directly practical, practical implementations that are small variants of the above theoretical algorithms are available ([27, 28]). We used the solver implemented by Dan Spielman and others [28] for our implementation.

5.1 Synthetic Experiments

We validated the correctness of our algorithm on several small synthetic random graphs (100s of vertices). We generated random graphs based on a random graph model. We generated the underlying labels and the observations according to the joint distribution defined by the graphs and the seed penalty, and selected a random subset of seeds. We ran our algorithm with the seeds and the graph as input and compared the algorithm's estimates to the MLE estimates computed using grid search.

We generated two types of random graphs. For type I, we independently generated two Erdös-Renyi random graphs with a given probability. They were combined according to some pre-specified weights. Our algorithm sees the two graphs but does not know the weight combination. Type II graphs were random feature graphs with 2 feature vertices each. Each feature-instance pair is connected independently according to a given probability. The two features are combined according to some pre-specified weights that are not available to the algorithm. Table 1 summarizes the results on these synthetic graphs for a representative set of parameters. The last column reports the average ℓ_1 distance between the weights estimated by our algorithm and the MLE. In all the cases presented here the fraction of vertices that are seeds was 0.2.

As can be observed, the average distance was always less than twice the resolution of the grid. In almost all the instances, when supplied with sufficient number of random labels the algorithm converged to a point very close to the MLE. While the objective function that we maximize, $\mathcal{L}$, is not concave in the weights, we observed from these experiments on random graphs that the loss landscape seems very well behaved and gradient ascent always seems to converges to the MLE.

5.2 Performance on Real-world Datasets

We tested our approach on three datasets involving scientific publications, namely, Cora, Citeseer and Pubmed [29]. These datasets were obtained from the LINQS website [30]. For experiments based on the splits provided by [3], the datasets were obtained from their github repository [31].

The three datasets have a very similar structure. Two types of graphs are constructed for each of these datasets. The first is a citation graph derived from bibliographic information. There is an edge for each citation connecting the cited paper to the citing paper. The second type of graph is constructed from text features associated with each publication. Papers have high-dimensional text feature vectors associated with them. Each co-ordinate in these feature vectors corresponds to some term. In the Cora and Citeseer datasets the feature vectors are 0/1 vectors indicating whether the corresponding word is absent or present in the publication. In the Pubmed dataset, the value represents the tf-idf statistic for the corresponding term. Given these feature vectors we construct bipartite feature graphs with edges connecting instance vertices to feature vertices. We construct feature graphs by selecting a subset of the features as described below. In all the datasets, the papers are classified into classes. Given true labels for a small subset of papers, the task is to use the citation graphs and feature vectors to obtain classifications for all the papers.

In all our experiments, we learn a weight for the citation graph and one weight for each of the feature graphs. We perform two types of experiments. In one we select seeds, test and validation vertices according to the split used in [3]. In the second, we randomly selected 20 instance vertices for each class to be the seed vertices, a random subset of 500 vertices to be validation vertices and a random subset of 1000 to be test vertices. Since the number of seeds is small it is not feasible to learn

Table 1: Results on synthetic graphs

Type	#Vertices	Density	Weights	#Instances	Grid resolution	Avg. ℓ_1 dist.
I	100	0.1	(1.0, 0.5)	25	0.05	0.083
I	200	0.1	(1.0, 0.5)	25	0.05	0.036
II	100	0.7	(1.0, 0.5)	9	0.1	0.079
II	100	0.7	(1.5, 0.5)	9	0.1	0.073
II	500	0.7	(1.5, 0.5)	9	0.1	0.045

Table 2: Dataset sizes

Dataset	Classes	Instances	Citation edges	Features	Feature edges	Labeled
Cora	7	2708	5429	10	2631	140
Citeseer	6	3327	4732	10	4124	120
Pubmed	3	19717	88676	10	5372	60

Table 3: Classification Accuracy in percent on fixed splits from [3, 31]

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003) with CMN	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu and Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
GCN (Kipf and Welling, 2017)	81.5%	70.3%	79.0%
AGNN (Thekumparampil et al., 2018)	82.6%	71.7%	79.9%
GAT (Velickovic et al., 2018)	83.0%	72.5%	79.0%
Multi-Component MRF (this paper)	80.8%	69.7%	75.3%

Table 4: Classification Accuracy in percent on random splits (all other numbers from [6])

Method	Cora	Citeseer	Pubmed
DeepWalk (Perozzi et al., 2014)	70.2%	47.2%	72.0%
Node2vec (Grover and Leskovec, 2016)	72.9%	47.3%	72.4%
Robust LP (Kato et al. 2009)	73.8 ± 2.6%	66.5 ± 2.0%	75.4 ± 1.4%
Bootstrap (Buchnik and Cohen, 2017)	78.2%	50.3%	75.6%
AGNN (Thekumparampil et al., 2018)	81.0%	69.8%	78.0%
Citation graph with CMN	74.0 ± 1.4%	47.6 ± 1.8%	73.8 ± 1.1%
Citation graph + feature graph with CMN	73.8 ± 2.6%	66.5 ± 2.0%	75.4 ± 1.4%
Multi-Component MRF (this paper)	78.8 ± 1.5%	66.8 ± 2.0%	76.9 ± 1.2%

as many weights as the total number of features (e.g., the Cora dataset has 1433 features and 140 seeds). Therefore, we perform feature selection to restrict the number of feature graphs. Table 2 summarizes the graph sizes after feature selection.

Feature selection procedure: To select the features we build a set of one-versus-all ℓ_1-regularized logistic regression models. From each model we add the features with the largest absolute coefficients to the set of selected features. We pick a value for the ℓ_1 penalty that results in the size of the set of selected features to be a pre-specified number, say 10 or 20.

Once the features are selected, we consider the set of feature graphs corresponding to these features. We treat the set of edges incident on each feature as a separate feature graph and learn its weight. The weights on the edges are given by the values corresponding to that feature in the feature vectors. We run our algorithm on these graphs to learn the weights. Following this, we ran label propagation on the graph until convergence, post-processed the learned labels using class mass normalization (CMN) as in [1], and classified each vertex according to the label with the largest weight. Class

Table 5: Classification Accuracy in percent on random splits under various feature selection scenarios

Feature Selection Method	Cora	Citeseer	Pubmed
Combined graph (all weights 1.0)	60.9 ± 2.7%	53 ± 2.1%	73.7 ± 2.7%
Reweighted graph (without feature selection)	72.8 ± 1.4%	57.6 ± 1.2%	73.7 ± 2.7%
Reweighted graph (with feature selection)	78.8 ± 1.5%	66.8 ± 2.0%	76.9 ± 1.2%

mass normalization scales the label weights so that for every label, the average label weight across all vertices matches the prior as computed from the seeds (in our case uniform). We selected the best set of weights based on the performance on the validation set.

Besides competing approaches, we compare ourselves against two configurations. The first assigns a weight of 1.0 to all the graphs and performs label propagation on this combination. The second is the citation graph, which implies a weight of 1.0 to the citation graph and 0.0 to all the feature graphs. The results are presented in Table 3 for the Planetoid splits and in 4 for the random splits (10 independent choices of seed, validation and test vertices). On the random splits the algorithm beats the available results for graph embedding based approaches on all three datasets. It is competitive with the Attention based Graph Neural Networks [6, 5]. On the Planetoid splits, the algorithm beats Planetoid [3] on Cora and Citeseer datasets. It is competitive with Planetoid on pubmed.

We implemented the method of Kato *et al.*, and report their performance on our citation network experiments. We observed that for all the citation networks and a very wide range of hyper-parameter values, the weight vector of the constituent graphs converged to a scaled all-ones vector. Thus, the final performance on the three datasets was identical to that of a combined graph with all weights equal to 1.0.

Effect of Feature Selection: To isolate the effect of feature selection, we performed an ablation study. We tested the performance of our algorithm with and without feature selection, and show the results in Table 5. The results are reported for random splits. Row 1 shows the performance obtained if all graphs are weighted equally and no optimization is performed. Row 2 shows results for our algorithm without feature selection, with significant improvements in the case of Citeseer and Cora. The number of parameters to be learned (Citeseer: 3704, Cora: 1434, Pubmed: 501) is much more than the number of labelled examples (20 * number of classes) when all the features are used. Hence, as shown in Row 3, feature selection does improve our results.

Acknowledgements

Sushant Sachdeva's research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), a Google Faculty Research award, and a Connaught New Researcher award.

References

[1] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *IN ICML*, pages 912–919, 2003.

[2] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, pages 321–328, Cambridge, MA, USA, 2003. MIT Press.

[3] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 40–48. JMLR.org, 2016.

[4] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[5] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations*, 2018.

[6] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li. Attention-based Graph Neural Network for Semi-supervised Learning. *ArXiv e-prints*, March 2018.

[7] Abhishek Kumar, Prasanna Sattigeri, and Tom Fletcher. Semi-supervised learning with gans: Manifold invariance with improved inference. In *Advances in Neural Information Processing Systems*, pages 5534–5544. Curran Associates, Inc., 2017.

[8] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Ruslan R Salakhutdinov. Good semi-supervised learning that requires a bad gan. In *Advances in Neural Information Processing Systems*, pages 6510–6520. Curran Associates, Inc., 2017.

[9] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *J. Mach. Learn. Res.*, 7:2399–2434, December 2006.

[10] Vikas Sindhwani, Partha Niyogi, and Mikhail Belkin. A co-regularization approach to semi-supervised learning with multiple views. In *Proceedings of the ICML Workshop on Learning with Multiple Views*, 2005.

[11] Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1168–1175, New York, NY, USA, 2008. ACM.

[12] Partha Pratim Talukdar and Koby Crammer. New regularized algorithms for transductive learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD '09, pages 442–457, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] Sujith Ravi and Qiming Diao. Large scale distributed semi-supervised learning using streaming approximation. In *IN AISTATS*, 2016.

[14] Thang D. Bui, Sujith Ravi, and Vivek Ramavajjala. Neural graph learning: Training neural networks using graphs. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 64–71, 2018.

[15] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *J. Mach. Learn. Res.*, 5:27–72, December 2004.

[16] Andreas Argyriou, Mark Herbster, and Massimiliano Pontil. Combining graph laplacians for semi-supervised learning. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, pages 67–74, Cambridge, MA, USA, 2005. MIT Press.

[17] Koji Tsuda, Hyunjung Shin, and Bernhard Schölkopf. Fast protein classification with multiple networks. *Bioinformatics*, 21(2):59–65, January 2005.

[18] M. Wang, X. S. Hua, R. Hong, J. Tang, G. J. Qi, and Y. Song. Unified video annotation via multigraph learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(5):733–746, May 2009.

[19] Tsuyoshi Kato, Hisahi Kashima, and Masashi Sugiyama. Robust label propagation on multiple networks. *Trans. Neur. Netw.*, 20(1):35–44, January 2009.

[20] B. Geng, D. Tao, C. Xu, L. Yang, and X. S. Hua. Ensemble manifold regularization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1227–1233, June 2012.

[21] Motoki Shiga and Hiroshi Mamitsuka. Efficient semi-supervised learning on locally informative multiple graphs. *Pattern Recognition*, 45(3):1035 – 1049, 2012.

[22] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58(2):8:1–8:34, April 2011.

[23] Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Found. Comput. Math.*, 15(5):1187–1212, October 2015.

[24] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. STOC '04, pages 81–90. ACM, 2004.

[25] R. Kyng and S. Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, Oct 2016.

[26] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m \log^{1/2} n$ time. In *STOC*, pages 343–352, 2014.

[27] Yiannis Koutis. `http://www.cs.cmu.edu/~jkoutis/cmg.html`.

[28] Daniel Spielman et al. `https://github.com/danspielman/Laplacians.jl`, 2016.

[29] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[30] Lisa Getoor and other. LINQS Datasets. `https://linqs.soe.ucsc.edu/data`. Accessed: 2018-02-08.

[31] Ruslan Salakhutdinov Zhilin Yang, William W. Cohen. Planetoid github repository. `https://github.com/kimiyoung/planetoid`, 2016. Accessed: 2018-02-08.