Efficient Vector-Based Label Propagation for Massive Low-Rank Graphs

Tengkai Yu
University of Victoria
Victoria, BC, Canada
yutengkai@uvic.ca

Venkatesh Srinivasan
Santa Clara University
Santa Clara, CA, USA
vsrinivasan4@scu.edu

Alex Thomo
University of Victoria
Victoria, BC, Canada
thomo@uvic.ca

Abstract

Label Propagation (LP) is a classic method in semi-supervised learning, where labels diffuse across graph edges until convergence. Its main obstacle on large dense graphs is scalability: classical LP requires an $n \times n$ adjacency, with quadratic memory and runtime costs. We focus on the common class of *low-rank graphs*, where the adjacency has the form $VV^\top$ for an embedding matrix $V \in \mathbb{R}^{n \times d}$. Such graphs arise naturally in applications including similarity graphs from embeddings, recommender systems, kernel methods, and dense affinity graphs in vision and biology. We introduce **VLP** (*Vector-Based Label Propagation*), which operates entirely in the embedding space without explicit edges. VLP reduces memory from $O(n^2)$ to $O(nd)$, runs efficiently on GPUs, and is mathematically equivalent to classical LP. We further give the first convergence proof of LP in this low-rank setting. VLP makes label propagation feasible for graphs with millions of nodes, far beyond the reach of traditional methods.

CCS Concepts

• **Information systems** → **Data mining**.

Keywords

Label Propagation, Low-Rank Graphs, Semi-supervised Learning

ACM Reference Format:

Tengkai Yu, Venkatesh Srinivasan, and Alex Thomo. 2026. Efficient Vector-Based Label Propagation for Massive Low-Rank Graphs. In *Proceedings of the Nineteenth ACM International Conference on Web Search and Data Mining (WSDM '26), February 22–26, 2026, Boise, ID, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3773966.3779368

1 Introduction

Label Propagation (LP) is a foundational method in semi-supervised learning on graphs [1, 2, 14, 24, 25]. By iteratively diffusing labels from a small set of labeled nodes across graph edges, LP achieves strong performance in tasks ranging from text categorization and image annotation to community detection [10, 16] and graph neural network post-processing [11, 13, 14].

Despite its versatility, classical LP faces a core scalability bottleneck: it requires the full adjacency, which for dense graphs means storing $\Theta(n^2)$ edges, a prohibitive quadratic memory cost [9, 18]. To cope, many methods apply kNN pruning [8] to sparsify the graph. While this reduces cost, it also removes many edges, weakening global connectivity and often distorting the underlying structure.

In this paper, we take a different perspective. Many modern dense graphs are not arbitrary but exhibit a *low-rank structure*: their adjacency can be written as $VV^\top$, where $V \in \mathbb{R}^{n \times d}$ is a node-embedding matrix with $d \ll n$. This form arises in numerous settings, including similarity graphs from text or image embeddings, latent-factor recommender systems, kernelized ML graphs, and dense affinity graphs in biology and vision [5, 9, 18, 23]. Low-rank graphs encode full all-to-all similarity, yet explicitly constructing the $n \times n$ adjacency costs $O(n^2)$ time and memory, making standard LP implementations impractical. This raises a natural question: *can we exploit the low-rank structure of such graphs to make label propagation scalable without sacrificing accuracy or global connectivity?*

We introduce **VLP** (*Vector-Based Label Propagation*), a new formulation of LP designed for low-rank graphs. Instead of enumerating edges, VLP performs updates directly in the embedding space using matrix-vector operations. This reduces the memory footprint from $O(n^2)$ to $O(nd)$ and enables fully vectorized computation on GPUs. VLP is *mathematically equivalent* to classical LP: it produces the same propagation outcomes without approximation or loss of connectivity. Moreover, for the low-rank setting, we establish rigorous convergence guarantees that are unavailable for general graphs.

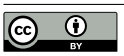
Two prior methods use low-rank label propagation: **BigLP** [20] scales via gradient-based approximations, and **LowrankTLP** [17] addresses multi-relational graphs using tensor-product and low-rank constructions. In contrast, **VLP** works directly with dense graphs of the form $VV^\top$ and performs *exact* propagation without constructing the quadratic adjacency or relying on approximations.

Contributions. The main contributions of this work are:

- We identify the low-rank setting as a practically important class of graphs and introduce **VLP**, a vector-based reformulation of LP that exploits this structure.
- VLP avoids the $\Theta(n^2)$ adjacency, reducing memory from $O(n^2)$ to $O(nd)$ and per-iteration cost from $O(n^2c)$ to $O(ndc)$. Each update is two matrix multiplications, ideal for GPUs.
- We prove convergence of VLP under natural assumptions on node embeddings, providing the first theoretical guarantees for LP in this regime.
- Experiments on large-scale graphs show that VLP scales to millions of nodes while exactly matching classical LP, reaching regimes existing methods cannot.

2 Vector-Based Label Propagation (VLP)

In this section, we outline the main steps of classical LP and then show how to adapt it to *low-rank* graphs.



WSDM '26, Boise, ID, USA

ACM ISBN 979-8-4007-2292-9/2026/02
https://doi.org/10.1145/3773966.3779368

2.1 Classical LP Formulation

Label propagation spreads label information from labeled to unlabeled nodes by iteratively averaging over neighbors. Suppose there are c classes. At iteration t, each node v maintains a row vector $\mathbf{Y}_v^{(t)} \in \mathbb{R}^{1\times c}$ that stores its current label distribution. Labeled nodes are initialized with one-hot vectors, while unlabeled nodes may be initialized with a uniform distribution.

The graph is represented by its weighted adjacency matrix $A \in \mathbb{R}^{n\times n}$, where A_{vu} is the weight of the edge from v to u and diagonal entries are zero. At each step, a node updates its label distribution by taking a degree-normalized weighted average of its neighbors:

$$\mathbf{Y}_v^{(t+1)} = \frac{1}{\sum_u A_{vu}} \sum_u A_{vu}\, \mathbf{Y}_u^{(t)}.$$

This way, label probabilities diffuse through the network, and after multiple iterations, the distributions tend to stabilize, yielding confident predictions for previously unlabeled nodes (see Alg. 1).

Algorithm 1 Classical Label Propagation (Adjacency-based)

Input: Adjacency matrix $A \in \mathbb{R}^{n\times n}$, initial label matrix $Y^{(0)} \in \mathbb{R}^{n\times c}$, number of iterations T
Output: Final label distributions $Y^{(T)}$

Compute degrees: $k_i = \sum_{j=1}^{n} A_{ij}$ for all i
Set inverse degrees: $\text{inv\_deg}_i = 1/k_i$
for $t = 0$ to $T-1$ **do**
  for each node $i = 1, \ldots, n$ **do**
    Update: $Y_i^{(t+1)} = \text{inv\_deg}_i \cdot \sum_{j=1}^{n} A_{ij}\, Y_j^{(t)}$
return $Y^{(T)}$

2.2 From Explicit Adjacency to Vector Factorisation

A key observation is that many practical graphs admit a low-rank form, $\mathbf{A} = VV^\top$ with $V \in \mathbb{R}^{n\times d}$, where each row $v_i \in \mathbb{R}^d$ is a feature vector (or embedding) for node i. This setting naturally covers graphs where edges represent generalized similarity measures between nodes, since many such measures can be expressed as inner products in a feature space. Common examples are similarity graphs built from normalized cosine similarity of embeddings and polynomial kernels with explicit feature maps. To be suitable for label propagation, these inner products must be nonnegative (e.g., polynomial kernels of even degree).

Cosine similarity as a special case. In our experiments, we adopt cosine-based similarity, a common choice in representation learning. Given unit-norm embeddings v_i, the weight between nodes i and j is

$$A_{ij} = \tfrac{1}{2}\big(1 + \tfrac{v_i^\top v_j}{\|v_i\|\|v_j\|}\big),$$

which linearly rescales cosine similarity from $[-1, 1]$ to $[0, 1]$, ensuring nonnegativity while retaining the full similarity signal. This can be written compactly by augmenting each vector as $v_i' = \frac{[\,v_i;1\,]}{\sqrt{2}}$, so that $A_{ij} = v_i'^\top v_j'$. Collecting rows gives $V = [v_1'; \ldots; v_n'] \in \mathbb{R}^{n\times(d+1)}$ and hence $\mathbf{A} = VV^\top$ (with diagonal entries set to zero). For ease of notation, in the remainder we drop the prime and simply write v_i for the augmented vectors.

Why factorisation matters. Working directly with V avoids the explicit construction of the $n \times n$ adjacency, reducing space from $O(n^2)$ to $O(nd)$, which is crucial since in practice $d \ll n$. This observation underpins our vector-based reformulation of LP.

2.3 Vector-Based Update Rule

With $\mathbf{A}$ in low-rank form, label propagation can be carried out *without ever constructing* the $n \times n$ adjacency:

$$\mathbf{Y}^{(t+1)} = \mathbf{inv\_deg} \odot \Big(V\big(V^\top \mathbf{Y}^{(t)}\big) - \mathbf{self\_loop} \odot \mathbf{Y}^{(t)}\Big),$$

where $\mathbf{self\_loop}_i = v_i^\top v_i$ and $\odot$ is the Hadamard product. Intuitively, each iteration first *gathers* label information into a compact embedding summary via $V^\top Y^{(t)}$, then *spreads* it back across all nodes through V, removes self-contributions, and normalizes by degree.

Efficient self-loop and degree computation. Both self-loops and degrees can be obtained once in vectorized form. The self-loop weights are the diagonal entries of $VV^\top$, which can be written as

$$s = (V \odot V)\, \mathbf{1}_d,$$

where each $s_i = v_i^\top v_i$. The degree of node i is the sum of all similarities $v_i^\top v_j$ to other nodes, which equals $v_i^\top V_{\text{sum}}$ with $V_{\text{sum}} = \sum_j v_j$, minus its own self-loop s_i. In vector form, this gives

$$k = V(V^\top \mathbf{1}_n) - s.$$

Thus k collects the row sums of $VV^\top$ with the diagonal removed. Finally, the normalization factors are $\text{inv\_deg} = k^{-1}$, applied elementwise.

Algorithm 2 Vector-Based Label Propagation (VLP) — Vectorized

Input: Node embedding matrix $V \in \mathbb{R}^{n\times d}$ (after augmentation), initial label matrix $Y^{(0)} \in \mathbb{R}^{n\times c}$, number of iterations T
Output: Final label distributions $Y^{(T)}$

$V_{\text{sum}} \leftarrow V^\top \mathbf{1}_n$ ($\in \mathbb{R}^d$)
$s \leftarrow (V \odot V)\, \mathbf{1}_d$ (self-loops, $\in \mathbb{R}^n$)
$k \leftarrow V V_{\text{sum}} - s$ (degrees, $\in \mathbb{R}^n$)
$\text{inv\_deg} \leftarrow k^{-1}$ (elementwise reciprocal)
for $t = 0$ to $T-1$ **do**
  $Z \leftarrow V^\top Y^{(t)}$ ($d \times c$ GEMM)
  $W \leftarrow VZ$ ($n \times c$ GEMM)
  $W \leftarrow W - (s\mathbf{1}_c^\top) \odot Y^{(t)}$ (rowwise broadcast)
  $Y^{(t+1)} \leftarrow (\text{inv\_deg}\, \mathbf{1}_c^\top) \odot W$ (rowwise normalize)
return $Y^{(T)}$

Alg. 2 summarizes the procedure. Each iteration involves two dense matrix-matrix multiplications ($V^\top Y^{(t)}$ and VZ), followed by vectorized subtraction of self-contributions and normalization. In this way, labels flow efficiently through the graph without constructing the full adjacency. The per-iteration complexity is $O(ndc)$ rather than $O(n^2 c)$ as in classical LP, a crucial saving when $n \gg d$. Because the heavy operations reduce to GEMMs, VLP is well suited to modern GPU architectures.

3 Convergence Analysis

Label propagation methods have long faced convergence issues. Majority-vote schemes can oscillate or fail to stabilize in dense graphs [10, 16]. Heuristic refinements such as LabelRank [21] and semi-synchronous updates [4] attempt to enforce convergence through tie-breaking or scheduling, while theoretical studies [15] provide guarantees only in restricted settings. In summary, most existing formulations lack general convergence results and are limited to the case of discrete labels. Here we show that classical LP admits a formal convergence guarantee in the low-rank setting, even for continuous labels.

Recall that, in our vector-based label propagation, the update is carried out via

$$\mathbf{Y}^{(t+1)} = \mathbf{inv\_deg} \odot \Big(V\big(V^\top \mathbf{Y}^{(t)}\big) - \big(\mathbf{self\_loop} \odot \mathbf{Y}^{(t)}\big)\Big),$$

where **self_loop** is the vector with the ith entry given by $v_i^\top v_i$. This yields an effective propagation matrix

$$P = \mathbf{inv\_deg} \odot \Big(VV^\top - \mathrm{diag}(VV^\top)\Big).$$

Normalizing by degree ensures each row sums to one, making P row-stochastic.

Lemma 3.1 (Row-Stochasticity and Nonnegativity). *The matrix P is nonnegative and row-stochastic; that is, for every row i, $\sum_j P_{ij} = 1$, and $P_{ij} \geq 0$ for all i, j.*

Given a row-stochastic $n \times n$ matrix P, construct a directed graph $G(P)$ with n nodes and an edge from node i to j whenever $P_{ij} > 0$.

Definition 3.2. A matrix P is said to be irreducible if $G(P)$ is strongly connected.

In our proof, we assume that the embedding vectors are drawn from a continuous distribution (with a density) on the sphere. This is natural in practice: modern embedding methods (e.g., neural encoders, PCA, random projections) produce continuous-valued vectors, and even in discrete settings, the probability of exact antipodality is negligible in high dimensions. Since the event of two vectors being exactly opposite has measure zero under a continuous distribution, with probability one, all off-diagonal similarities are positive, making the digraph complete and strongly connected.

Lemma 3.3 (Irreducibility). *The matrix P is irreducible.*

Definition 3.4. For each node i in $G(P)$, consider all possible cycle lengths (closed walks from i back to itself). Define the period of i, $p(i)$, as the greatest common divisor(gcd) of those cycle lengths. If $p(i) = 1$ for all i, then P is called aperiodic.

Since the graph is complete, both 2 and 3-cycles exist, so the gcd of cycle lengths is 1, implying aperiodicity.

Lemma 3.5 (Aperiodicity). *The matrix P is aperiodic.*

As P is finite, irreducible, and aperiodic, by the Fundamental Theorem of Markov chains, P has a stationary distribution π. Furthermore, $P^t \to \mathbf{1}\pi^T$ where $\mathbf{1}$ is the n-vector of 1's. Hence, for any initial label column y, $P^t y \to \mathbf{1}(\pi^T y)$: all nodes reach consensus. Thus, the low rank setting admits a formal convergence guarantee for LP unlike the general setting where convergence can fail. In practice, we halt propagation before complete convergence and use the intermediate state for downstream tasks.

4 Experiments

We evaluate our vector-based label propagation algorithm on four real-world datasets: Amazon Products, Yelp, Taobao, and Flickr, each accessed through PyTorch Geometric [7]. A brief overview of their node counts and feature dimensionalities is given in Table 1. For Amazon Products, Flickr, and Yelp, node feature vectors come from the GraphSAINT repository [22], while Taobao is processed according to user–category usage data [12].

All experiments were conducted using Google Colab Pro with A100 GPU instances. The system specifications include 83.5 GB of RAM, 40 GB of GPU memory, and a total disk space of 235.7 GB. Our Python implementation is available at https://anonymous.4open.science/r/vector-based-label-propagation-D458/README.md.

Table 1: Summary of the Four Datasets

Dataset	Number of Nodes	Feature Dim.
Amazon Products	1,569,960	200
Yelp	716,777	300
Taobao	936,946	66
Flickr	89,249	500

Choice of metrics. VLP is mathematically equivalent to adjacency-based LP. Our experiments therefore focus on *scalability*: demonstrating that VLP scales to millions of nodes, whereas existing implementations cannot.

Choice of baselines. In our runtime evaluation, we compare VLP against three widely used implementations of the classical label propagation algorithm of Zhu and Ghahramani [24, 25]: *scikit-learn* [19], *scikit-network* [3], and *PyTorch Geometric* [6]. Each provides a linear propagation scheme in which labels are iteratively diffused across edges based on weighted adjacency. We focus on these popular implementations, as other known approaches, such as kNN-pruned LP [9] or stabilized variants like LabelRank [21], achieve scalability by modifying the propagation process (e.g., through neighborhood-voting heuristics, pruning edges, or altering the update rule), which differs fundamentally from the linear propagation setting considered here.

Experimental Set-up. Throughout our experiments, we assign each node a label from $[0, 50)$ at random, yielding a total of 50 label categories. We set the number of label propagation iterations (the inner loop) to 1000 for each library, and for every run, we repeat the entire procedure five times and average the runtime. Notably, both *scikit-learn* and *scikit-network* rely on an explicit adjacency matrix, which we precompute from node features via cosine similarity; this cost is included in their total runtime. Likewise, *PyTorch Geometric* requires constructing the edge list (on GPU or CPU) before running its label propagation operator. We include these precomputation times to ensure a fair comparison.

We measure performance across datasets by sampling subgraphs at various thresholds. For each threshold, we keep a fraction of the node feature matrix to control the resulting subgraph size. We then run label propagation with all four implementations (our VLP plus the three libraries) and record their average total runtime. This

procedure is repeated for subgraphs of increasing size to illustrate each method's scaling behaviour. When the library fails (due to memory or time constraints), we label its entry with 'X'. Finally, we consider 50% and 100% of the nodes to show full-scale performance; at these higher fractions, the adjacency-based approaches typically cannot complete.

Experimental Results. As shown in Table 2, we start by sampling 0.2%, 0.4%, 0.6%, 0.8%, and 1.0% of the Amazon dataset, corresponding to 3,139 up to 15,699 nodes. Across these subgraphs, our vector-based label propagation (VLP) consistently yields the fastest runtimes (all entries in bold), while the other libraries—scikit-learn, scikit-network, and PyTorch Geometric—incur higher costs and eventually run out of memory as node counts exceed 15,000. At 50% of the dataset (784,980 nodes) and the full dataset (1,569,960 nodes), only VLP succeeds, with respective runtimes of **6.85 s** and **13.37 s**. All other methods fail (marked as 'X') at these larger sizes, underscoring VLP's scalability on dense graphs derived from node vectors. At the highest threshold that some of the other methods can reach (1.0%), our VLP method outperforms the best of them by two orders of magnitude. This pattern holds across all datasets.

Table 2: Runtime (in seconds) for Amazon over 1000 iterations. Bold denotes our VLP, always the best. 'X' indicates library failure.

Thr. (%)	Nodes	VLP	scikit-learn	scikit-net	PyG
0.2	3,139	**0.18**	0.75	7.10	4.81
0.4	6,279	**0.19**	2.58	22.36	21.10
0.6	9,419	**0.22**	6.03	50.63	112.61
0.8	12,559	**0.23**	11.32	92.57	X
1.0	15,699	**0.25**	18.38	134.11	X
50.0	784,980	**6.85**	X	X	X
100.0	1,569,960	**13.37**	X	X	X

Table 3 presents the Yelp dataset results. Up to around 8–11k nodes, scikit-learn and scikit-network both succeed but cost more time than VLP for each row, while PyTorch Geometric fails once node counts surpass roughly 11,000. At 50% (358,423 nodes) and 100% (716,847 nodes), only our VLP can successfully perform label propagation, taking **3.96 s** and **7.72 s**, respectively, whereas all others run out of memory.

Table 3: Runtime (in seconds) for Yelp dataset over 1000 iterations. VLP always the fastest in bold. 'X' indicates library failure.

Thr. (%)	Nodes	VLP	scikit-learn	scikit-net	PyG
0.4	2,867	**0.59**	0.80	6.45	4.63
0.8	5,734	**0.17**	2.88	18.70	17.99
1.2	8,602	**0.19**	5.44	38.60	70.31
1.6	11,469	**0.20**	9.49	67.38	X
2.0	14,336	**0.25**	14.44	114.25	X
50.0	358,423	**3.96**	X	X	X
100.0	716,847	**7.72**	X	X	X

Finally, Table 4 shows Taobao, which similarly features near-complete edge density once users are mapped to categories via TF-IDF. Even at small thresholds (e.g. 9,369 nodes), scikit-learn and scikit-network remain slower, while PyTorch Geometric soon fails above 14k. VLP alone scales to 50% (468,473 nodes) and 100% (936,946 nodes) of the dataset, requiring only **3.06 s** and **5.94 s** respectively to propagate 50 random labels for all nodes. Similar results obtained for Flickr are omitted due to space constraints.

Table 4: Runtime (in seconds) for Taobao dataset over 1000 iterations. 'X' denotes failure. VLP times in bold.

Thr. (%)	Nodes	VLP	scikit-learn	scikit-net	PyG
0.5	4,684	**0.17**	1.69	12.99	12.53
1.0	9,369	**0.19**	5.79	46.60	108.84
1.5	14,054	**0.21**	12.62	99.20	X
2.0	18,738	**0.23**	22.51	174.90	X
2.5	23,423	**0.22**	X	X	X
3.0	28,108	**0.24**	X	X	X
50.0	468,473	**3.06**	X	X	X
100.0	936,946	**5.94**	X	X	X

In every dataset, VLP outperforms adjacency-based label propagation, irrespective of graph size. Once node counts surpass around 11,000 for PyTorch Geometric or 15–20,000 for scikit-learn and scikit-network, these methods exceed their memory budgets or exhibit runtime failures. At half or all of each dataset, millions of nodes in Amazon and hundreds of thousands in Yelp or Taobao, only VLP completes label propagation successfully. This consistent advantage highlights that our approach is the fastest and most scalable option for large, low-rank graphs where adjacency construction becomes infeasible.

Clamping in baseline implementations. The baselines differ in how they handle initially labeled nodes: scikit-learn applies hard clamping (labels fixed), PyTorch Geometric uses soft clamping controlled by a parameter α, and scikit-network does not explicitly document its policy. Our VLP, like the unclamped adjacency-based LP, lets all nodes update freely. These differences do not affect runtime, since the cost is dominated by matrix multiplication and clamping is only an $O(nc)$ step.

5 Conclusion

Traditional label propagation struggles on super-dense graphs because storing the full adjacency information is infeasible and k-nearest neighbor pruning, often used to save memory, discards valuable global information. Constructing explicit adjacency structures also adds significant latency. Our vector-based label propagation (VLP) bypasses these issues by operating directly on node embeddings, computing normalized cosine similarities, and normalizing with inverse degrees, making it highly space-efficient. Furthermore, the propagation matrix is such that it converges under standard assumptions. Experiments on Flickr, Amazon, Yelp, and Taobao show that while adjacency-based methods fail or slow dramatically beyond 11K–20K nodes, VLP efficiently scales to hundreds of thousands or even millions of nodes, retaining full similarity information without pruning. We believe the low-rank approach offers a unifying, high-impact path for scaling many classical graph-mining algorithms.

References

[1] Blum, A., Chawla, S.: Learning from labeled and unlabeled data using graph mincuts (2001)

[2] Blum, A., Lafferty, J., Rwebangira, M.R., Reddy, R.: Semi-supervised learning using randomized mincuts. In: Proceedings of the twenty-first international conference on Machine learning. p. 13 (2004)

[3] Bonald, T., de Lara, N., Lutz, Q., Charpentier, B.: Scikit-network: Graph analysis in python. Journal of Machine Learning Research **21**(185), 1–6 (2020), http://jmlr.org/papers/v21/20-412.html

[4] Cordasco, G., Gargano, L.: Community detection via semi-synchronous label propagation algorithms. In: 2010 IEEE international workshop on: business applications of social network analysis (BASNA). pp. 1–8. IEEE (2010)

[5] Daniluk, M., Dabrowski, J., Rychalska, B., Gołuchowski, K.: Synerise at kdd cup 2021: Node classification in massive heterogeneous graphs. KDD Cup OGB Challenge 2021 (2021)

[6] Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)

[7] Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428 (2019)

[8] Fix, E.: Discriminatory analysis: nonparametric discrimination, consistency properties, vol. 1. USAF school of Aviation Medicine (1985)

[9] Fujiwara, Y., Irie, G.: Efficient label propagation. In: International conference on machine learning. pp. 784–792. PMLR (2014)

[10] Garza, S.E., Schaeffer, S.E.: Community detection with the label propagation algorithm: a survey. Physica A: Statistical Mechanics and its Applications **534**, 122058 (2019)

[11] Gasteiger, J., Bojchevski, A., Günnemann, S.: Predict then propagate: Graph neural networks meet personalized pagerank. arXiv preprint arXiv:1810.05997 (2018)

[12] Group, A.: Taobao user behavior dataset. https://tianchi.aliyun.com/dataset/649, accessed: 2024-12-31

[13] Huang, Q., He, H., Singh, A., Lim, S.N., Benson, A.R.: Combining label propagation and simple models out-performs graph neural networks. arXiv preprint arXiv:2010.13993 (2020)

[14] Joachims, T.: Transductive learning via spectral graph partitioning. In: Proceedings of the 20th international conference on machine learning (ICML-03). pp. 290–297 (2003)

[15] Kothapalli, K., Pemmaraju, S.V., Sardeshmukh, V.: On the analysis of a label propagation algorithm for community detection. In: International Conference on Distributed Computing and Networking. pp. 255–269. Springer (2013)

[16] Leung, I.X., Hui, P., Lio, P., Crowcroft, J.: Towards real-time community detection in large networks. Physical Review E—Statistical, Nonlinear, and Soft Matter Physics **79**(6), 066107 (2009)

[17] Li, Z., Petegrosso, R., Smith, S., Sterling, D., Karypis, G., Kuang, R.: Scalable label propagation for multi-relational learning on the tensor product of graphs. IEEE Transactions on Knowledge and Data Engineering **34**(12), 5964–5978 (2021)

[18] Liu, W., He, J., Chang, S.F.: Large graph construction for scalable semi-supervised learning. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 679–686. Citeseer (2010)

[19] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[20] Petegrosso, R., Zhang, W., Li, Z., Saad, Y., Kuang, R.: Low-rank label propagation for semi-supervised learning with 100 millions samples. arXiv preprint arXiv:1702.08884 (2017)

[21] Xie, J., Szymanski, B.K.: Labelrank: A stabilized label propagation algorithm for community detection in networks. In: 2013 IEEE 2nd Network Science Workshop (NSW). pp. 138–143. IEEE (2013)

[22] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Graphsaint: Graph sampling based inductive learning method. arXiv preprint arXiv:1907.04931 (2019)

[23] Zhang, W., Yang, M., Sheng, Z., Li, Y., Ouyang, W., Tao, Y., Yang, Z., Cui, B.: Node dependent local smoothing for scalable graph learning. Advances in Neural Information Processing Systems **34**, 20321–20332 (2021)

[24] Zhu, X., Ghahramani, Z.: Learning from labeled and unlabeled data with label propagation.(2002) (2002)

[25] Zhu, X., Ghahramani, Z., Lafferty, J.D.: Semi-supervised learning using gaussian fields and harmonic functions. In: Proceedings of the 20th International conference on Machine learning (ICML-03). pp. 912–919 (2003)