Interface Laplace Learning: Learnable Interface Term Helps Semi-Supervised Learning

Tangjun Wang Tsinghua University Beijing, China wangtj20@mails.tsinghua.edu.cn Chenglong Bao*
Tsinghua University
Beijing, China
clbao@mail.tsinghua.edu.cn

Zuoqiang Shi* Tsinghua University Beijing, China zqshi@tsinghua.edu.cn

Abstract

We introduce a novel framework, called Interface Laplace learning, for graph-based semi-supervised learning. Motivated by the observation that an interface should exist between different classes where the function value is non-smooth, we introduce a Laplace learning model that incorporates an interface term. This model challenges the long-standing assumption that functions are smooth at all unlabeled points. In the proposed approach, we add an interface term to the Laplace learning model at the interface positions. We provide a practical algorithm to approximate the interface positions using k-hop neighborhood indices, and to learn the interface term from labeled data without artificial design. Our method is efficient and effective, and we present extensive experiments demonstrating that Interface Laplace learning achieves better performance than other recent semi-supervised learning approaches at extremely low label rates on the MNIST, FashionMNIST, and CIFAR-10 datasets.

CCS Concepts

Computing methodologies → Machine learning algorithms.

Keywords

Graph-based semi-supervised learning, Laplace learning, Interface, Nonlocal model

ACM Reference Format:

1 Introduction

The success of machine learning methods often depends on a large amount of training data. However, collecting training data can be labor-intensive, and is sometimes impossible in many application fields due to privacy or safety issues. To alleviate the dependency on training data, semi-supervised learning (SSL) [14, 48] has received great interest in recent years. Semi-supervised learning typically

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

https://doi.org/10.1145/nnnnnn.nnnnnnn

uses a large amount of unlabeled data, together with the labeled data, to improve model performance and generalization ability. The idea of combining labeled and unlabeled data has been widely used long before the term SSL was coined [8, 20]. By incorporating the geometric structure or data distribution of unlabeled data, SSL algorithms aim to extract more informative features, thereby enhancing the performance of machine learning models.

This paper focuses on a type of SSL method: graph-based SSL [37]. Graph-based SSL algorithms have received much attention as the graph structure can effectively encode relationships among data points, thereby allowing for full utilization of the information contained in unlabeled data. Graph-based SSL is based on the assumption that nearby nodes tend to have the same labels. In a graph, each sample is represented by a vertex, and the weighted edge measures the similarity between samples. One of the most widely used methods in Graph-based SSL is the Gaussian Fields and Harmonic Functions algorithm [47], later commonly called Laplace learning. Laplace learning aims to minimize the graph Dirichlet energy with the constraint on labeled points, resulting in a harmonic function. Many variants of Laplace learning have been proposed. One way is to replace the hard label constraint with soft label regularization [2, 5, 23, 43]. Another way is to generalize the ℓ^2 distance, which corresponds to Laplace learning, into ℓ^p distance, known as p-Laplace learning [9, 10, 18, 36, 44]. In the limit as p approaches infinity, p-Laplace learning is called Lipschitz learning [27].

However, it has been observed that Laplace learning and its variants exhibit poor performance when the label rate is low [12, 18, 32]. In these situations, the solutions tend to converge to non-informative, nearly constant functions with spikes near labeled points, significantly deteriorating the model's accuracy. To address the issue, several methods have been proposed, e.g. higher-order regularization [45], graph re-weighting [13, 34], spectral cutoff [6] and centered kernel [31]. While these methods have been explored, they are either much more computationally burdensome, or still perform poorly when the label rate is extremely low. Recently, Poisson learning [12] has shown promising results in this challenging scenario. Poisson learning replaces the Dirichlet boundary conditions in the Laplace equation with a source term in the Poisson equation, achieving a significant performance increase in classification tasks under extreme label rates.

Nonetheless, most existing methods assume that the solution should exhibit a certain degree of smoothness across all unlabeled points. In this work, we demonstrate that, ideally, there should exist an *interface* between two different classes. The solution should exhibit discontinuity on the interface, rather than being globally smooth. Interface problems are commonly encountered in fields

^{*}Corresponding authors

like materials science [19], fluid dynamics [15, 35], and electromagnetic [30, 38], where the solution is expected to have clear boundaries between different regions or classes, rather than a smooth transition.

In this paper, we formulate the interface problem in graph-based SSL as solving the Laplace equation with jump discontinuity across the interface. By deriving the nonlocal counterpart of this system, we find that it is naturally related to Laplace learning, but with the addition of an explicit interface term. By accounting for the non-smoothness across the interface, our approach can more accurately model the underlying data distribution, resulting in improved performance compared to standard Laplace learning and its variants.

Our main contribution can be summarized as follows:

- We are the first to propose the concept of interface discontinuity in graph-based SSL, offering a new perspective and algorithmic design in this domain.
- We introduce a Laplace learning model that explicitly accounts for interface discontinuities, improving upon existing approaches.
- We develop a practical algorithm to approximate the interface and learn the interface term without deliberate design.
- On benchmark classification tasks with extremely low label rates, our method achieves state-of-the-art performance.

2 Motivation

2.1 Laplace Learning and Poisson Learning

SSL aims to infer the labels of unlabeled samples $\{x_{m+1}, \dots, x_n\}$ with the help of a labeled set, $\{(x_1, y_1), \dots, (x_m, y_m)\}$. For a classification problem with c classes, the label y_i is often represented as a one-hot vector in \mathbb{R}^c , where the l-th element is 1 if the sample belongs to class l, and the other elements are all zeros. Typically, the number of all samples n is much larger than the number of labeled samples m. Graph-based SSL methods construct a graph where the nodes correspond to the samples, $V = \{x_1, x_2, \dots, x_n\}$. An edge e_{ij} is created between nodes x_i and x_j if the two samples are similar, and a non-negative weight w_{ij} is assigned to the edge, indicating the degree of similarity between x_i and x_j .

Among numerous graph-based SSL approaches, Laplace learning [47], also known as label propagation, is one of the most widely used methods. Laplace learning propagates labels to unlabeled nodes by solving the following Laplace equation:

$$Lu(x_i) = 0$$
 $m+1 \le i \le n$
 $u(x_i) = y_i$ $1 \le i \le m$

where L is the graph Laplacian operator given by

$$Lu(x_i) = \sum_{j=1}^{n} w_{ij}(u(x_i) - u(x_j))$$

The solution $u: V \to \mathbb{R}^c$ gives the prediction of size c for each vertex x_i . The label decision is then determined by the largest component in $u(x_i)$. In Laplace learning, labeled data is incorporated as Dirichlet boundary conditions $u(x_i) = y_i$.

Recently, Poisson learning [12] is proposed as an alternative to Laplace learning to deal with extremely low label rate. Poisson learning treats unlabeled data in the same way as Laplace Learning, but it differs in the way of handling labeled data. Poisson learning

replaces the Dirichlet boundary condition with the Poisson equation and a given source term $y_i - \bar{y}$,

$$Lu(x_i) = 0 m+1 \le i \le n$$

$$Lu(x_i) = y_i - \bar{y} 1 \le i \le m$$

where $\bar{y} = \frac{1}{m} \sum_{j=1}^{m} y_j$. Surprisingly, such modification can result in huge improvement under extremely low label rates, e.g. 16.73% \rightarrow 90.58% in MNIST [28] 1-label per class classification.

2.2 Interface discontinuity

Both Laplace learning and Poisson learning assume that the graph Laplacian of the target function $Lu(x_i)=0$ on all unlabeled points. However, we question whether this is the most appropriate way to model the underlying data distribution. To illustrate our idea, we will consider a synthetic 2-class classification example.

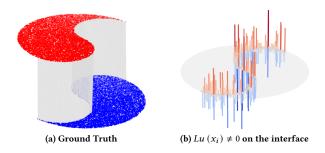


Figure 1: 3D visualization of a synthetic classification example.

We uniformly sample 20,000 points $x_i=(a_i,b_i)\in\mathbb{R}^2$ from a unit circle. The decision boundary between the two classes is defined as the union of two half-circles: $\{a<0,a^2+(b-0.5)^2=0.5^2\}\cup\{a\geq0,a^2+(b+0.5)^2=0.5^2\}$. The ground truth label for each x_i is in $\{-1,+1\}\in\mathbb{R}$. A 3D visualization of this toy example is given in Figure 1(a). To construct the similarity matrix $\mathbf{W}=(w_{ij})\in\mathbb{R}^{n\times n}$, we use a Gaussian kernel defined as:

$$w_{ij} = \exp\left(-\frac{4\|x_i - x_j\|^2}{d_K(x_i)^2}\right)$$
 (1)

where $d_K(x_i)$ is the distance between x_i and its K-th nearest neighbor. We choose K = 10, and we sparsify the matrix \mathbf{W} by truncating the weight for points farther than the K-th nearest neighbor to zero.

We will examine the problem from two different perspectives. Firstly, we assign the ground truth labels to the function $u(x_i)$ and then compute and plot $Lu(x_i) \in \mathbb{R}$ in Figure 1(b). From the figure, it is clear that the Laplacian of the labeling function is nonzero along the interface between the two classes, while being zero in the interior of each class. This nonzero Laplacian near the decision boundary is due to the interface discontinuity in the labeling function. However, this interface discontinuity is ignored by both Laplace learning and Poisson learning, as they assume the function is harmonic almost everywhere, except at the labeled points. The assumption of harmonicity contradicts the true Laplacian behavior observed in the ground truth labeling function.

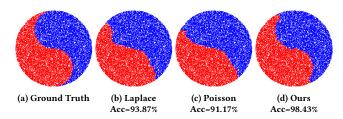


Figure 2: 2D visualization of a synthetic classification example.

Secondly, we will show that adding an interface term to account for the interface discontinuity is helpful for classification. Specifically, we choose to modify the Laplace equation by introducing an interface term f_i that will be learned from the labeled data:

$$Lu(x_i) = 0 \quad i \notin I$$

$$Lu(x_i) = f_i \quad i \in I$$
(2)

Here, $\mathcal I$ denotes the set of indices corresponding to the interface positions. This formulation allows us to explicitly model the nonzero Laplacian at the interface, in contrast to the assumptions made by standard Laplace and Poisson learning methods.

The theoretical basis for introducing this interface term to account for interface discontinuity will be provided in the next subsection. In this toy example, the set I is obtained by identifying the indices where the ground truth Laplacian, as shown in Figure 1(b), is nonzero. The values of f_i will then be inferred from the labeled points using the algorithm detailed in Section 3. Finally, we use the solution u to Eq. (2) for classification.

We randomly select 25 labeled samples from each class, and use Laplace learning, Poisson learning, and our proposed method to classify the remaining data points. The classification results are provided in Figure 2. We can observe that our method significantly outperforms both Laplace learning and Poisson learning in terms of classification accuracy. Moreover, the decision boundary obtained by our method is also much closer to the ground truth. Indeed, the comparison is unfair because our method utilizes additional information about the interface positions, which the other two methods do not have access to. However, this toy classification problem serves to verify our argument that incorporating an interface term to account for the discontinuity is both necessary and beneficial for improving classification performance.

2.3 Laplace equation with interface and associate nonlocal model

In this subsection, we will provide a theoretical verification of our approach from the perspective of nonlocal models. Nonlocal models [17] play a crucial role in many fields, such as peridynamical theory of continuum mechanics, nonlocal wave propagation and nonlocal diffusion process [1, 3, 7, 16, 24, 39]. The terminology *nonlocal* is the counterpart of *local* operators. A linear operator L is considered local if the support set $\sup\{L(f)\}\subset\sup\{f\}$ for any function f. Differential operators, such as the Laplace operator Δ , are local operators.

As discussed in previous subsection, the interface problem in SSL can be modeled by the following Laplace equations with jump discontinuity on the interface Γ ,

$$\Delta u^{+}(x) = 0 \quad x \in \mathcal{M}_{1}$$

$$\Delta u^{-}(x) = 0 \quad x \in \mathcal{M}_{2}$$

$$u^{+}(x) - u^{-}(x) = 1 \quad x \in \Gamma$$

$$\frac{\partial u^{+}}{\partial \mathbf{n}}(x) - \frac{\partial u^{-}}{\partial \mathbf{n}}(x) = 0 \quad x \in \Gamma$$
(3)

Here Γ is a (k-1)-dimensional smooth manifold that splits a k-dimensional manifold \mathcal{M} into two submanifolds \mathcal{M}_1 and \mathcal{M}_2 , so that $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$ and $\mathcal{M} = \mathcal{M}_1 \cup \Gamma \cup \mathcal{M}_2$. \mathbf{n} is the outer normal of Γ .

To get nonlocal approximation, we introduce a rescaled kernel function $R_{\delta}(x,y) = C_{\delta}R(\frac{\|x-y\|^2}{4\delta^2})$, where $R \in C^2([0,1])$ is a non-negative compact function that is supported over [0,1]. $C_{\delta} = (4\pi\delta^2)^{-d/2}$ is a normalization factor for $x \in \mathbb{R}^d$. The rescaled functions $\bar{R}_{\delta}(x,y)$ and $\bar{R}_{\delta}(x,y)$ are defined similarly, where $\bar{R}(r) = \int_r^{+\infty} R(s)ds$ and $\bar{R}(r) = \int_r^{+\infty} \bar{R}(s)ds$. The role of kernel function is similar to the similarity matrix \mathbf{W} in Eq. (1), where only nearest neighbors have nonzero weights. If we assume $\int_{\mathcal{M}_1} \bar{R}_{\delta}(y,s)d\mu_s = \int_{\mathcal{M}_2} \bar{R}_{\delta}(y,s)d\mu_s := w_{\delta}(y)$, then [42] gives a nonlocal model to approximate Eq. (3) as:

$$\int_{\mathcal{M}_{1}} R_{\delta}(x,y) (u_{\delta}^{+}(x) - u_{\delta}^{+}(y)) d\mu_{y} = \int_{\Gamma} \bar{R}_{\delta}(x,y) v_{\delta}(y) d\tau_{y}$$

$$x \in \mathcal{M}_{1}$$

$$\int_{\mathcal{M}_{2}} R_{\delta}(x,y) (u_{\delta}^{-}(x) - u_{\delta}^{-}(y)) d\mu_{y} = -\int_{\Gamma} \bar{R}_{\delta}(x,y) v_{\delta}(y) d\tau_{y}$$

$$x \in \mathcal{M}_{2}$$

where

$$v_{\delta}(y) = \frac{w_{\delta}(y) + \int_{\mathcal{M}_2} u_{\delta}^-(s) \bar{R}_{\delta}(y,s) d\mu_s - \int_{\mathcal{M}_1} u_{\delta}^+(s) \bar{R}_{\delta}(y,s) d\mu_s}{2 \int_{\Gamma} \bar{\bar{R}}_{\delta}(y,s) d\tau_s}$$

is an approximation to the normal derivative up to a constant.

The solutions of the nonlocal model are proven to converge to the solutions of the local model with the rate of $O(\delta)$ in H^1 norm.

Theorem ([42]). (1) (Well-Posedness) For any $f \in L^2(\mathcal{M})$, there exists a unique solution $(u_{\delta}^+, u_{\delta}^-) \in (H^1(\mathcal{M}_1), H^1(\mathcal{M}_2))$ to Eq. (4). (2) (Convergence) For any $f \in H^1(\mathcal{M}_1 \cup \mathcal{M}_2)$, let $(u^+, u^-) \in (H^3(\mathcal{M}_1), H^3(\mathcal{M}_2))$ be the solution to Eq. (3), and $(u_{\delta}^+, u_{\delta}^-)$ be the solution to Eq. (4), then

$$\begin{split} \left\| u^{+} - u_{\delta}^{+} \right\|_{H^{1}(\mathcal{M}_{1})} + \left\| u^{-} - u_{\delta}^{-} \right\|_{H^{1}(\mathcal{M}_{2})} \\ & \leq C \delta \left(\left\| u^{+} \right\|_{H^{3}(\mathcal{M}_{1})} + \left\| u^{-} \right\|_{H^{3}(\mathcal{M}_{2})} \right) \end{split}$$

where the constant C only depends on M and Γ .

Nonlocal model formulation (4) provides theoretical justification of incorporating an interface term to account for interface discontinuity in our algorithm (2). On the left-hand side of the nonlocal model, We can identify the integral $\int_{\mathcal{M}_1} R_{\delta}(x,y) (u_{\delta}^+(x) - u_{\delta}^+(y)) d\mu_y$ with the graph Laplacian operator $Lu(x_i) = \sum_{j=1}^n w_{ij}(u(x_i) - u(x_j))$ by discretizing the integral. On the right-hand side, since $\bar{R}_{\delta}(x,y)$ has compact support, the integral $\int_{\Gamma} \bar{R}_{\delta}(x,y) v_{\delta}(y) d\tau_y$ is

nonzero only when x lies in a layer adjacent to the interface Γ with width 2δ . We can relate this adjacent layer of Γ to the interface positions $\mathcal I$ introduced earlier. This explains our choice of introducing $f_i = Lu(x_i) \neq 0$ for $i \in \mathcal I$.

While the nonlocal formulation provides theoretical justification for the interface term, it is not directly applicable to semi-supervised learning. The variables v_{δ} and $(u_{\delta}^+, u_{\delta}^-)$ are coupled together, making it challenging to write Eq. (4) into a linear system. Furthermore, the interface Γ is not given explicitly, which makes the computation of the integral over Γ impossible. To address these limitations, we will make the interface terms learnable and provide a practical algorithm in subsequent section.

REMARK. The assumptions $\frac{\partial u}{\partial n}^+(x) = \frac{\partial u}{\partial n}^-(x)$ for $x \in \Gamma$ and $\int_{\mathcal{M}_1} \bar{R}_{\delta}(y,s) d\mu_s = \int_{\mathcal{M}_2} \bar{R}_{\delta}(y,s) d\mu_s$ are introduced to simplify the expression of the nonlocal model Eq. (4). However, these assumptions can be relaxed by introducing coefficients: $\lambda_1 \frac{\partial u}{\partial n}^+(x) = \lambda_2 \frac{\partial u}{\partial n}^-(x)$, $\gamma_{\delta}(y) = \int_{\mathcal{M}_2} \bar{R}_{\delta}(y,s) d\mu_s / \int_{\mathcal{M}_1} \bar{R}_{\delta}(y,s) d\mu_s$. Correspondingly, the nonlocal model should be slightly modified to incorporate these coefficients. The exact form of the modified nonlocal model is provided in [42].

3 Method

In the previous section, from the premise that the function u should be discontinuous at the interface, we propose a formal algorithm:

$$Lu(x_i) = 0$$
 $i \notin I$
 $Lu(x_i) = f_i$ $i \in I$

However, two key questions remain unsolved:

- For $i \in \mathcal{I}$, how do we decide the value of interface term f?
- For general classification problems where the interface is not provided a priori, how do we determine *I*?

In the subsequent two subsections, we will provide ideas to address these questions, and the final algorithm will be presented in Section 3.3.

3.1 Interface term learning

The idea is straight forward: we want to learn a interface term such that the solution to the modified Laplace equation on labeled points is as close as possible to the given labels. To this end, we use the Mean Squared Error (MSE) on the labeled points as the objective function:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{m} \sum_{i=1}^{m} \|u(x_i) - y_i\|_2^2$$

In addition to the MSE loss, we find that a regularizer on the norm of f_i is beneficial for learning:

$$\mathcal{L}_{\text{reg}} = \sum_{i=1}^{n} \|f_i\|_2^2$$

This regularization not only prevents overfitting to the few labeled data, but also improves the conditioning of the problem. The final objective function is

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda \mathcal{L}_{\text{reg}}$$

where λ is a weighting factor (later called the ridge parameter).

By minimizing this objective function, we can learn the interface term from label information. We will formulate the problem as a standard least squares (LS) problem, and the detailed optimization algorithm will be provided in Section 3.3.

3.2 Interface positions approximation

In the synthetic 2-class classification example presented in Section 2.2, the interface positions I are obtained by leaking label information and given as an additional input. However, in real-world classification tasks, the interface positions are generally not known a priori. Furthermore, compared to the two-dimensional and well-separated synthetic data, real-world datasets often possess high-dimensional data with much more sophisticated data distributions. Even if the interface between categories exists, it is impossible for us to know the exact locations of these interfaces.

Here, We approximate the interface positions by excluding the k-hop neighbors of training samples, where k-hop neighbors of a node v are those within distance k from v in the graph. The distance between two vertices in a graph is defined as the number of edges in a shortest path connecting them. We efficiently obtain the k-hop neighbors using an iterative approach, starting with the training samples (distance-0) and finding their direct neighbors (distance-1), then their neighbors (distance-2), and so on. We provide the Python implementation of this get_interface_idx() method in Appendix D.

The reason we approximate the interface positions in this way is two-fold. Theoretically, training samples and their k-hop neighbors are more likely to lie in the interior of each class rather than on the boundary. By excluding these k-hop neighbors, the remaining index can be assumed to represent the interface index. Empirically, the k-hop samples are closer to the labeled samples and have a larger impact on the prediction $u(x_i)$ on labeled points. Since the labeled samples are scarce, including the k-hop neighbors in f_i can easily lead to overfitting. A discussion of other possible approaches to approximate interface positions will be provided in Section 4.2.

3.3 Algorithm

For ease of writing, we introduce the following notations. $\mathbf{f} = [f_1, \cdots, f_n]^\top$, $\mathbf{u} = [u(x_1), \cdots, u(x_n)]^\top$, both belong to $\mathbb{R}^{n \times c}$. $\mathbf{y} = [y_1, \cdots, y_m]^\top \in \mathbb{R}^{m \times c}$. The graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W} \in \mathbb{R}^{n \times n}$, where $\mathbf{D} = \operatorname{diag}(d_i)$, $d_i = \sum_{j=1}^n w_{ij}$. L is related to the graph Laplacian operator L in that $\mathbf{L}\mathbf{u} = [Lu(x_1), \cdots, Lu(x_n)]^\top$.

The objective is to solve the following optimization problem after we obtain the interface positions I using get_interface_idx().

$$\underset{\mathbf{f}}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \|u(x_i) - y_i\|_2^2 + \lambda \sum_{i=1}^{n} \|f_i\|_2^2$$
s.t. $f_i = 0, i \notin I$ (5)

 ${\bf u}$ and ${\bf f}$ are related by the Poisson equation ${\bf L}{\bf u}={\bf f}$. Since we cannot directly write ${\bf u}={\bf L}^{-1}{\bf f}$ because L is singular, we adopt the iterative solver in Poisson learning [12]. Specifically, we initialize ${\bf u}_0$ as an all-zero matrix in $\mathbb{R}^{n\times c}$ and writes the iteration step as

$$\mathbf{u}_{t+1} \leftarrow \mathbf{u}_t + \mathbf{D}^{-1}(\mathbf{f} - \mathbf{L}\mathbf{u}_t) \tag{6}$$

The stopping criterion is determined by another loop $\mathbf{p}_{t+1} = \mathbf{W}\mathbf{D}^{-1}\mathbf{p}_t$, where $\mathbf{p}_0 \in \mathbb{R}^n$ is initialized as a vector with ones at the positions of all labeled vertices and zeros elsewhere. Once $\|\mathbf{p}_t - \mathbf{p}_{\infty}\|_{\infty} \le \frac{1}{n}$, where $\mathbf{p}_{\infty} = W\mathbf{1}/(\mathbf{1}^{\top}W\mathbf{1})$ represents the invariant

distribution, the iteration is stopped. The number of iterations is denoted as T, which is often around 200-300.

We can unroll the iteration Eq. (6) and write $\mathbf{u} = \mathbf{u}_T$ in terms of \mathbf{f} directly.

$$\mathbf{u} = \sum_{i=0}^{T-1} (\mathbf{D}^{-1} \mathbf{W})^i \mathbf{D}^{-1} \mathbf{f} := \mathbf{A} \mathbf{f}$$

Then the objective function in optimization problem (5) can be written as

$$\underset{\mathbf{f}}{\operatorname{argmin}} \ \frac{1}{m} \sum_{i=1}^{m} \| (\mathbf{A}\mathbf{f})_i - y_i \|_2^2 + \lambda \sum_{i=1}^{n} \| f_i \|_2^2$$

Since $f_i = 0$ for $i \notin I$, we extract the interface positions of f and denote them as $f_I \in \mathbb{R}^{|I| \times c}$. f_I is the interface term to be learned. Correspondingly, we extract the columns of A and denote them as $A_I \in \mathbb{R}^{n \times |I|}$. Since the MSE loss only considers the m labeled points, we can further extract the rows of A_I that correspond to the m training indices, denoted as $\tilde{A}_I \in \mathbb{R}^{m \times |I|}$. This approach significantly reduces the space complexity because $m \ll n$. Finally, the optimization problem (5) becomes

$$\underset{\mathbf{f}_{\mathcal{I}}}{\operatorname{argmin}} \ \frac{1}{m} \sum_{i=1}^{m} \| (\tilde{\mathbf{A}}_{\mathcal{I}} \mathbf{f}_{\mathcal{I}})_i - y_i \|_2^2 + \lambda \sum_{i \in \mathcal{I}} \| f_i \|_2^2$$

It is in fact the well-known ℓ^2 regularized LS problem (a.k.a ridge regression) with m observations and |I| variables. It has an explicit solution

$$\mathbf{f}_{I}^{*} = (\tilde{\mathbf{A}}_{I}^{\top} \tilde{\mathbf{A}}_{I} + m\lambda \mathbf{I})^{-1} \tilde{\mathbf{A}}_{I}^{\top} \mathbf{y}$$
 (7)

After we learn the interface term \mathbf{f}_{I}^{*} , we can obtain the complete \mathbf{f}^{*} by filling in the values of \mathbf{f}_{I}^{*} at the indices $i \in I$, and setting the remaining elements to zero.

Notably, we employ the following methods to address complexity issues and solution non-uniqueness.

 In Eq. (7), we need to invert an |I| × |I| matrix, which is slow when |I| is large. A straightforward improvement comes from famous Sherman-Morrison-Woodbury formula, which performs inversion on m × m matrix instead.

$$\mathbf{f}_{\mathcal{I}}^* = \tilde{\mathbf{A}}_{\mathcal{I}}^\top (\tilde{\mathbf{A}}_{\mathcal{I}} \tilde{\mathbf{A}}_{\mathcal{I}}^\top + m \lambda \mathbf{I})^{-1} \mathbf{y}$$

• Lu = f does not have a unique solution. Thus, we enforce a zero mean on each column of u by subtracting $\bar{\mathbf{u}} = \frac{1}{n} \sum_{i=1}^{n} u(x_i)$ along with each iteration step Eq. (6). Consequently, the matrix A should be slightly modified as $\mathbf{A} = \sum_{i=0}^{T-1} \mathbf{J} (\mathbf{D}^{-1} \mathbf{W} \mathbf{J})^i \mathbf{D}^{-1}$, where $\mathbf{J} = \mathbf{I} - \frac{1}{n} \mathbf{1} \mathbf{1}^T$ is a projection matrix that removes the mean from each column.

The final algorithm is provided in Algorithm 1. Notice that we use the iterative solver during inference because we need the predictions $u(x_i)$ for all samples, rather than only for training samples as in the training stage.

4 Experiments

4.1 Classification at very low label rates

In this subsection, we conduct experiments to validate the effectiveness of our method under extreme label rates, with 1, 2, 3, 4, 5 labeled points per class, on the following real-world datasets: MNIST [28], FashionMNIST [40], and CIFAR-10 [26]. The MNIST

Algorithm 1 Interface Laplace Learning

```
Input: W, ridge \lambda, y, k-hop, iteration step T
Preprocess:

1: I = \text{get\_interface\_idx} (train_idx, all_idx, W, k)

2: Calculate A = \sum_{i=0}^{T-1} J(D^{-1}WJ)^iD^{-1}

3: \tilde{A}_I = A[\text{train\_idx}, I]

Training:

4: f_I^* = \tilde{A}_I^\top (\tilde{A}_I \tilde{A}_I^\top + m\lambda I)^{-1}y

5: f^* = f_I^* if i \in I; f^* = 0, otherwise.

Inference:

6: \mathbf{u}_0 = \mathbf{0}

7: for t = 1, 2, \dots, T - 1 do

8: \mathbf{u}_{t+1} \leftarrow \mathbf{u}_t + D^{-1}(f^* - \mathbf{L}\mathbf{u}_t)

9: \mathbf{u}_{t+1} = \mathbf{u}_{t+1} - \overline{\mathbf{u}_{t+1}}

10: \mathbf{u} = \mathbf{u}_T
```

and FashionMNIST datasets each contain 70,000 images, while CIFAR-10 contains 60,000 images, all collected from 10 distinct classes. Rather than using raw images to build the similarity graph, we follow the approach of [12], which trains an autoencoder [25] to extract important features from the raw images. This generates a graph with higher quality for our method. The network architecture, loss function, and training procedure used for the autoencoder can be found in [12]. For fair comparison, we directly use the precomputed features provided by [12] in our experiments.

After the features are extracted, we build a graph in the corresponding latent space. We use the Gaussian kernel (as defined in Eq. (1)) to compute the edge weights between nodes in the graph. The pre-processing procedures used to construct the graph are exactly the same as those described in [12]: we set $\mathbf{W} = (\mathbf{W} + \mathbf{W}^\top)/2$ for symmetry, and we set the diagonal entries of \mathbf{W} to zero.

We compare our method with Laplace learning [47], Random Walk [43], multiclass MBO [21], Weighted Nonlocal Laplacian (WNLL) [34], Centered Kernel method [31], Sparse Label Propagation [22], p-Laplace learning [33], Poisson learning [12] and Variance-enlarged Poisson learning (V-Poisson) [46] in Table 1. A nearest neighbor classifier, which decided the label according to the closest labeled vertex with respect to the graph geodesic distance, is provided as a baseline. The results for all methods, excluding V-Poisson, are obtained using the GraphLearning Python package [11]. However, our implementation of the V-Poisson algorithm produces results that differ from the reported performance in [46]. We discuss this discrepancy further in Appendix A.3. In all experiments, we report the average accuracy and standard deviation across 100 random trials, with different labeled points selected each time. We test all methods on the same random permutations. Our method significantly outperforms others on various datasets, and we provide an evaluation of its time and space complexity in Appendix A.1.

4.2 Ablation study on algorithm design

There are several components that contribute to the overall performance of our method, such as the MSE objective function, ℓ^2 regularization, interface positions approximation, and zero mean on each column. It is worthwhile to explore whether there exist superior alternatives to these components. Due to space limitations, the

Table 1: Average accuracy scores over 100 trials with standard deviation on MNIST, FashionMNIST and CIFAR-10.

# Lab	el Per Class	1	2	3	4	5
	Laplace [47]	16.73 ± 7.41	28.04 ± 10.04	42.98 ± 12.18	54.90 ± 12.79	66.94 ± 12.06
	Nearest Neighbor	55.24 ± 4.13	62.88 ± 3.02	67.31 ± 2.56	69.81 ± 2.37	71.39 ± 2.29
	Random Walk [43]	83.12 ± 4.57	88.61 ± 2.12	91.18 ± 1.26	92.33 ± 0.98	93.06 ± 0.86
	MBO [21]	13.03 ± 8.32	16.34 ± 9.37	21.23 ± 10.77	27.47 ± 10.50	33.62 ± 10.81
	WNLL [34]	55.32 ± 13.61	84.86 ± 5.89	91.34 ± 2.80	93.68 ± 1.57	94.60 ± 1.17
MNIST	Centered Kernel [31]	20.43 ± 2.18	25.94 ± 3.05	30.73 ± 3.52	34.63 ± 4.13	37.84 ± 4.07
	Sparse LP [22]	10.14 ± 0.13	10.14 ± 0.21	10.14 ± 0.22	10.18 ± 0.20	10.19 ± 0.22
	p-Laplace [33]	65.93 ± 4.89	75.72 ± 2.83	80.54 ± 1.99	83.02 ± 1.71	84.54 ± 1.56
	Poisson [12]	90.58 ± 4.07	93.35 ± 1.64	94.47 ± 0.99	94.99 ± 0.65	95.29 ± 0.58
	V-Poisson [46]	90.68 ± 4.89	93.98 ± 1.80	94.88 ± 0.94	95.20 ± 0.66	95.38 ± 0.56
	Inter-Laplace	93.13 ± 3.72	95.22 ± 1.01	95.72 ± 0.64	95.94 ± 0.49	96.09 ± 0.43
	Laplace [47]	18.77 ± 6.54	32.34 ± 8.98	43.44 ± 9.59	51.66 ± 7.50	57.38 ± 7.17
	Nearest Neighbor	43.98 ± 4.87	49.51 ± 3.19	53.00 ± 2.57	55.20 ± 2.37	56.95 ± 2.15
	Random Walk [43]	55.43 ± 4.97	62.01 ± 3.20	66.00 ± 2.61	67.93 ± 2.45	69.64 ± 2.07
	MBO [21]	11.27 ± 5.46	13.35 ± 6.24	15.76 ± 6.90	19.16 ± 7.85	22.63 ± 8.50
	WNLL [34]	45.31 ± 7.08	59.24 ± 4.27	65.61 ± 3.32	68.30 ± 2.75	70.35 ± 2.40
FashionMNIST	Centered Kernel [31]	12.03 ± 0.37	13.35 ± 0.52	14.58 ± 0.81	15.95 ± 1.14	16.88 ± 1.05
	Sparse LP [22]	10.11 ± 0.18	10.17 ± 0.23	10.27 ± 0.25	10.26 ± 0.15	10.25 ± 0.17
	p-Laplace [33]	49.86 ± 5.13	56.91 ± 3.18	61.12 ± 2.48	63.46 ± 2.36	65.34 ± 2.03
	Poisson [12]	60.13 ± 4.85	66.57 ± 3.07	69.97 ± 2.50	71.37 ± 2.20	72.67 ± 1.96
	V-Poisson [46]	60.30 ± 5.64	66.70 ± 3.88	70.17 ± 2.97	71.44 ± 2.54	72.52 ± 2.14
	Inter-Laplace	61.52 ± 5.04	68.04 ± 3.44	71.43 ± 2.69	72.77 ± 2.40	74.12 ± 1.95
	Laplace [47]	10.50 ± 1.35	11.27 ± 2.43	11.55 ± 2.62	12.78 ± 3.81	13.88 ± 4.59
	Nearest Neighbor	30.06 ± 3.96	33.36 ± 2.95	35.21 ± 2.63	36.51 ± 2.32	37.70 ± 2.17
	Random Walk [43]	38.97 ± 4.94	45.55 ± 3.70	49.15 ± 3.47	51.75 ± 2.98	53.48 ± 2.29
	MBO [21]	11.07 ± 6.11	12.77 ± 6.76	14.01 ± 7.10	15.91 ± 7.15	17.43 ± 7.33
	WNLL [34]	17.67 ± 5.58	27.28 ± 7.06	34.98 ± 6.80	40.52 ± 5.93	44.78 ± 4.69
CIFAR-10	Centered Kernel [31]	15.86 ± 1.81	17.71 ± 1.84	19.67 ± 2.15	21.53 ± 2.18	22.91 ± 2.42
	Sparse LP [22]	10.08 ± 0.10	10.07 ± 0.12	10.11 ± 0.22	10.03 ± 0.13	10.09 ± 0.15
	p-Laplace [33]	34.33 ± 4.65	40.33 ± 3.56	43.58 ± 3.10	45.99 ± 2.61	47.80 ± 2.17
	Poisson [12]	40.43 ± 5.48	46.63 ± 3.80	49.96 ± 3.84	52.39 ± 2.99	54.03 ± 2.35
	V-Poisson [46]	34.40 ± 4.85	36.73 ± 4.02	37.65 ± 3.60	38.41 ± 3.67	39.07 ± 3.65
	Inter-Laplace	41.71 ± 6.09	49.36 ± 4.20	53.45 ± 3.73	56.32 ± 3.23	58.30 ± 2.44

ablation study for the other choices is provided in Appendix B. In this subsection, we will focus on the approximation of the interface positions, as this is a critical component of the algorithm.

In our algorithm, we remove k-hop neighbors of training samples to approximate the interface positions. To distinguish this method from others, we denote the resulting interface positions as I_{khop} . To validate this choice, we test several other approaches: (1) **Ground truth**. Similar to the method of getting interface positions in Section 2.2, we leak the label information and identify the indices of nonzero ground truth Laplacian values. (2) **Training**. $I_{\text{training}} = \{1, 2, \cdots, m\}$. (3) **All**. $I_{\text{all}} = \{1, 2, \cdots, n\}$. (4) **Random**. For fair comparison, the size of random indices is chosen to be equal to $|I_{\text{khop}}|$. (5) **Laplace-base**. Firstly, we adopt Laplace learning [47] to get the prediction score $u(x_i)$ of each sample. This serves as a base method, not for direct classification, but for deciding the possible interface positions. We calculate the variance of each $u(x_i)$ and pick $|I_{\text{khop}}|$ indices with the smallest variances. The base method

is not limited to Laplace learning, as all classification methods can be possible options. We also test **Poisson-base** by adopting Poisson learning [12] as the base method. **(6) Geodesic.** We define the geodesic distance between two vertices as the sum of edge weights along the shortest path connecting them. we calculate the geodesic distance of other nodes to the training nodes using Dijkstra's algorithm, and choose the farthest $|\mathcal{I}_{\text{khop}}|$ nodes.

We test each method on MNIST, FashionMNIST and CIFAR-10 with 1 labeled sample per class. The results are reported in Table 2. Let's discuss each option one by one: Surprisingly, the so-called "ground truth" option, which leaks label information, does not perform well in real-world classification tasks. This indicates that finding interface positions for high-dimensional multi-class classification is much more complex than for synthetic data. Using the training indices, which is similar to Poisson learning [12], indeed performs similarly to Poisson learning results in Table 1. The key difference is that Poisson learning uses a fixed source term, while

Table 2: Performance of different methods to approximate the interface positions with 1 label per class.

	MNIST	Fashion- MNIST	CIFAR-10
Remove k-hop	93.13 ± 3.72	61.52 ± 5.04	41.71 ± 6.09
Ground truth	90.33 ± 4.60	57.17 ± 5.19	38.45 ± 5.00
Training	90.53 ± 4.14	60.13 ± 4.87	40.55 ± 5.17
All	90.37 ± 4.33	59.96 ± 5.14	38.84 ± 5.01
Random	90.53 ± 4.46	59.89 ± 5.35	39.32 ± 5.43
Laplace-base	93.08 ± 3.54	60.99 ± 5.22	41.84 ± 5.84
Poisson-base	93.62 ± 3.47	61.96 ± 5.88	41.69 ± 6.12
Geodesic	92.85 ± 3.87	61.34 ± 5.17	41.21 ± 6.18

our approach tries to learn the interface term. However, their performance is inferior to the k-hop removal method, suggesting that the given labeled points should be treated as the interior rather than the boundary. Treating all samples as the interface is even worse than learning on a random subset. This indicates that the solution u, while exhibiting discontinuity on the interface, should still maintain a certain level of smoothness in the interior. The Laplace-base and Poisson-base methods are useful as they further improve the classification accuracy by about 0.4% on MNIST and FashionMNIST. This aligns with our intuition because the samples with the smallest prediction variance can be viewed as the hardest samples, and these are often located near the interface. However, in our proposed algorithm, we choose the k-hop removal approach because it is more efficient, easier to understand, and does not rely on other methods. The geodesic index approach, which uses a slightly more advanced version of graph distance, did not show improvements in the results. In conclusion, taking into account the implementation difficulty, classification performance, and execution efficiency, we choose to remove k-hop neighbors to approximate the interface positions.

4.3 Ablation study on parameters

There are two parameters in our experiment: k in k-hop and the ridge parameter λ . In Table 1, we report the best results obtained through a grid search over the model parameters, and the optimal parameters will be provided in Appendix A.2. Here, we will study the effect of these two parameters.

4.3.1 k-hop. We conduct experiments on MNIST and CIFAR-10, reporting the classification accuracy when k ranges from -1 to 4, and the number of labeled samples per class ranges from 1 to 5. Here, k=-1 means that all index are treated as the interface. For each value of k, we report the best performance with respect to the ridge regularization parameter λ . The results are presented in Figure 3.

From the results, it is evident that removing k-hop indices is crucial for classification accuracy. The best accuracy at different label rates significantly outperforms the corresponding accuracy when k=-1. Moreover, we have more observations on the choice of k.

 When k is too large, the performance collapses. This is because the remaining positions for training the interface term are too

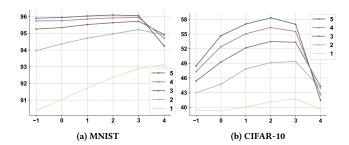


Figure 3: Ablation study on k-hop parameter. x-axis: k-hop. y-axis: accuracy(%). Each line corresponds to a different number of labeled samples per class.

small. For example, on the CIFAR-10 dataset, when the labeled number per class is 1 and k=4, there are only 2730/60000 points beyond the k-hop neighborhood. When the labeled number per class is 5 and k=4, there are only 44/60000 points beyond the k-hop neighborhood. With such a small number of trainable parameters, it may be too challenging for the algorithm to learn a good interface term that generalizes well.

- On the same dataset, the optimal k decreases as the number of labeled samples per class increases. On MNIST, when there is 1 label per class, the optimal k=4. When we increase the label number to 4, k=3 gives the best result. This phenomenon meets our expectation, as we want to keep the number of remaining indices moderate. The former setting gives 38018/70000 remaining indices, while the latter gives 42059/70000 remaining indices, which is comparable.
- For different datasets with the same number of labeled samples per class, the optimal k is different. This is because the connectivity of the datasets varies. Although the similarity graph W is constructed with the same number of nearest neighbors K=10, the underlying data distributions for different datasets are disparate. Hence, even with the same k, the remaining number of nonzero indices after removing the k-hop indices can be very different. For example, when the label number per class is 1 and k = 4, there are 38018/70000 points left for MNIST, but only 2730/60000 points left for CIFAR-10. Such distinctions definitely lead to the different optimal k values for different datasets: the better connectivity of CIFAR-10 suggests that optimal k is smaller.

4.3.2 Ridge. Another important parameter is the ridge regularization parameter λ . In Figure 3, we provide the accuracy with respect to λ , under different values of the k-hop parameter k. The results show that for each k, there exists an optimal λ that yields the best performance. Interestingly, for smaller values of k, the optimal λ tends to be larger. It is also noteworthy that the performance is fairly stable with respect to λ . For example, on the MNIST dataset with k = 4, the accuracy remains above 92.5% for λ values ranging from 0.01 to 0.1, indicating that the accuracy is not sensitive to the choice of λ . We hypothesize that the optimal λ may be related to the scale of the solution u, and investigating this relationship further could be an interesting direction for future work.

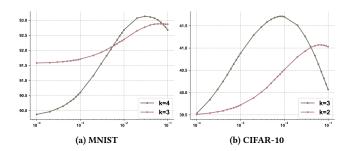


Figure 4: Ablation study on ridge parameter λ with 1 label per class. x-axis: λ . y-axis: accuracy(%).

4.4 Broader application scenarios

In this subsection, we extend the application of our method to more scenarios, including cases with unbalanced label distribution and higher label rates per class.

4.4.1 Unbalanced label distribution. To account for the issue of unbalanced label distribution, we consider the following setup: for the even-numbered classes $\{0, 2, 4, 6, 8\}$, we use 1 labeled sample per class, while for the odd-numbered classes $\{1, 3, 5, 7, 9\}$, we use 5 labeled samples per class.

Other methods, such as Poisson Learning [12], deal with unbalanced training data in a post-processing way by introducing a factor $s_i = b_i/n_i$ to the prediction $u(x_i)$:

$$\underset{j \in \{1,\dots,c\}}{\operatorname{arg\,max}} \{s_j u_j(x_i)\}$$

where b_j is the true fraction of data in class j, and n_j is the number of training examples from class j. On the contrary, Our method naturally learns a more suitable interface term from the unbalanced distribution during the training stage. We adopt the same weighting factor s_j , and apply it as the weight for the MSE loss in our objective function,

$$\frac{1}{m} \sum_{i=1}^{m} s_{y_i} \| (A\mathbf{f})_i - y_i \|_2^2 + \lambda \sum_{i=1}^{n} \| f_i \|_2^2$$

Here, y_i in the expression s_{y_i} refers to the integer-valued class label, not a one-hot encoded vector. We slightly abuse the notation for the sake of simplicity. This weighted loss approach is a widely applied technique when designing loss functions for handling unbalanced training data. Similar to Section 3.3, we can derive the explicit solution of the above objective function.

$$\mathbf{f}_{I}^{*} = \tilde{\mathbf{A}}_{I}^{\top} (\mathbf{S} \tilde{\mathbf{A}}_{I} \tilde{\mathbf{A}}_{I}^{\top} + m \lambda \mathbf{I})^{-1} \mathbf{S} \mathbf{y}$$

where $S = diag(s_{y_i}) \in \mathbb{R}^{m \times m}$. The inference procedure remains unchanged from the previous algorithm, without the need for any additional post-processing steps.

The results are presented in Table 3. We compare our method to Poisson Learning, as it provides an approach to handle unbalanced training data. As a sanity check, both methods outperform their counterparts where only 1 label per class is provided, indicating that the additional labeled samples for the odd-numbered classes are indeed helpful. Importantly, our method outperforms Poisson Learning across all the datasets. This advantage stems from the fact

Table 3: Performance of unbalanced label distribution: even classes 1 label per class, odd classes 5 labels per class.

	MNIST	FashionMNIST	CIFAR-10
Poisson [12]	93.88 ± 2.35	66.47 ± 3.80	46.87 ± 4.16
Inter-Laplace	95.21 ± 2.14	68.16 ± 3.66	49.30 ± 4.72

that we incorporate the class imbalance directly into the objective function, allowing the interface term to learn from the unbalanced distribution, rather than relying on manual post-processing adjustments as in Poisson Learning.

4.4.2 Higher label rate. Although the motivation of interface Laplace learning is to get accurate classification with low label rate, this approach remains effective in higher label rate scenarios. In experiments with 100 labeled samples per class, presented in Table 4, our method outperforms other approaches across the datasets tested. While the gains are modest on simpler datasets like MNIST and FashionMNIST, the improvements become significant on more challenging dataset CIFAR-10. This result demonstrates the robustness of our approach in leveraging available labeled data effectively, regardless of the per-class label rate.

Table 4: Performance of higher label rate: 100 labels per class.

	MNIST	Fashion- MNIST	CIFAR-10
Laplace [47]	96.83 ± 0.10	81.48 ± 0.39	63.88 ± 1.22
Nearest Neighbor	85.58 ± 0.44	71.06 ± 0.55	49.32 ± 0.49
Random Walk [43]	96.63 ± 0.11	81.37 ± 0.30	67.47 ± 0.51
MBO [21]	96.88 ± 0.18	74.72 ± 0.91	42.40 ± 0.78
WNLL [34]	96.25 ± 0.11	81.04 ± 0.30	67.04 ± 0.50
Centered Kernel [31]	88.66 ± 0.72	57.52 ± 2.24	57.30 ± 1.43
Sparse LP [22]	37.76 ± 1.12	28.12 ± 0.66	8.90 ± 0.22
p-Laplace [33]	93.54 ± 0.20	78.10 ± 0.39	63.46 ± 0.38
Poisson [12]	96.77 ± 0.09	80.41 ± 0.62	66.09 ± 0.57
V-Poisson [46]	95.89 ± 0.09	77.85 ± 0.52	41.67 ± 2.17
Inter-Laplace	97.25 ± 0.08	82.74 ± 0.27	$\textbf{73.25} \pm 0.34$

5 Conclusion

Inspired by the observation that interfaces exist between different classes, we propose a Laplace equation model with jump discontinuity and derive its nonlocal counterpart. Based on the nonlocal model, we introduce an interface term to enhance Laplace learning. We then design an effective algorithm to approximate the interface positions and learn the interface term. Experimental results verify that our method can accurately describe the data distribution and improve the performance of semi-supervised learning tasks. Future work involves incorporating the interface concept into neural network architectures, such as graph neural networks, investigating strategies for choosing hyperparameters, and exploring alternative approaches to approximate the interface position.

References

- Matthieu Alfaro and Jérôme Coville. 2017. Propagation phenomena in monostable integro-differential equations: acceleration or not? *Journal of Differential Equations* 263, 9 (2017), 5727–5758.
- [2] Rie K Ando and Tong Zhang. 2007. Learning on graph with Laplacian regularization. In Advances in neural information processing systems. 25–32.
- [3] Zdeněk P Bažant and Milan Jirásek. 2003. Nonlocal integral formulations of plasticity and damage: survey of progress. In Perspectives in Civil Engineering: Commemorating the 150th Anniversary of the American Society of Civil Engineers. ASCE, 21–52.
- [4] Amir Beck and Marc Teboulle. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences 2, 1 (2009), 183–202.
- [5] Mikhail Belkin, Irina Matveeva, and Partha Niyogi. 2004. Regularization and semisupervised learning on large graphs. In Learning Theory: 17th Annual Conference on Learning Theory, COLT 2004, Banff, Canada, July 1-4, 2004. Proceedings 17. Springer, 624–638.
- [6] Mikhail Belkin and Partha Niyogi. 2002. Using manifold stucture for partially labeled classification. Advances in neural information processing systems 15 (2002).
- [7] Sebastien Blandin and Paola Goatin. 2016. Well-posedness of a conservation law with non-local flux arising in traffic flow modeling. *Numer. Math.* 132, 2 (2016), 217–241.
- [8] Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In Proceedings of the eleventh annual conference on Computational learning theory. 92–100.
- [9] Nick Bridle and Xiaojin Zhu. 2013. p-voltages: Laplacian regularization for semisupervised learning on high-dimensional data. In *Eleventh Workshop on Mining* and Learning with Graphs (MLG2013).
- [10] Thomas Bühler and Matthias Hein. 2009. Spectral clustering based on the graph p-Laplacian. In Proceedings of the 26th annual international conference on machine learning. 81–88.
- [11] Jeff Calder. 2022. GraphLearning Python Package. https://doi.org/10.5281/zenodo. 5850940
- [12] Jeff Calder, Brendan Cook, Matthew Thorpe, and Dejan Slepcev. 2020. Poisson learning: Graph based semi-supervised learning at very low label rates. In International Conference on Machine Learning. PMLR, 1306–1316.
- [13] Jeff Calder and Dejan Slepčev. 2020. Properly-weighted graph Laplacian for semisupervised learning. Applied mathematics & optimization 82 (2020), 1111–1159.
- [14] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. 2009. Semi-supervised learning. IEEE Transactions on Neural Networks 20, 3 (2009), 542–542.
- [15] Peter Constantin. 2001. Some open problems and research directions in the mathematical study of fluid dynamics. *Mathematics unlimited—2001 and beyond* (2001), 353–360.
- [16] Kaushik Dayal and Kaushik Bhattacharya. 2007. A real-space non-local phase-field model of ferroelectric domain patterns in complex geometries. Acta materialia 55, 6 (2007), 1907–1917.
- [17] Qiang Du. 2019. Nonlocal Modeling, Analysis, and Computation: Nonlocal Modeling, Analysis, and Computation. SIAM.
- [18] Ahmed El Alaoui, Xiang Cheng, Aaditya Ramdas, Martin J Wainwright, and Michael I Jordan. 2016. Asymptotic behavior of \ell_p-based laplacian regularization in semi-supervised learning. In Conference on Learning Theory. PMLR, 270, 200.
- [19] Heike Emmerich. 2003. The diffuse interface approach in materials science: thermodynamic concepts and applications of phase-field models. Vol. 73. Springer Science & Business Media.
- [20] S Fralick. 1967. Learning to recognize patterns without a teacher. IEEE Transactions on Information Theory 13, 1 (1967), 57–64.
- [21] Cristina Garcia-Cardona, Ekaterina Merkurjev, Andrea L Bertozzi, Arjuna Flenner, and Allon G Percus. 2014. Multiclass data segmentation using diffuse interface methods on graphs. IEEE transactions on pattern analysis and machine intelligence 36, 8 (2014), 1600–1613.
- [22] Alexander Jung, Alfred O Hero III, Alexandru Mara, and Saeed Jahromi. 2016. Semi-supervised learning via sparse label propagation. arXiv preprint arXiv:1612.01414 (2016).
- [23] Feng Kang, Rong Jin, and Rahul Sukthankar. 2006. Correlated label propagation with application to multi-label learning. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), Vol. 2. IEEE, 1719–1726.
- [24] Chiu-Yen Kao, Yuan Lou, and Wenxian Shen. 2010. Random dispersal vs. non-local dispersal. Discrete & Continuous Dynamical Systems 26, 2 (2010), 551.
- [25] Diederik P Kingma and Max Welling. 2022. Auto-Encoding Variational Bayes. arXiv:1312.6114 [stat.ML] https://arxiv.org/abs/1312.6114
- [26] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
- [27] Rasmus Kyng, Anup Rao, Sushant Sachdeva, and Daniel A Spielman. 2015. Algorithms for Lipschitz learning on graphs. In Conference on Learning Theory. PMLR, 1190–1223.

- [28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 11 (1998), 2278–2324.
- [29] Dong C Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. Mathematical programming 45, 1 (1989), 503–528.
- [30] AI Mahan. 1956. Reflection and refraction at oblique incidence on a dielectric-metallic interface as a boundary value problem in electromagnetic theory. *Journal of the Optical Society of America* 46, 11 (1956), 913–926.
- [31] Xiaoyi Mai. 2018. A random matrix analysis and improvement of semi-supervised learning for large dimensional data. *Journal of Machine Learning Research* 19, 79 (2018), 1–27.
- 32] Boaz Nadler, Nathan Srebro, and Xueyuan Zhou. 2009. Semi-supervised learning with the graph laplacian: The limit of infinite unlabelled data. Advances in neural information processing systems 22 (2009), 1330–1338.
- [33] Mauricio Flores Rios, Jeff Calder, and Gilad Lerman. 2019. Algorithms for lp-based semi-supervised learning on graphs. arXiv preprint arXiv:1901.05031 (2019), 1–3.
- [34] Zuoqiang Shi, Stanley Osher, and Wei Zhu. 2017. Weighted nonlocal laplacian on interpolation from sparse data. Journal of Scientific Computing 73 (2017), 1164-1177.
- [35] Wei Shyy and Ranga Narayanan. 1999. Fluid dynamics at interfaces. Cambridge University Press.
- [36] Dejan Slepcev and Matthew Thorpe. 2019. Analysis of p-Laplacian regularization in semisupervised learning. SIAM Journal on Mathematical Analysis 51, 3 (2019), 2085–2120.
- [37] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. 2022. Graph-based semi-supervised learning: A comprehensive review. IEEE Transactions on Neural Networks and Learning Systems 34, 11 (2022), 8174–8194.
- [38] E Stephan. 1983. Solution procedures for interface problems in acoustics and electromagnetics. In *Theoretical acoustics and numerical techniques*. Springer, 291–348.
- [39] Juan Luis Vázquez. 2012. Nonlinear diffusion with fractional Laplacian operators. In Nonlinear partial differential equations. Springer, 271–298.
- [40] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747 [cs.LG] https://arxiv.org/abs/1708.07747
- [41] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semisupervised learning with graph embeddings. In *International conference on ma*chine learning. PMLR, 40–48.
- [42] Yajie Zhang and Zuoqiang Shi. 2021. A nonlocal model of elliptic equation with jump coefficients on manifold. Communications in Mathematical Sciences 19, 7 (2021), 1881–1912.
- [43] Dengyong Zhou and Bernhard Schölkopf. 2004. Learning from labeled and unlabeled data using random walks. In Joint Pattern Recognition Symposium. Springer, 237–244.
- [44] Dengyong Zhou and Bernhard Schölkopf. 2005. Regularization on discrete spaces. In Joint Pattern Recognition Symposium. Springer, 361–368.
- [45] Xueyuan Zhou and Mikhail Belkin. 2011. Semi-supervised learning by higher order regularization. In Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 892–900.
- [46] Xiong Zhou, Xianming Liu, Hao Yu, Jialiang Wang, Zeke Xie, Junjun Jiang, and Xiangyang Ji. 2024. Variance-enlarged Poisson Learning for Graph-based Semi-Supervised Learning with Extremely Sparse Labeled Data. In The Twelfth International Conference on Learning Representations. https://openreview.net/forum?id=yeeVBMDAwy
- [47] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In Proceedings of the 20th International conference on Machine learning (ICML-03). 912–919.
- [48] Xiaojin Zhu and Andrew B Goldberg. 2009. Introduction to semi-supervised learning. Synthesis lectures on artificial intelligence and machine learning 3, 1 (2009), 1–130.

Table 5: Average accuracy scores over 100 trials with standard deviation on new MNIST. See Appendix A.3 for description for new MNIST.

# Label Per Class	1	2	3	4	5
Laplace [47]	17.74 ± 8.80	32.20 ± 12.23	49.47 ± 15.12	66.23 ± 12.80	76.45 ± 10.50
Nearest Neighbor	57.50 ± 4.53	65.87 ± 3.22	70.49 ± 2.51	73.24 ± 2.43	74.98 ± 2.26
Random Walk [43]	85.00 ± 4.28	90.21 ± 2.13	92.53 ± 1.46	93.56 ± 1.23	94.24 ± 0.95
MBO [21]	13.28 ± 8.66	17.10 ± 9.21	22.19 ± 10.77	28.01 ± 10.37	34.14 ± 11.67
WNLL [34]	66.12 ± 14.03	90.51 ± 4.45	94.66 ± 1.54	95.77 ± 0.89	96.20 ± 0.50
Centered Kernel [31]	19.52 ± 1.77	24.94 ± 2.70	29.86 ± 3.06	33.49 ± 3.22	36.72 ± 3.83
Sparse LP [22]	10.02 ± 0.14	9.97 ± 0.22	10.00 ± 0.14	9.94 ± 0.14	9.79 ± 0.13
p-Laplace [33]	69.04 ± 4.79	78.56 ± 2.94	83.16 ± 2.25	85.58 ± 2.04	87.07 ± 1.80
Poisson [12]	93.11 ± 3.87	95.20 ± 1.40	95.93 ± 0.71	96.22 ± 0.57	96.42 ± 0.35
V-Poisson [46]	93.27 ± 4.35	95.49 ± 1.61	96.18 ± 0.48	96.31 ± 0.37	96.43 ± 0.38
Ours	94.91 ± 3.83	96.34 ± 1.20	96.72 ± 0.46	96.80 ± 0.50	96.94 ± 0.36

A Experimental Details

A.1 Time and space complexity

Time Complexity. The main computation burden is the calculation of the iteration matrix \mathbf{A} , which involves T matrix multiplications between a dense $m \times n$ matrix and a sparse $n \times n$ matrix. Matrix multiplication is a highly optimized operation on GPUs. It takes approximately 0.4 seconds to compute \mathbf{A} on a single NVIDIA GeForce RTX 3090Ti GPU. The time consumed by other parts of the algorithm is negligible in comparison.

Table 6: Wall time elapsed (seconds) for each stage, evaluated on MNIST with 1 label per class. A single NVIDIA 3090Ti GPU is used.

	Preprocess		Training	Inference
get T	get_interface_idx()	get A	get f *	get u
0.05	0.0065	0.4	0.0004	0.06

Space Complexity. The storage of the $m \times n$ dense matrix \tilde{A} accounts for the majority of the memory requirements. On a single NVIDIA GeForce RTX 3090Ti GPU with 24GB memory, our method can handle number of labeled samples m up to 10,000.

A.2 Parameters

We provide the set of parameters used to reproduce the results in Table 1. These parameters are selected based on a grid search.

Table 7: Parameters (k, λ) used to reproduce the results in Table 1.

	1	2	3	4	5
MNIST	4, 0.03	3, 0.06	.,	.,	2, 0.04
FashionMNIST	5, 0.06	4, 0.06	4, 0.03	3, 0.06	3, 0.05
CIFAR	3, 0.009	3, 0.002	2, 0.01	2, 0.01	2, 0.007

A.3 V-Poisson reproducibility issue

For the method Variance-enlarged Poisson learning (V-Poisson) proposed in [46], we can only find a zip file on OpenReview, containing only partially reproducible code. The provided Github repository in the paper is empty. So we reproduce the results of V-Poisson by ourselves

- We use a different MNIST distance matrix from that used in V-Poisson. The authors of Poisson learning [12] provide a more fine-tuned distance matrix of MNIST in their GitHub repository two years after the original paper was published. However, since the exact training procedure for this updated distance matrix is not given, we decide to use the original distance matrix for experiments in the main paper. Nonetheless, we also provide the performance comparison on this updated "new MNIST" distance matrix in Table 5. Our method still outperforms other approaches using this updated distance matrix.
- The code for CIFAR-10 is missing in the OpenReview zip file mentioned earlier. The results reproduced by ourselves are not as good as those reported in [46]. Missing of the original code makes it difficult to check the correctness of our reproduced results.

To clarify, we use the same off-the-shelf distance matrix for all methods in our experiments for a fair comparison .

B Additional ablation study on algorithm design

In this section, we provide additional ablation study on the algorithm design, including the MSE objective function, ℓ^2 regularization, and zero mean on each column.

B.1 MSE loss

In multi-class classification problems, cross-entropy loss (CE loss) is among one of the most popular choices of loss function. Specifically, the CE loss between the prediction $u(x_i) \in \mathbb{R}^c$ and integer-valued class label $y_i \in \mathbb{R}$ is defined as

$$\mathcal{L}_{CE} = -\frac{1}{m} \sum_{i=1}^{m} \log \frac{\exp(u(x_i)y_i)}{\sum_{j=1}^{c} \exp(u(x_i)j)}$$

Table 8: Comparison of Mean Squared Error and Cross-Entropy Loss performance.

	# Label	MSE Loss	CE Loss
	1	93.13 ± 3.72	93.01 ± 3.77
	2	95.22 ± 1.01	95.10 ± 1.13
MNIST	3	95.72 ± 0.64	95.67 ± 0.68
	4	95.94 ± 0.49	95.89 ± 0.50
	5	96.09 ± 0.43	96.00 ± 0.38
	1	61.52 ± 5.04	61.12 ± 4.97
	2	68.04 ± 3.44	67.53 ± 3.54
FashionMNIST	3	71.43 ± 2.69	71.17 ± 2.81
	4	72.77 ± 2.40	72.32 ± 2.53
	5	74.12 ± 1.95	73.89 ± 1.97
	1	41.71 ± 6.09	40.43 ± 6.04
	2	49.36 ± 4.20	48.41 ± 4.31
CIFAR-10	3	53.45 ± 3.73	52.20 ± 3.90
	4	56.32 ± 3.23	54.93 ± 3.23
	5	58.30 ± 2.44	57.20 ± 2.52

in which $u(x_i)_j$ denotes the *j*-th component of vector $u(x_i)$. The reason that we finally choose the MSE loss in our algorithm is two-fold:

- If we choose the CE loss, then the optimization problem can no longer be formulated as a ridge regression problem, which in turn has no explicit solution. In this case, we may use a gradient-based optimizer to find the minimizer of the loss function. We choose the L-BFGS method [29] with learning rate of 0.1 for optimization, and find that 5 iterations are suffice for convergence. Nonetheless, this approach requires about 0.4 seconds for training, which is $1000 \times$ slower than calculating the explicit solution for the MSE loss.
- The results using the MSE loss are better than using the CE loss, as shown in Table 8. On MNIST and FashionMNIST datasets, the increase is marginal, but on CIFAR-10 we can see an increase of more than 1%.

B.2 Regularization norm

The ℓ^2 regularizer $\lambda ||f||_2^2$ may be replaced by other regularizers. We test its two alternatives: ℓ^1 norm, which is the sum of absolute value

$$\|f\|_1 = \sum |f_i|$$

and ℓ^0 "norm", which is the number of nonzero elements,

$$||f||_0 = \#\{f_i \neq 0\}$$

Notice that ℓ^0 "norm" is not actually a norm because it is not homogeneous. The LS problem with ℓ^1 regularizer is often called LASSO, while with ℓ^0 regularizer is also known as best subset selection problem. The ℓ^1 and ℓ^0 regularizer can help constrain the sparsity of the learned f. However, unlike ℓ^2 regularization, ℓ^1 and ℓ^0 regularization problem do not have explicit solution in general. Moreover, as ℓ^1 norm is non-differentiable near zero, and ℓ^0 norm is non-differentiable everywhere, it is not suitable to use gradient-based methods to find the corresponding minimizers.

Table 9: Comparison of ℓ^2 , ℓ^1 and ℓ^0 regularization performance on MNIST.

# Label	ℓ^2	ℓ^1	ℓ^0
1	93.13 ± 3.72	89.51 ± 5.90	92.05 ± 4.12
2	95.22 ± 1.01	93.06 ± 2.45	93.89 ± 2.33
3	95.72 ± 0.64	94.31 ± 1.29	94.49 ± 2.03
4	95.94 ± 0.49	94.92 ± 0.95	94.72 ± 1.57
5	96.09 ± 0.43	95.21 ± 0.85	94.70 ± 1.67

To deal with the issue, we use Fast Iterative Shrinkage Thresholding Algorithm (FISTA) [4] to solve the optimization problem. Simply speaking, the idea of the Iterative Shrinkage Thresholding Algorithm (ISTA) is to manually shrink the value smaller than threshold after each gradient descent step on the non-regularized objective. ℓ^1 norm corresponds to a soft thresholding while ℓ^0 norm corresponds to a hard thresholding. Compared to ISTA, FISTA introduces Nesterov acceleration to speed up the convergence.

We report the results using ℓ^1 and ℓ^0 regularizer on MNIST dataset in Table 9. It can be seen that they are not as good as ℓ^2 regularization. Moreover, even though we use FISTA with 10,000 iteration steps, which takes much more time than calculating the explicit solution, we find that FISTA has still not reached the global minimum. Thus, we choose the ℓ^2 regularizer in our algorithm.

B.3 Zero mean

Table 10: Comparison of with (w/) and without (w/o) zero mean technique.

	# Label	$\mathbf{w}/$	w/o
	1	93.13 ± 3.72	92.91 ± 3.89
	2	95.22 ± 1.01	95.11 ± 1.11
MNIST	3	95.72 ± 0.64	95.66 ± 0.70
	4	95.94 ± 0.49	95.90 ± 0.52
	5	96.09 ± 0.43	96.03 ± 0.45
	1	61.52 ± 5.04	60.83 ± 4.87
	2	68.04 ± 3.44	67.58 ± 3.47
FashionMNIST	3	71.43 ± 2.69	71.20 ± 2.81
	4	72.77 ± 2.40	72.50 ± 2.45
	5	74.12 ± 1.95	73.92 ± 1.98
	1	41.71 ± 6.09	41.22 ± 6.13
	2	49.36 ± 4.20	49.09 ± 4.30
CIFAR-10	3	53.45 ± 3.73	53.20 ± 3.89
	4	56.32 ± 3.23	56.00 ± 3.19
	5	58.30 ± 2.44	58.02 ± 2.41

In our algorithm, we add a mean subtraction step to ensure the uniqueness of solution, since the graph Laplacian matrix L is singular and admits infinite solutions. We manually choose the solution with zero mean along each component. To show that the effectiveness of our method does not come solely from extracting the mean, we report the results of our algorithm with and without

mean subtraction step in Table 10. It is observed that the mean subtraction step helps increase the performance by less than 0.5% on average. Notably, when we apply the same mean subtraction step to Poisson learning [12], the performance of Poisson learning with and without this step shows very little difference, with less than 0.05% variation.

C Other tries

C.1 Dirichlet-type interface term

Table 11: Performance of Dirichlet-type interface term with 1 label per class

	MNIST	Fashion- MNIST	CIFAR-10
Laplace [47]	16.73 ± 7.41	18.77 ± 6.54	10.50 ± 1.35
Poisson [12]	90.58 ± 4.07	60.13 ± 4.85	40.43 ± 5.48
Inter-Laplace-D	84.68 ± 4.06	59.18 ± 5.23	37.19 ± 4.44
Inter-Laplace	93.13 ± 3.72	61.52 ± 5.04	41.71 ± 6.09

From the perspective of nonlocal models, we introduce the interface term f_i through the graph Laplacian, i.e., $Lu(x_i) = f_i$. However, as Laplace learning [47] uses a Dirichlet-type boundary condition to incorporate the label information, it is natural to test whether our interface term can also be incorporated in a Dirichlet style. Specifically, we formulate the Dirichlet-type interface problem as the following:

$$Lu(x_i) = 0$$
 $i \notin I$
 $u(x_i) = f_i$ $i \in I$

Similar to Section 3.3, we want to write \mathbf{u} in terms of \mathbf{f} in the form $\mathbf{u} = \mathbf{Af}$. Denote \mathcal{K} as the set of interior indices $i \notin I$. $\mathbf{L}_{\mathcal{K}} \in \mathbb{R}^{|\mathcal{K}| \times |\mathcal{K}|}$ extracts the corresponding rows and columns from the graph Laplacian matrix \mathbf{L} . $\mathbf{W}_{\mathcal{K},I} \in \mathbb{R}^{|\mathcal{K}| \times |I|}$ extracts \mathcal{K} rows and I columns from the similarity matrix \mathbf{W} . Notice that unlike singular matrix \mathbf{L} , $\mathbf{L}_{\mathcal{K}}$ is invertible. Thus we can write the prediction $\mathbf{u}_{\mathcal{K}}$ on $i \in \mathcal{K}$ as:

$$\mathbf{u}_{\mathcal{K}} = \mathbf{L}_{\mathcal{K}}^{-1} \mathbf{W}_{\mathcal{K}, \mathcal{I}} \mathbf{f}_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{K}| \times c}$$

The matrix inversion of $L_{\mathcal{K}}$ is computationally expensive. However, as we only need the m rows corresponding to the labeled indices in $\mathbf{u}_{\mathcal{K}}$ during training, we can instead solve the following equation

$$\tilde{L}_{\mathcal{K}}^{-1}\cdot L_{\mathcal{K}}=\tilde{I}$$

to calculate only the m corresponding rows of $L_{\mathcal{K}}^{-1}$. Here $\tilde{L}_{\mathcal{K}}^{-1}$ and \tilde{I} indicates the rows that correspond to the m training indices of $L_{\mathcal{K}}^{-1}$ and identity matrix I, respectively. Finally, we can write

$$\tilde{\mathbf{A}}_{\mathcal{I}} = \tilde{\mathbf{L}}_{\mathcal{K}}^{-1} \mathbf{W}_{\mathcal{K},\mathcal{I}}$$

We can then learn the interface term \mathbf{f}^* in the same way as in Section 3.3. The algorithm with Dirichlet-type interface term is denoted as Inter-Laplace-D, and we compare its performance with Laplace learning, Poisson learning and our Inter-Laplace in Table 11.

The proposed Inter-Laplace-D method can significantly improve performance compared to Laplace learning. This further substantiate our viewpoint that the function u should exhibit discontinuities

at category interfaces, while remaining smooth within their interiors

However, the performance of Inter-Laplace-D does not quite match the results of Poisson learning and our Inter-Laplace method. The key difference is that Inter-Laplace imposes the interface term through $Lu(x_i)=f_i$, while Inter-Laplace-D imposes it through $u(x_i)=f_i$. We argue that the $Lu(x_i)=f_i$ constraint in Inter-Laplace is a more theoretically-principled approach, as it is derived from nonlocal models. Additionally, this formulation may be a smoother way to incorporate the required discontinuities. By restricting the second derivative $Lu(x_i)$, rather than just the function value $u(x_i)$, Inter-Laplace can more effectively capture the desired discontinuities at category boundaries. This distinction in interface term formulation may help explain why Poisson learning outperforms Laplace learning by a significant margin.

C.2 Neural network parametrized interface term

In our algorithm, we have directly treated the interface term f_i as trainable parameters. However, an alternative approach could be to model f_i as the output of a neural network. Specifically, we could construct a neural network with the extracted feature x_i as input, and the network output $f_{\theta}(x_i)$ representing the interface term, where f_{θ} is the neural network with parameters θ . This neural network-based formulation means that an explicit solution is no longer possible, as neural networks are inherently non-convex. Nonetheless, we can choose to optimize the same objective function as before:

$$\underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \| (Af_{\theta})_i - y_i \|_2^2 + \lambda \sum_{i=1}^{n} \| f_{\theta}(x_i) \|_2^2$$

where $\mathbf{f}_{\theta} = [f_{\theta}(x_1), \dots, f_{\theta}(x_n)]^{\top}$. One may view the addition of neural networks as a way to incorporate feature information, since now the interface term $f_{\theta}(x_i)$ is dependent on the extracted feature x_i . This dependence on the input features can provide additional information to the learned interface term.

We conduct experiments on the MNIST dataset using 1 label per class. We construct a simple 2-layer MLP with a hidden dimension of 64 and ReLU activation function to parametrize f_{θ} . The model is trained for 1000 epochs using the Adam optimizer with a learning rate of 0.01. Without incorporating feature information, our method achieves an average accuracy of 93.13 \pm 3.72%. However, when using a neural network approach, the performance drops to 92.19 \pm 3.93%

One possible explanation for this performance difference is that the feature information may already be sufficiently captured in the construction of the similarity matrix W. Incorporating the same information again through the neural network parametrization of f_{θ} could impose unnecessary restrictions on the learning process. Additionally, using a neural network introduces more hyperparameters, such as the hidden dimension, choice of optimizer, learning rate, and network structure, which may need to be carefully tuned to achieve optimal performance.

Given the slightly worse performance observed when using a neural network to parametrize the interface term f, we opt not

to use a neural network in our approach. However, we hypothesize that feature information might be helpful in graph node-classification tasks, such as the Cora [41] dataset. In these tasks, the graph edge information is often considered orthogonal to the graph node information. Incorporating both the graph structure and the node features may lead to improved performance. We plan to explore this direction as part of our future work.

C.3 Synthetic regression

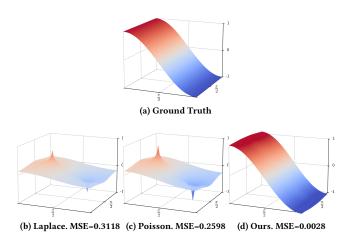


Figure 5: A synthetic regression example.

In Section 2.2, we provide a toy classification example. It is noteworthy that our approach can also be applied to regression problems naturally by setting scalar values as the labels. In this subsection, we provide a comparison of regression results for Laplace learning, Poisson learning and our method on synthetic data.

We uniformly sample a regular 50×50 grid in $(x,y)\in [0,\pi]\times [0,\pi]$. The target is set as $\cos(x)$. We manually pick two labeled points, $(\frac{\pi}{4},\frac{\pi}{2})$ and $(\frac{3\pi}{4},\frac{\pi}{2})$. The similarity matrix is constructed as same as Eq. (1). For Poisson learning and our method, we use T=10,000 to ensure convergence. We pick k=5 and $\lambda=0.05$ in our method. The regression results and corresponding MSE are presented in Figure 5. It is evident from the results that our method significantly outperforms the Laplace learning and Poisson learning approaches. While this particular regression problem does not exhibit a clear interface between distinct classes, it is still beneficial to assume that the underlying function is not harmonic almost everywhere.

D Python code for get_interface_idx()

```
# train_idx : numpy array, shape=[m]
# all_idx
              numpy array, shape=[n]
             scipy sparse matrix, shape=[n, n]
# k
             k hop
import numpy as np
def get_interface_idx(train_idx, all_idx, W, k):
   if k == -1:
        return all_idx
    else:
        khop_idx = train_idx
        for _ in range(k):
            neighbor_idx = W[khop_idx].nonzero()[1]
            khop_idx = np.append(khop_idx, neighbor_idx)
            khop_idx = np.unique(khop_idx)
        interface_idx = np.setdiff1d(all_idx, khop_idx)
   return interface_idx
```