GENERALIZED LABEL PROPAGATION METHODS FOR SEMI-SUPERVISED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The key challenge in semi-supervised learning is how to effectively leverage unlabeled data to improve learning performance. The classical label propagation method, despite its popularity, has limited modeling capability in that it only exploits graph information for making predictions. In this paper, we consider label propagation from a graph signal processing perspective and decompose it into three components: signal, filter, and classifier. By extending the three components, we propose a simple generalized label propagation (GLP) framework for semi-supervised learning. GLP naturally integrates graph and data feature information, and offers the flexibility of selecting appropriate filters and domain-specific classifiers for different applications. Interestingly, GLP also provides new insight into the popular graph convolutional network and elucidates its working mechanisms. Extensive experiments on three citation networks, one knowledge graph, and one image dataset demonstrate the efficiency and effectiveness of GLP.

1 INTRODUCTION

The success of deep learning and neural networks comes at the cost of large amount of training data and long training time. Semi-supervised learning (Zhu, 2005; Chapelle et al., 2006) is interesting and important as it can leverage ample available unlabeled data to aid supervised learning, thus greatly saving the cost, trouble, and time for human labeling. Many researches have shown that when used properly, unlabeled data can significantly improve learning performance (Zhu & Goldberg, 2009; Kingma et al., 2014; Kipf & Welling, 2017). The key challenge for semi-supervised learning is how to effectively leverage the information of unlabeled data, such as graph structures and data features.

Label propagation (Zhu et al., 2003; Zhou et al., 2004; Bengio et al., 2006) is arguably the most popular method for graph-based semi-supervised learning. As a simple and effective tool, it has been widely used in many scientific research fields and has found numerous industrial applications. Given a non-oriented graph $\mathcal{G} = (\mathcal{V}, W, X)$ with $n = |\mathcal{V}|$ vertices, a nonnegative symmetric affinity matrix $W \in \mathbb{R}_+^{n \times n}$ encoding edge weights, and a feature matrix $X \in \mathbb{R}^{n \times m}$ which contains an m-dimensional feature vector of each vertex. For semi-supervised classification, only a small subset of vertices are labeled, and the goal is to predict the labels of other vertices. Denote by $Y \in \{0, 1\}^{n \times l}$ the labeling matrix[1] with l being the number of classes. The objective of of label propagation (LP) is to find a prediction (embedding) matrix $Z \in \mathbb{R}^{n \times l}$ which agrees with Y while being smooth on the graph such that nearby vertices have similar embeddings:

$$Z = \arg\min_{Z} \{ \underbrace{||Z - Y||_2^2}_{\text{Least square fitting}} + \alpha \underbrace{\text{Tr}(Z^\top L Z)}_{\text{Laplcacian regularization}} \}, \tag{1}$$

where α is a balancing parameter, $L = D - W$ is the graph Laplacian[2] and D is the degree matrix. The term enforcing smoothness is called graph Laplacian regularization or Tikhonov regularization. Solving the quadratic regularization framework gives the prediction of LP.

As LP makes predictions only based on graph information (W), its performance depends on whether the underlying graph structure can well represent the class information of data – vertices in the same

[1] If the label of vertex v_i is known, then $Y(i,:)$ is a one-hot embedding of v_i with $y_{ij} = 1$ if v_i belongs to the j-th class and $y_{ij} = 0$ otherwise. If the label of vertex v_i is not given, then $Y(i,:)$ is a vector of all zeros.

[2] Other variants such as the normalized Laplacian matrices are also applicable.

cluster tend to have same labels. For some applications such as social network analysis, data exhibits a natural graph structure. For some other applications such as image or text classification, data may come in a vector form, and a graph is usually constructed using data features. Nevertheless, in many cases, graphs only partially encode data information. Take document classification in a citation network as an example, the citation links between documents form a graph which represents their citation relation, and each document is represented as a bag-of-words feature vector which describes its content. To correctly classify a document, both the citation relations (W) and the content information (X) need to be taken into account, as they contain different aspects of document information. However, in this case, LP can only exploit the graph information to make predictions without using any of the feature information, thus resulting in poor performance.

To go beyond the limit of LP and jointly model graph and feature information, a common approach is to train a supervised learner to classify data features while regularizing the classifier using graph information. Manifold regularization (Belkin et al., 2006) trains a support vector machine with a graph Laplacian regularizer. Deep semi-supervised embedding (Weston et al., 2008) and Planetoid (Yang et al., 2016) train a neural network with an embedding-based regularizer. The recently proposed graph convolutional neural networks (Kipf & Welling, 2017) adopts a different approach by integrating graph and feature information in each of its convolutional layer, which is coupled with a projection layer for classification.

In this paper, we extends the modeling capability of LP in the context of graph signal processing. Casted in the spectral domain, LP can be interpreted as low-pass graph filtering (Ekambaram et al., 2013; Girault et al., 2014). In light of this, we decompose LP into three components: graph signal, graph filter, and classifier. By naturally extending the three components, we propose a generalized label propagation (GLP) framework for semi-supervised learning. In GLP, a low-pass graph filter is applied on vertex features to produce smooth features, which are then fed to a supervised learner for classification. After filtering, the data features within each class are more similar and representative, making it possible to train a good classifier with few labeled examples.

GLP not only extends LP to incorporate vertex features in a simple way, but also offers the flexibility of designing appropriate graph filters and adopting domain-specific classifiers for different semi-supervised applications. The popular graph convolutional networks (GCN) (Kipf & Welling, 2017) is closely related to GLP. In fact, GCN without internal ReLUs is a special case of GLP with a certain graph filter and a multilayer perceptron classifier. When revisited under the GLP framework, it makes clear the working mechanisms of GCN including its design of convolutional filter and model parameter setting. Extensive experiments on citation networks, knowledge graphs, and image datasets show substantial improvement of GLP over GCN and other baselines for semi-supervised classification, confirming the effectiveness of this simple and flexible framework.

The rest of the paper is organized as follows. Section 2 interprets LP in the context of graph signal processing. Section 3 presents the proposed GLP framework. Section 4 revisits GCN under GLP. Section 5 discusses the design of graph filters for GLP. Section 6 presents experimental results. Section 7 discusses related works. Finally, section 8 concludes the paper.

2 A Spectral View of Label Propagation

In this section, we provide a spectral view of LP in the context of graph signal processing.

2.1 Graph Signals and Filters

In graph signal processing (Shuman et al., 2013), the eigenvectors and eigenvalues of the graph Laplacian play the role of Fourier basis and frequencies in parallel with classical harmonic analysis. The graph Laplacian matrix can be eigen-decomposed as: $L = \Phi\Lambda\Phi^{-1}$, where $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_n)$ are the eigenvalues in an increasing order, i.e., $0 = \lambda_1 \leq \cdots \leq \lambda_n$, and $\Phi = (\boldsymbol{\phi}_1, \cdots, \boldsymbol{\phi}_n)$ are the associated orthogonal eigenvectors. Note that the row normalized graph Laplacian $L_r = D^{-1}L$ and the symmetrically normalized graph Laplacian $L_s = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ have similar eigen-decomposition. The eigenvalues $(\lambda_i)_{1\leq i\leq n}$ are interpreted as frequencies and the associated eigenvectors $(\phi_i)_{1\leq i\leq n}$ are interpreted as Fourier basis.

A *graph signal* is a real-valued function $f : \mathcal{V} \to \mathbb{R}$ defined on the vertex set of a graph. Denote by $\boldsymbol{f} = (f(v_1), \cdots, f(v_n))^\top$ a graph signal in a vector form. Consider $(\phi_i)_{1 \le i \le n}$ as basis functions. Any graph signal $\boldsymbol{f}$ can be decomposed into a linear combination of the basis functions:

$$\boldsymbol{f} = \Phi \boldsymbol{c}, \tag{2}$$

where $\boldsymbol{c} = (c_1, \cdots, c_n)^\top$ and c_i is the coefficient of $\boldsymbol{\phi}_i$. The magnitude of the coefficient $|c_i|$ represents the strength of the basis function $\boldsymbol{\phi}_i$ presented in the signal $\boldsymbol{f}$.

A *graph filter* is defined as a matrix $G \in \mathbb{R}^{n \times n}$. G is *linear shift-invariant* (Sandryhaila & Moura, 2013), if and only if there exists an function $p(\cdot) : \mathbb{R} \to \mathbb{R}$, satisfying $G = \Phi p(\Lambda) \Phi^{-1}$, where $p(\Lambda) = \text{diag}(p(\lambda_1), \cdots, p(\lambda_n))$. The real-valued function $p(\cdot)$ is called the *frequency response function* of G.

It is well known that the basis functions associated with lower frequencies (smaller eigenvalues) are smoother (Zhu & Goldberg, 2009), as the smoothness of $\boldsymbol{\phi}_i$ can be measured by λ_i:

$$\sum_{(v_j, v_k) \in \mathcal{E}} w_{jk} [\boldsymbol{\phi}_i(j) - \boldsymbol{\phi}_i(k)]^2 = {\boldsymbol{\phi}_i}^\top L \boldsymbol{\phi}_i = \lambda_i. \tag{3}$$

This indicates that a smooth signal $\boldsymbol{f}$ should contain more low-frequency components than high-frequency components. To produce a smooth signal, the graph filter G should be able to preserve the low-frequency components in $\boldsymbol{f}$ while filtering out the high-frequency components. By Eq. (2), we have

$$\bar{\boldsymbol{f}} = G\boldsymbol{f} = \Phi p(\Lambda) \Phi^{-1} \cdot \Phi \boldsymbol{c} = \sum_i p(\lambda_i) c_i \boldsymbol{\phi}_i. \tag{4}$$

In the filtered signal $\bar{\boldsymbol{f}}$, the coefficient c_i of the basis function $\boldsymbol{\phi}_i$ is scaled by $p(\lambda_i)$. To preserve the low-frequency components and remove the high-frequency components, $p(\lambda_i)$ should amplify c_i when λ_i is small and suppress c_i when λ_i is large. Simply put, $p(\cdot)$ should behave like a *low-pass* filter in classical harmonic analysis.

2.2 Three Components of Label Propagation

The prediction (embedding) matrix of LP can be obtained by taking the derivative of the unconstrained quadratic optimization problem in Eq. (1) and setting it to zero:

$$Z = (I + \alpha L)^{-1} Y. \tag{5}$$

With the prediction matrix Z, each unlabeled vertex v_i is usually classified by simply comparing the elements in $Z(i, :)$. In some methods, a normalization scheme may be applied on the columns of Z first before the comparison (Zhu et al., 2003).

Casted in the context of graph signal processing, LP can be decomposed into three components: signal, filter, and classifier. By Eq. (5), the input signal matrix of LP is the labeling matrix Y, where it has l channels and each column $Y(:, i)$ can be considered as a graph *signal*. In $Y(:, i)$, only the labeled vertices in class i have value 1 and others 0.

The graph *filter* used in LP is

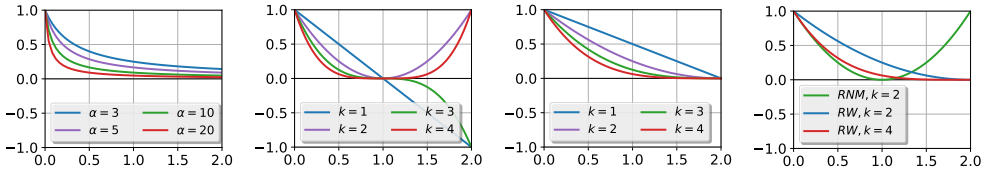
$$(I + \alpha L)^{-1} = [\Phi(I + \alpha\Lambda)\Phi^{-1}]^{-1} = \Phi(I + \alpha\Lambda)^{-1}\Phi^{-1},$$

with frequency response function

$$p(\lambda_i) = \frac{1}{1 + \alpha\lambda_i}. \tag{6}$$

Note that this also holds for the normalized graph Laplacians. As shown in Fig. 1(a), the frequency response function of LP is low-pass. For any $\alpha > 0$, $p(\lambda_i)$ is near 1 when λ_i is close to 0 and $p(\lambda_i)$ decreases and approaches 0 as λ_i increases. Applying the filter on signal $Y(:, i)$, it will produce a smooth signal $Z(:, i)$ in which vertices of the same class have similar values and vertices in class i have larger values than others under the cluster assumption. The balancing parameter α controls the degree of the graph Laplacian regularization. When α increases, the filter becomes more low-pass (Fig. 1(a)) and will produce smoother embeddings.

Finally, LP applies a nonparametric *classifier* on the embeddings to classify the unlabeled vertices, i.e., the label of an unlabeled vertex v_i is given by $y_i = \arg\max_j Z(i, j)$.



(a) AR: $(1+\alpha\lambda)^{-1}$ (b) RNM: $(1-\lambda)^k$ (c) RW: $(1-\frac{1}{2}\lambda)^k$ (d) RNM and RW

Figure 1: Response functions of AR, RNM, and RW filters, and their comparison.

3 Generalized Label Propagation Methods

We propose a generalized label propagation (GLP) framework by naturally generalizing the three components of LP for semi-supervised classification:

- Signal: Use the feature matrix X instead of the labeling matrix Y as input signal.
- Filter: The filter G can be any low-pass, linear, shift-invariant filter.
- Classifier: The classifier can be any classifer trained on the embeddings of labeled vertices.

GLP consists of two steps. First, a low-pass, linear, shift-invariant graph filter G is applied on the feature matrix X to obtain a smooth feature matrix $\bar{X} \in \mathbb{R}^{n\times m}$:

$$\bar{X} = GX. \tag{7}$$

The next step is to train a supervised classifier (e.g., multilayer perceptron, convolutional neural networks, support vector machines, etc.) with the filtered features of labeled data, and then apply the classifier on the filtered features of unlabeled data to predict their labels.

GLP naturally combines graph and feature information in Eq. (7), and allows taking advantage of a powerful supervised classifier. The rationale behind GLP is to learn representative feature vectors of each class for easing the downstream classification task. After filtered by G, vertices in the same class are expected to have more similar and representative features, which makes it much easier to train a good classifier with very few samples.

Consider an extreme case that each class is a connected component of the graph. In this case, we can learn perfect features by an extremely low-pass filter G, whose spectrum $p(\cdot)$ is unit impulse function, i.e., $p(0) = 1$ and $p(\lambda) = 0$ if $\lambda \neq 0$. We can compute $G = \Phi p(\Lambda)\Phi^{-1}$ in the spatial domain. In particular, $G_{ij} = \frac{1}{l_k}$ if v_i and v_j are of the same class, otherwise $G_{ij} = 0$, where l_k is the number of labeled samples in class k. After filtered by G, vertices in the same class will have an identical feature vector which is its class mean. Then any classifier that can correctly classify the labeled data will achieve 100% accuracy on the unlabeled data, and only one labeled example per class is needed to train the classifier.

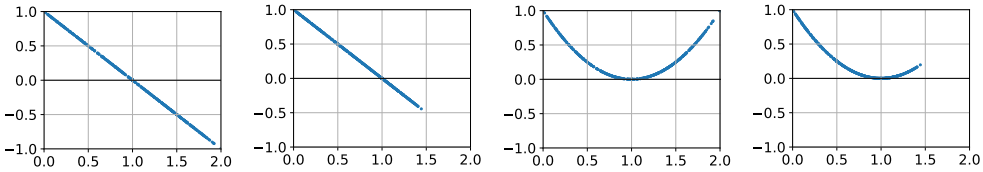
4 Revisit Graph Convolutional Networks

In this section, we show that the graph convolutional networks (GCN) (Kipf & Welling, 2017) for semi-supervised classification can be interpreted under the GLP framework, which explains its implicit design features including the number of layers, the choice of the normalized graph Laplacian, and the renormalization trick on the convolutional filter.

Graph Convolutional Networks. The GCN model contains three steps. First, a renormalization trick is applied on the adjacency matrix W by adding an self-loop to each vertex, which results in a new adjacency matrix $\tilde{W} = W + I$ with the degree matrix $\tilde{D} = D + I$. After that, symmetrically normalize $\tilde{W}$ and get $\tilde{W}_s = \tilde{D}^{-\frac{1}{2}}\tilde{W}\tilde{D}^{-\frac{1}{2}}$. Second, define the propagation rule

$$H^{(t+1)} = \sigma\left(\tilde{W}_s H^{(t)}\Theta^{(t)}\right), \tag{8}$$

where $H^{(t)}$ is the matrix of activations in the t-th layer and $H^{(0)} = X$, $\Theta^{(t)}$ is the trainable weight matrix in layer t, and σ is the activation function, e.g., $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. The graph convolution



(a) $1-\lambda$ (b) $1-\tilde{\lambda}$ (c) $(1-\lambda)^2$ (d) $(1-\tilde{\lambda})^2$

Figure 2: Effect of the renormalization trick. Left two figures plot points $(\lambda_i, p(\lambda_i))$. Right two figures plot points $(\tilde{\lambda}_i, p(\tilde{\lambda}_i))$.

is defined by multiplying the input of each layer with the renormalized adjacency matrix $\tilde{W}_s$ from the left, i.e., $\tilde{W}_s H^{(t)}$. The convoluted features are then fed into a projection matrix $\Theta^{(t)}$. Third, stack two layers up and apply a softmax function on the output features to produce a prediction matrix:

$$Z = \text{softmax}\left(\tilde{W}_s \text{ ReLU}\left(\tilde{W}_s X \Theta^{(0)}\right)\Theta^{(1)}\right), \tag{9}$$

and train the model using the cross-entropy loss over the labeled instances.

4.1 Link to Generalized Label Propagation Methods

The graph convolution in each layer of the GCN model actually performs feature smoothing with a low-pass filter $\tilde{W}_s = I - \tilde{L}_s$, where $\tilde{L}_s$ is the symmetrically normalized graph Laplacian of the graph with extra self-loops. Suppose that $\tilde{L}_s$ can be eigen-decomposed as $\tilde{L}_s = \Phi\tilde{\Lambda}\Phi^{-1}$, then we have $I - \tilde{L}_s = \Phi(I - \tilde{\Lambda})\Phi^{-1}$. The frequency response function of the filter is

$$p(\tilde{\lambda}_i) = 1 - \tilde{\lambda}_i. \tag{10}$$

Clearly, as shown in Fig. 1(b), this function is linear and low-pass on the interval [0, 1], but not on [1, 2], as it amplifies the eigenvalues near 2.

Interestingly, by removing the activation function ReLU in Eq. (9), we can see that GCN is a special case of GLP, where the input signal is X, the filter is $\tilde{W}_s^2$, and the classifier is a two-layer multi-layer perceptron (MLP).

Why the Normalized Graph Laplacian. Note that the eigenvalues of the normalized Laplacians L_s and L_r all fall into interval $[0, 2]$ (Chung, 1997), while the unnormalized Laplacian L has eigenvalues in $[0, +\infty]$. If using the unnormalized graph Laplacian, the response function in Eq. (10) will amplify eigenvalues in $[2, +\infty]$, which will introduce noise and undermine performance.

Why Two Convolutional Layers. In Eq. (9), the GCN model stacks two convolutional layers. Without the activation function, the feature matrix is actually filtered by $I - \tilde{L}_s$ twice, which is equivalent to be filtered by $(I - \tilde{L}_s)^2$ with response function $(1-\lambda)^2$. As we can see from Fig. 1(b), $(1-\lambda)^2$ is more low-pass than $(1-\lambda)$ by suppressing the eigenvalues in the mid-range of [0, 2] harder, which explains why GCNs with two convolutional layers perform better than those with only one.

Why the Renormalization Trick. The effect of the renormalization trick is illustrated in Fig. 2, where the response functions on the eigenvalues of L_s and $\tilde{L}_s$ on the Cora citation network are plotted. We can see that by adding a self-loop to each vertex, the range of eigenvalues shrink from [0, 2] to [0, 1.5], thus avoiding amplifying eigenvalues near 2 and reducing noise. This explains why the renormalization trick works.

5 Filter Design and Computation

In this section, we discuss the design and computation of low-pass graph filters for GLP.

Auto-Regressive. The Auto-Regressive (AR) filter is the one used in LP:

$$p_{\text{ar}}(L) = (I + \alpha L)^{-1}. \tag{11}$$

Actually p_{ar} is an auto-regressive filter of order one (Tremblay et al., 2018). We have shown p_{ar} is low-pass in section 2.2. However, the computation of p_{ar} involves matrix inversion, which is also computationally expensive with complexity $\mathcal{O}(n^3)$. Fortunately, we can circumvent this problem by approximating p_{ar} using its polynomial expansion:

$$(I + \alpha L)^{-1} = \frac{1}{1+\alpha} \sum_{i=0}^{+\infty} \left[\frac{\alpha}{1+\alpha} W \right]^i, (\alpha > 0). \tag{12}$$

We can then compute $\bar{X} = p_{\text{ar}}(L)X$ iteratively with

$$X'^{(0)} = \mathbf{O}, \cdots, X'^{(i+1)} = X + \frac{\alpha}{1+\alpha} W X'^{(i)},$$

and let $\bar{X} = \frac{1}{1+\alpha} X'^{(k)}$. Empirically, we find that $k = \lceil 4\alpha \rceil$ is enough to get a good approximation. Hence, the computational complexity is reduced to $\mathcal{O}(nm\alpha + Nm\alpha)$ (note that X is of size $n \times m$), where N is the number of nonzero entries in L, and $N \ll n^2$ when the graph is sparse.

Renormalization. The renormalization (RNM) filter is an exponential function of the renormalized adjacency filter used in GCN:

$$p_{\text{rnm}}(\tilde{L}) = \left(I - \tilde{L} \right)^k. \tag{13}$$

We have shown in section 4.1 that although the response function $p_{\text{rnm}}(\lambda) = (1-\lambda)^k$ is not low-pass, the renormalization trick shrinks the range of eigenvalues of $\tilde{L}$ and makes p_{rnm} resemble a low-pass filter. The exponent parameter k controls the low-pass effect of p_{rnm}. When $k = 0$, p_{rnm} is all-pass. When k increases, p_{rnm} becomes more low-pass. Note that for a sparse graph, $(I - \tilde{L})$ is a sparse matrix. Hence, the fastest way to compute $\bar{X} = p_{\text{rnm}}(\tilde{L})X$ is to left multiply X by $(I - \tilde{L})$ repeatedly for k times, which has the computational complexity $\mathcal{O}(Nmk)$.

Random Walk. We also propose to design a random walk (RW) filter:

$$p_{\text{rw}}(L_r) = \left(\frac{I + D^{-1}W}{2} \right)^k = \left(I - \frac{1}{2} L_r \right)^k. \tag{14}$$

We call p_{rw} the random walk filter because $\left(\frac{I+D^{-1}W}{2} \right)$ is a stochastic matrix of a lazy random walk which at each step returns to the current state with probability $\frac{1}{2}$, and $\left(\frac{I+D^{-1}W}{2} \right)^k$ is the k-step transition probability matrix. Similarly, we can derive the response function of p_{rw} as

$$p_{\text{rw}}(\lambda) = (1 - \frac{1}{2}\lambda)^k. \tag{15}$$

Note that L_r has the same eigenvalues with L_s, with range [0, 2]. Unlike the RNM, p_{rw} is a typical low-pass filter on [0, 2], as shown in Fig. 1(c). We can also see in Fig. 1(d) that the curves of $(1-\lambda)^2$ and $(1 - \frac{1}{2}\lambda)^4$ are very close, implying that to have the same level of low-pass effect, k in p_{rw} should be set twice as large as in p_{rnm}. This may be explained by the fact that the two functions $(1-\lambda)^k$ and $(1 - \frac{1}{2}\lambda)^{2k}$ have the same derivative k at $\lambda = 0$. On the computation side, RW has the same complexity $\mathcal{O}(Nmk)$ as RNM.

An important issue of filter design for GLP is how to control the strength of filters by setting parameters such as α and k. Intuitively, when labeled data is scarce, it would be desirable for the filtered features of each instance to be closer to its class mean and be more representative of its own class. Hence, in this case, α and k should be set large to produce smoother features. However, over-smoothing usually results in inaccurate class boundaries. Therefore, when the amount of labeled data is reasonably large, α and k should be set relatively small to preserve feature diversity in order to learn more accurate class boundaries.

6 EXPERIMENTS

Datasets In this section, we test GLP on three citation networks – Cora, CiteSeer and PubMed (Sen et al., 2008), one knowledge graph – NELL (Carlson et al., 2010), and one handwritten digit image dataset – MNIST (LeCun et al., 1998). Dataset discriptions are provided in Appendix A.

Table 1: Classification Accuracy and running time on citation networks and NELL.

Label Rate	20 labels per class			4 labels per class			10%	1%	0.1%
	Cora	CiteSeer	PubMed	Cora	CiteSeer	PubMed	NELL		
ManiReg	59.5	60.1	70.7	-	-	-	63.4	41.3	21.8
SemiEmb	59.0	59.6	71.7	-	-	-	65.4	43.8	26.7
DeepWalk	67.2	43.2	65.3	-	-	-	79.5	72.5	58.1
ICA	75.1	**69.1**	73.9	62.2	49.6	57.4	-	-	-
Planetoid	75.7	64.7	**77.2**	43.2	47.8	64.0	**84.5**	**75.7**	**61.9**
MLP	56.7 (0.5s)	55.9 (0.6s)	69.1 (0.5s)	37.8 (0.5s)	39.6 (0.6s)	57.6 (0.5s)	63.7 (9.4s)	41.3 (4.9s)	21.5 (7.1s)
LP	67.8 (0.4s)	43.9 (0.3s)	66.4 (1.2s)	60.9 (0.1s)	40.0 (0.2s)	62.6 (4.3s)	71.4	44.8	26.5
GCN	**79.5** (1.5s)	**68.7** (2.3s)	**77.2** (16s)	64.0 (1.8s)	56.4 (2.5s)	66.7 (17s)	81.6 / 82.1 (39s)	62.9 / 67.7 (33s)	39.1 / 58.2 (40s)
GLP (RNM, MLP)	**79.1** (0.5s)	67.6 (0.6s)	**77.2** (0.5s)	**68.4** (0.5s)	**57.5** (0.6s)	**67.2** (0.6s)	**85.2** / 85.9 (49s)	**77.4** / 78.8 (22s)	**64.5** / 68.1 (26s)
GLP (RW, MLP)	**79.1** (0.5s)	67.7 (0.6s)	77.0 (0.5s)	**68.5** (0.5s)	**57.6** (0.6s)	**67.2** (0.6s)	**85.3** / 86.6 (87s)	**77.4** / 79.5 (35s)	**64.9** / 70.5 (39s)
GLP (AR, MLP)	**80.3** (0.5s)	**68.3** (0.7s)	**78.3** (0.5s)	**69.4** (0.7s)	**58.3** (0.9s)	**68.7** (0.8s)	84.2 / 84.0 (487s)	63.9 / 66.1 (375s)	59.9 / 62.9 (637s)

Baselines On citation networks and NELL, we compare GLP against GCN (Kipf & Welling, 2017), LP (Wu et al., 2012), multi-layer perceptron (MLP), Planetoid (Yang et al., 2016), DeepWalk (Perozzi et al., 2014), manifold regularization (ManiReg) (Belkin et al., 2006), semi-supervised embedding (SemiEmb) (Weston et al., 2008), and iterative classification algorithm (ICA) (Sen et al., 2008). On MNIST, we compare GLP against GCN, LP, MLP, and convolutional neural networks (CNN).

Experimental Setup We test GLP with RNM, RW and AR filters (section 5) on all the datasets. We use MLP as the classifier for GLP on citation networks and NELL, and use CNN as the classifier on MNIST. Guided by our analysis in section 5, the filter parameters k and α should be set large with small label rate and set small with large label rate. We use fixed parameters $k = 10$ for RNM, $k = 20$ for RW, and $\alpha = 20$ for AR when label rate is less than or equal to 2%, and set them to $5, 10, 10$ respectively otherwise. We follow Kipf & Welling (2017) to set the parameters of MLP, including learning rate, dropout, and weight decay. To make sure GLP works in practice and for more fair comparison with baselines, we do not use a validation set for classifier model selection as in Kipf & Welling (2017), instead we select the classifier with the highest training accuracy in 200 steps. Results of GLP and GCN on all the datasets are reported without using a validation set, except that on NELL, we also report the results with validation (on the right of "/"). More implementation details are provided in Appendix B due to space limitations.

Performance of GLP The results are summarized in Tables 1 and 2, where the top 3 classification accuracies are highlighted in bold. Overall, GLP performs the best on all the datasets. On citation networks, with 20 labels per class, GLP performs comparably with GCN and outperforms other baselines by a considerable margin. With 4 labels per class, GLP significantly outperforms all baselines including GCN. On NELL, GLP wtih RW and RNM filters consistently outperforms the best baseline Planetoid for each setting, and outperforms other baselines including GCN by a large margin. Note that GLP achieves this performance without using any additional validation set. The performance of GLP (and GCN) will be further boosted with validation, as shown on the right of "/". On MNIST, GLP consistently outperforms all baselines for every setting.

The running times of GLP and some other baselines are also reported in Tables 1 and 2. GLP runs much faster than GCN on most datasets, except for NELL, on which the running times of GLP with two filters are similar with GCN. More discussions about running times are included in Appendix E.

Results Analysis Compared with LP and DeepWalk which only use graph information, the large performance gains of GLP clearly comes from leveraging both graph and feature information. Compared with purely supervised MLP and CNN which are trained on raw features, the performance gains of GLP come from the unsupervised feature filtering. Fig. 3 visualizes the raw and filtered features (by RNM filter) of Cora projected by t-SNE (Van der Maaten & Hinton, 2008). The filtered features exhibit a much more compact cluster structure, thus making classification much easier. In Appendix C, we show that feature filtering improves the accuracy of various classifiers significantly.

Compared with GCN and other baselines which use both graph and feature information, the performance gains of GLP come in two folds. First, GLP allows using stronger filters to extract higher level data representations to improve performance when label rate is low, which can be easily achieved by increasing the filter parameters k and α, as shown in Fig. 3. But this cannot be easily achieved in

Labels per class	10	20	30
MLP	66.1 (0.5s)	74.9 (0.5s)	78.5 (0.6s)
CNN	86.3 (2.5s)	92.1 (4.1s)	94.3 (6.3s)
LP	92.9 (0.2s)	94.4 (0.2s)	95.1 (0.2s)
GCN	82.4 (143.0s)	88.2 (148.0s)	90.3 (147.0s)
GLP (RNM, CNN)	**94.1** (4.9s)	**95.3** (6.6s)	**95.6** (8.7s)
GLP (RW, CNN)	**94.1** (5.1s)	**95.3** (6.7s)	**95.6** (8.9s)
GLP (AR, CNN)	**94.1** (7.2s)	**95.5** (8.8s)	**95.8** (11.1s)

Table 2: Classification accuracy and running time on MNIST.

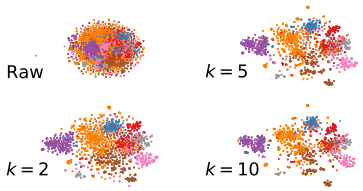


Figure 3: Visualization of raw and filtered features of Cora.

GCN. As each convolutional layer of GCN is coupled with a projection layer, to increase smoothness one needs to stack many layers, and a deep GCN is difficult to train. Second, GLP allows adopting domain-specific classifiers such as CNN to deal with vision tasks. As shown in Table 2, the performance of CNN trained on raw features of labeled data is very competitive and grows fast.

Due to space limitations, we include the stability analysis of GLP in Appendix D.

7 Related Works

Many graph-based semi-supervised learning methods adopt a common assumption that nearby vertices are likely to have same labels. One idea is to learn smooth low-dimensional embedding of data points by using Markov random walks (Szummer & Jaakkola, 2002), Laplacian eigenmaps (Belkin & Niyogi, 2004), spectral kernels (Chapelle et al., 2003; Zhang & Ando, 2006), and context-based methods (Perozzi et al., 2014). Another idea hinges on graph partition, where the cuts should agree with the labeled data and be placed in low density regions (Blum & Chawla, 2001; Zhu et al., 2003; Joachims, 2003; Blum et al., 2004). Perhaps the most popular idea is to formulate a quadratic regularization framework to explicitly enforce the consistency with the labeled data and the cluster assumption, which is known as label propagation (Zhou et al., 2004; Chapelle & Zien, 2005; Bengio et al., 2006; Kveton et al., 2010).

To leverage more data information to improve predictions, a variety of methods proposed to jointly model data feature and graph information. Zhou et al. (2004) proposed to combine label propagation with external classifiers by attaching a "dongle" vertex to each unlabeled vertex. Iterative classification algorithm (Sen et al., 2008) iteratively classifies an unlabeled vertex using its neighbors' labels and features. Manifold regularization (Belkin et al., 2006), deep semi-supervised embedding (Weston et al., 2008), and Planetoid (Yang et al., 2016) regularize a supervised classifier with a Laplacian regularizer or an embedding-based regularizer. Graph convolutional networks (Kipf & Welling, 2017) combine graph and feature information in convolutional layers, which is actually doing Laplacian smoothing on data features (Li et al., 2018). Follow-up works include graph attention networks (Velikovi et al., 2018), attention-based graph neural network (Thekumparampil et al., 2018), and graph partition neural networks (Liao et al., 2018).

The idea of feature smoothing has been widely used in computer graphics community for fairing 3D surface (Taubin, 1995b;a; Desbrun et al., 1999). Hein & Maier (2007) proposed manifold denoising which uses feature smoothing as a preprocessing step for running a label propagation algorithm, i.e, the denoised features are used to construct a better graph for LP. This method is still "one-dimensional", as it cannot use the preexisting graph information in data such as citation networks. In contrast, the proposed GLP and the GCN frameworks are "two-dimensional".

8 Conclusion

In this paper, we have proposed a simple, flexible, and efficient framework GLP for semi-supervised learning, and demonstrated its effectiveness theoretically and empirically. GLP offers new insights into existing methods and opens up possible avenues for new methods. An important direction for future research is the design and selection of graph filters for GLP in different application scenarios. Other directions include making GLP readily applicable to inductive problems, developing faster algorithms for GLP, and applying GLP to solve large-scale real-world problems.

REFERENCES

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

M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7:2434, 2006.

Y. Bengio, O. Delalleau, and N. Le Roux. Label propagation and quadratic criterion. *Semi-supervised learning*, pp. 193–216, 2006.

A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *International Conference on Machine Learning*, pp. 19–26. ACM, 2001.

A. Blum, J. Lafferty, M.R. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *International Conference on Machine Learning*, pp. 13. ACM, 2004.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI Conference on Artificial Intelligence*, volume 5, pp. 3, 2010.

O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pp. 57–64. Max-Planck-Gesellschaft, 2005.

O. Chapelle, J. Weston, and B. Scholkopf. Cluster kernels for semi-supervised learning. In *Conference on Neural Information Processing Systems*, 2003.

O. Chapelle, B. Schölkopf, A. Zien, et al. *Semi-supervised learning*. MIT Press, 2006.

Fan RK Chung. *Spectral graph theory*. American Mathematical Soc., 1997.

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

Venkatesan N Ekambaram, Giulia Fanti, Babak Ayazifar, and Kannan Ramchandran. Wavelet-regularized graph semi-supervised learning. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pp. 423–426. IEEE, 2013.

Benjamin Girault, Paulo Gonçalves, Eric Fleury, and Arashpreet Singh Mor. Semi-supervised learning for graph to signal mapping: A graph signal wiener filter interpretation. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 1115–1119. IEEE, 2014.

Matthias Hein and Markus Maier. Manifold denoising. In *Conference on Neural Information Processing Systems*, 2007.

T. Joachims. Transductive learning via spectral graph partitioning. In *International Conference on Machine Learning*, pp. 290–297. ACM, 2003.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pp. 3581–3589, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

B. Kveton, M. Valko, A. Rahimi, and L. Huang. Semisupervised learning with max-margin graph cuts. In *AISTATS*, pp. 421–428, 2010.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Association for the Advancement of Artificial Intelligence 2018*. AAAI, 2018.

Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L Gaunt, Raquel Urtasun, and Richard Zemel. Graph partition neural networks for semi-supervised classification. *arXiv preprint arXiv:1803.06272*, 2018.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656, 2013.

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.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

M. Szummer and T. Jaakkola. Partially labeled classification with Markov random walks. In *Conference on Neural Information Processing Systems*, pp. 945–952, 2002.

Gabriel Taubin. Curve and surface smoothing without shrinkage. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pp. 852–857. IEEE, 1995a.

Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 351–358. ACM, 1995b.

Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

Nicolas Tremblay, Paulo Gonçalves, and Pierre Borgnat. Design of graph filters and filterbanks. In *Cooperative and Graph Signal Processing*, pp. 299–324. Elsevier, 2018.

L.J.P. Van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th international conference on Machine learning*, pp. 1168–1175. ACM, 2008.

Xiaoming Wu, Zhenguo Li, Anthony M. So, John Wright, and Shih-fu Chang. Learning with Partially Absorbing Random Walks. In *Conference on Neural Information Processing Systems*, pp. 3077–3085. Curran Associates, Inc., 2012.

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International conference on Machine learning*, pp. 40–48. ACM, 2016.

T. Zhang and R.K. Ando. Analysis of spectral kernel design based semi-supervised learning. In *Conference on Neural Information Processing Systems*, pp. 1601–1608, 2006.

Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Conference on Neural Information Processing Systems*, pp. 321–328. MIT Press, 2004.

X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

X. Zhu and A.B. Goldberg. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130, 2009.

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning 2003*, pp. 912–919. ACM, 2003.

Appendices

We include dataset descriptions, experimental details, supplementary experiments, stability analysis, and running time analysis here.

APPENDIX A DATASET DESCRIPTION

Citation networks (Sen et al., 2008) are networks that record documents' citation relationship. In citation networks, vertices are documents and edges are citation links. A pair of vertices are connected by an undirected edge if and only if one cites another. Each vertex is associated with a feature vector, which encodes the document content. In the three citation networks we tested on, CiteSeer, Cora and PubMed, feature vectors are 0/1 vectors that have the same length as the dictionary size and indicate whether a word appears in a document. The statistics of datasets are summarized in Table 3.

Never Ending Language Learning (NELL) (Carlson et al., 2010) is a knowledge graph introduced by Carlson et al.. Yang et al. extracted an entity classification dataset from NELL, and converted the knowledge graph into a single relation graph. For each relation type r, they created two new vertices r_1 and r_2 in the graph. For each triplet (e_1, r, e_2), they created two edges (e_1, r_1) and (e_2, r_2). We follow Kipf & Welling (2017) to extend the features by assigning a unique one-hot representation for every relation vertex, resulting in a 61,278-dimensional sparse feature vector for each vertex. Dataset statistics are also provided in Table 3.

MNIST contains 70,000 images of handwritten digits from 0 to 9 of size 28×28. Each image is represented by a dense 784-dimensional vector where each dimension is a gray intensity pixel value. A 5-NN graph is constructed based on the Euclidean distance between images. If the i-th image is within the j-th image's 5 nearest neighbors or vice versa, then $w_{ij} = w_{ji} = 1$, otherwise $w_{ij} = w_{ji} = 0$.

Table 3: Dataset statistics.

Dataset	Type	Vertices	Edges	Classes	Features
CiteSeer	Citation network	3,327	4,732	6	3703
Cora	Citation network	2,708	5,429	7	1433
PubMed	Citation network	19,717	44,338	3	500
NELL	Knowledge graph	65,755	266,144	210	5414
MNIST	Images	70,000	2,060,504	10	784

APPENDIX B EXPERIMENTAL DETAILS

We provide more experimental details here for the sake of reproduction.

Parameters We set $k = 10$ for RNM, $k = 20$ for RW, and $\alpha = 20$ for AR, if label rate is less or equal than 2%; otherwise, we set them to $5, 10, 10$ respectively.

Networks On citation networks, we follow Kipf & Welling (2017) to use a two-layer MLP with 16 hidden units for citation networks, 0.01 learning rate, 0.5 dropout rate, and 5×10^{-4} L2 regularization. On NELL, we also follow Kipf & Welling (2017) to use 64 hidden units, 10^{-5} L2 regularization, 0.1 dropout rate and two layer-structure. On MNIST, we use 256 hidden units, 0.01 learning rate, 0.5 dropout rate, and 5×10^{-4} L2 regularization. The CNN we use consists of six layers, whose structure is specified in Table 4. For CNN, we use 0.003 learning rate and 0.5 dropout. All results of MNIST are averaged over 10 runs. We train all networks using Adam (Kingma & Ba, 2014).

Baselines Results of some baselines including ManiReg, SemiEmb, DeepWalk, ICA, Planetoid are taken from Kipf & Welling (2017), except for the 4-labels-per-class setting, for which we run

Table 4: CNN structure.

Layer Structure	Output Size
Conv. $5 \times 5 \times 32$ with BN	$28 \times 28 \times 32$
Max-pooling 2×2 with BN	$14 \times 14 \times 32$
Conv. $3 \times 3 \times 64$ with BN	$14 \times 14 \times 64$
Conv. $3 \times 3 \times 64$ with BN	$14 \times 14 \times 64$
Max-pooling 2×2 with BN	$7 \times 7 \times 64$
Conv. $3 \times 3 \times 128$ with BN	$7 \times 7 \times 128$
Conv. $1 \times 1 \times 10$ with BN	$7 \times 7 \times 10$
Drop out 0.5	
FC 490×10 with BN	10

ICA[2] implemented by Kipf and Planetoid implemented by Yang with their choices of parameters. Results of LP on NELL are also taken from Yang et al. (2016). All other results are reported by us.

APPENDIX C GLP WITH VARIOUS CLASSIFIERS

To demonstrate the benefit of GLP, we compare training various supervised classifiers with raw and filtered features. The classifiers include support vector machine (SVM), decision tree (DT), logistic regression (LR), and multilayer perceptron (MLP). The results are summarized in Table 5. We can see that for all classifiers and on all datasets, there is a huge improvement in classification accuracy with the smooth features produced by the three filters we proposed. This clearly demonstrates the advantage of filtered features over raw features.

Table 5: Classification accuracy with 20 labels per Class.

	Cora				CiteSeer				PubMed			
	SVM	DT	LR	MLP	SVM	DT	LR	MLP	SVM	DT	LR	MLP
Raw Features	22.3	45.2	54.6	56.7	50.0	42.2	58.8	55.9	63.2	58.0	68.9	69.1
GLP with RNM	55.3	63.5	74.3	79.1	61.1	51.9	66.6	67.6	68.5	66.8	74.2	77.2
GLP with RW	55.3	64.2	74.3	79.1	62.4	50.3	66.5	67.7	68.5	67.2	74.1	77.0
GLP with AR	51.3	63.3	74.1	80.3	62.0	51.7	67.1	68.3	70.3	67.1	74.6	78.3

In this experiment, we use 0.01 learning rate and 5×10^{-4} L_2 regularization for LR. For SVM, we use the RBF kernel with $\gamma = 1/n$ and 1.0 L_2 regularization. For DT, we use Gini impurity as quality measure. We use the same parameters for MLP as described in Appendix B.

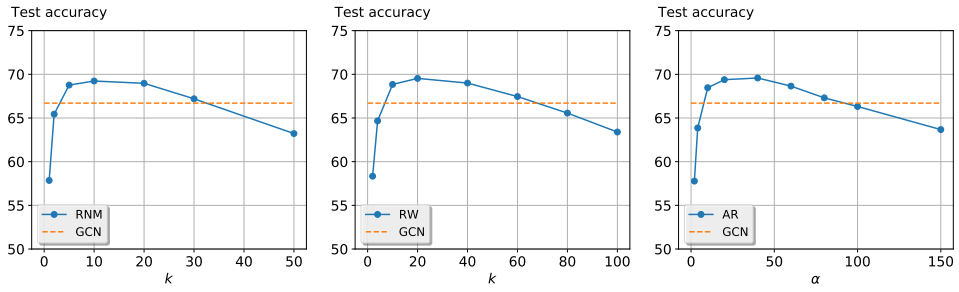
APPENDIX D STABILITY ANALYSIS OF GLP

We test how the filter parameters k and α influence the performance of GLP. Figs. 4 to 6 plot the classification accuracies of GLP with different k and α on three citation networks, with 4 labels per class. Consistent with our analysis in section 5, the classification accuracy of GLP first increases and then decreases as k and α increases. The results shows that GLP consistently outperforms GCN for a wide range of k and α.

APPENDIX E RUNNING TIMES OF GLP

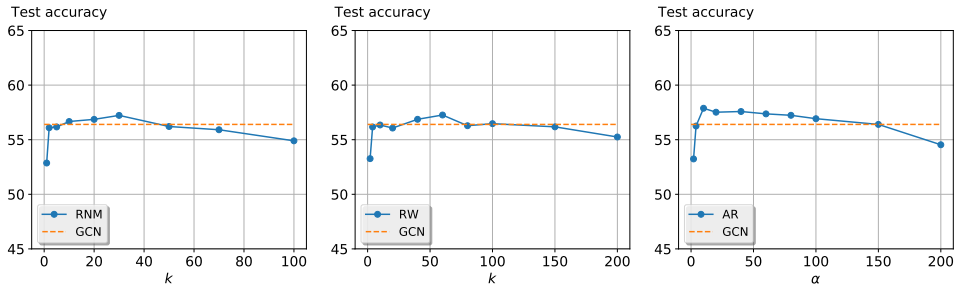
The running times of GLP and some other baselines are also reported in Tables 1 and 2. On citation networks and MNIST, GLP runs much faster than GCN, especially for large-scale datasets such as PubMed and MNIST. The reason is that feature filtering only needs to be done once in GLP, while GCN basically performs filtering in every training step.

[2]https://github.com/tkipf/ica



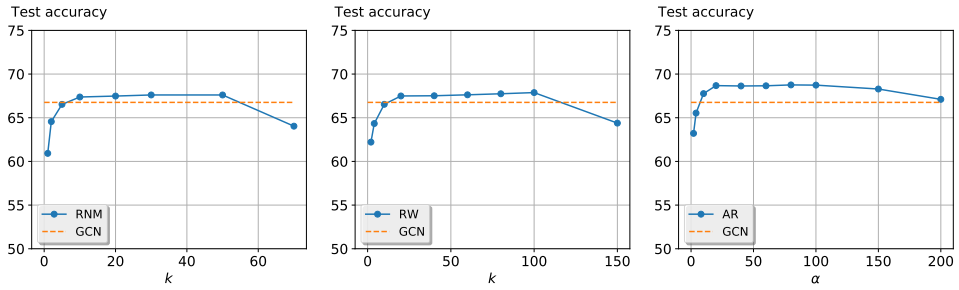
(a) GLP (RNM, MLP) (b) GLP (RW, MLP) (c) GLP (AR, MLP)

Figure 4: Classification accuracy on Cora with different parameters.



(a) GLP (RNM, MLP) (b) GLP (RW, MLP) (c) GLP (AR, MLP)

Figure 5: Classification accuracy on CiteSeer with different parameters.



(a) GLP (RNM, MLP) (b) GLP (RW, MLP) (c) GLP (AR, MLP)

Figure 6: Classification accuracy on PubMed with different parameters.

When the data feature dimension m is very large, such as on NELL ($m = 61278$), GLP becomes slower. However, the computation of filtering (convolution) of GCN in Eq. (8) can be speeded up by matrix chain multiplication, and the larger m is, the more times it can be accelerated. Therefore, on NELL, GLP with RNM and RW filters have similar running times as GCN.