Class-Imbalanced Learning on Graphs: A Survey[Ma+23]

Rishav Das

National Institute of Science Education and Research

June 7, 2024

Introduction

Graphs

A graph, G can be defined as a tuple of two sets $\mathcal{V}$ and $\mathcal{E}$, i.e., G = ($\mathcal{V}$,$\mathcal{E}$), where $\mathcal{V}$ is the set of vertices or nodes and $\mathcal{E}$ is the set of edges which are again tuples of vertices, (u,v) ϵ $\mathcal{E}$, where u, v ϵ $\mathcal{V}$.

Introduction

Graphs

A graph, G can be defined as a tuple of two sets $\mathcal{V}$ and $\mathcal{E}$, i.e., G = $(\mathcal{V},\mathcal{E})$, where $\mathcal{V}$ is the set of vertices or nodes and $\mathcal{E}$ is the set of edges which are again tuples of vertices, (u,v) ϵ $\mathcal{E}$, where u, v ϵ $\mathcal{V}$.

We can also define a node feature set $\mathbf{X} \; \epsilon \; \boldsymbol{R}^{n\times d}$, where n is the number of vertices, $n = |\mathcal{V}|$ and d is the number of features of a vertex. And we can define a label set $\mathbf{y} \; \epsilon \; \boldsymbol{R}^{n\times 1}$ which contains the label of the vertices.

Introduction

Graph Representation Learning (GRL)

Meaningful representation of graphs for various graph mining applications.

Categories of GRL approaches:

1. Network embedding models:
   Network Embedding is a technique used in machine learning to represent nodes or links in a network as points in a continuous vector space. The goal is to learn latent low-dimensional feature representations for these nodes or links.

Introduction

2. Graph Neural Networks (GNNs)
GNNs typically adopt a message-passing mechanism through neighborhood aggregation, updating a node's representation by aggregating information from its neighboring nodes and edges. After k iterations of aggregation, a node's representation encapsulates the structural information within its k-hop neighborhood, which is defined as:

$$h_v^{(k)} = Update(h_v^{(k-1)}, Aggregate(\{h_u^{k-1} | \forall u \epsilon \mathcal{N}(v)\}))$$

where h_v^k is the representation vector od node $v \epsilon \mathcal{V}$ in the k-th GNN layer, $\mathcal{N}(v)$ is the set of neigbors of node v, Aggregate(.) is the neighbor aggregation function and Update(.) is the combination function. Here h_n^0 is initialized with node attribute $\mathbf{X}_v$.

Introduction

3. Knowledge Graph Embedding Methods
   A knowledge graph $\mathcal{G} = \{E, R, F\}$ is a collection of entities E, relations R, and facts F. A fact is a triple $(h, r, t) \in F$ that denotes a link $r \in R$ between the head $h \in E$ and the tail $t \in E$ of the triple. Here the task is to learn node and edge embeddings by computing the acceptability score of fact triplets.

Introduction

Class Imbalance Learning

We define Class distribution, P_k, and imbalance ratio, ρ as follows,

$$P_k = \frac{|C_k|}{\sum_{i=1}^{K} |C_i|}, \quad \rho = \frac{max_k |C_k|}{min_k |C_k|},$$

where C_k is the set of labeled samples in the class $k \in \{1, 2, ..., K\}$.

Introduction

Class Imbalance Learning approaches

1. Data-level methods
    1. over-sampling minority,
    2. under-sampling majority,
    3. hybrid-sampling
2. Algorithm-level methods
    1. cost-sensitive learning,
    2. ensemble learning,
    3. loss function engineering, ...

Class Imbalance Learning on Graphs (CILG)

Chalanges

1. The data is non-Euclidean.
2. Imbalance could be at different levels.

Approaches

1. Data-level methods
   1. Data interpolation
   2. Adversarial generation
   3. Pseudo-labeling
2. Algorithm-level methods
   1. Model refinement
   2. loss function engineering
   3. post-hoc adjustments

CILG models, arranged in chronological order.

Data-Level						
Model	**Year**	**Venue**	**Data Interpolation**	**Adversarial Generation**	**Pseudo-Labeling**	**Key Components**
SPARC [80]	2018	KDD			✓	Label propagation; Self-paced learning
GraphSMOTE [78]	2021	WSDM	✓			SMOTE; Pre-training
GraphENS [35]	2021	ICLR	✓			mixup; Neighbor sampling; Saliency filtering
ImGAGN [38]	2021	KDD		✓		Graph structure reconstruction
D-GCN [58]	2021	CSAE				k-NN; Graph structure generator
SET-GNN [21]	2021	ICONIP			✓	Label propagation; Self-training
GraphMixup [64]	2022	ECML/PKDD	✓			mixup; Auxiliary objectives
GATSMOTE [26]	2022	Mathematics	✓			SMOTE; Attention
SORAG [10]	2022	ECML/PKDD		✓		Node generator; Edge generator
DPGNN [59]	2022	MLG			✓	Label propagation; Metric learning
GNN-CL [24]	2022	arXiv	✓			SMOTE; Curriculum learning; Attention
GraphSR [81]	2023	AAAI			✓	Label propagation; Reinforcement learning

Algorithm-Level						
Model	**Year**	**Venue**	**Model Refinement**	**Loss Function Engineering**	**Post-hoc Adjustments**	**Key Components**
RSDNE [62]	2018	AAAI	✓			Random walk; Intra/Inter-class similarity
ImVerde [63]	2018	Big Data	✓			Random walk; Balanced sampling
DR-GCN [46]	2020	IJCAI			✓	Adversarial training; Distribution alignment
RECT [61]	2020	TKDE	✓			Class-label relaxation
ReNode [6]	2021	NeurIPS		✓		Topology imbalance; Node reweighting
FRAUDRE [73]	2021	ICDM		✓		Imbalanced distribution-oriented loss
TAM [48]	2022	ICLR		✓		Topology-aware margin loss
CM-GCL [37]	2022	NeurIPS	✓	✓		Contrastive learning; Network pruning
LTE4G [69]	2022	CIKM	✓		✓	Degree imbalance; Knowledge distillation
ACS-GNN [28]	2022	ICNSC	✓	✓		Cost-sensitive learning; Attention
EGCN [55]	2022	ICNSC	✓			Class-weighted aggregation
KINC-GCN [1]	2022	ICNSC	✓			Kernel propagation; Node clustering
FACS-GCN [43]	2022	IJCNN		✓		Cost-sensitive learning; Adversarial training
GraphDec [72]	2022	GLFrontiers	✓	✓		Contrastive learning; Sparsity training
ImGCL [71]	2023	AAAI	✓	✓		Contrastive learning; Balanced sampling

GraphSMOTE

- Step1: learning embeding,
  $h_v^1 = \sigma(\mathbf{W}^1.CONCAT(F[v,:], F.A[:,v]))$
- Step2: Synthetic node generation,
  finding the nearest neighbor from same class,
  $nn(v) = argmin_u ||h_u^1 - h_v^1|| \quad s.t. \quad Y_u = Y_v$
  then,
  $h_{v'}^1 = (1-\delta).h_v^1 + \delta h_{nn(v)}^1$, where $\delta \in [0,1]$ is a random variable and v' is the generated node.
- Step3: Edge generator,
  predicted relation information between nodes u and v,
  $E_{u,v} = softmax(\sigma(h_v^1.S.h_u^1))$
  Loss function for learning the parameter S,
  $L_edge = ||E - A||_F^2$
  new adjacency matrix, $\tilde{A} = 1 \; if \;\; E > \eta \; or \; 0 \; otherwise$

Data Interpolation

GraphSMOTE

- Step4: GNN classifier,
  If GraphSage GNN is used,
  $h_v^2 = \sigma(W^2.CONCAT(h_v^1, \tilde{H}^1.\tilde{A}[:, v])),$
  $P_v = softmax(\sigma(W^c.CONCAT(h_v^2, H^2.\tilde{A}[:, v])))$
  with the loss function as cross entropy loss,
  $L_{node} = \sum_{u \in \tilde{\mathcal{V}}_L} \sum_c (\mathbf{1}(Y_u == c). \log P_v(c))$

Data Interpolation

GraphENS

1. Graph ego network synthesis.
2. uses mixup,
   $\tilde{x} = \lambda x_i + (1 - \lambda)x_j$, $\tilde{y} = \lambda y_i + (1 - \lambda)y_j$,
   i,j are random instances, where one belongs to minority class and $\lambda \in [0, 1]$
3. filters out class-specific attributes of the target node using a gradient-based feature saliency that measures the importance of each feature in predicting the target node's label

Data Interpolation

GraphMixup

1. performs mixup in the semantic space rather than the input or embedding space, preventing the generation of out-of-domain minority sample
2. incorporates two auxiliary self-supervised learning objectives: local-path prediction and global-path prediction

Data Interpolation

GraphMixup

1. performs mixup in the semantic space rather than the input or embedding space, preventing the generation of out-of-domain minority sample
2. incorporates two auxiliary self-supervised learning objectives: local-path prediction and global-path prediction

GATSMOTE and GNN-CL

1. They employ attention mechanisms in their edge generators to enhance predicted edge quality between synthetic and real nodes (similar to GraphSMOTE)

Adversarial Generation

Imbalanced Network Embedding via Generative (ImGAGN)

1. first to apply Generative Adversarial Networl (GAN) to graphs
2. introduces a generator network for synthesizing minority nodes and their links to real minority nodes
3. derives features from averaging neighboring minority node features
4. GCN is used as the discriminator to differentiate between real and synthetic nodes and classify whether they belong to the minority class
5. limited to binary classification, multi-class classification would need a generator for each class

Adversarial Generation

SORAG

1. Targets multi-class node classification in graphs
2. An ensemble of a GAN and a conditional GAN (cGAN) as the node generator
3. GAN generated unlabeled synthetic minority nodes
4. cGAN creates labeled synthetic minority nodes
5. connection between synthetic and real nodes is based on the inner

Pseudo Labeling

Label propagation

iteratively propagates a node's label to neighboring nodes based on their proximity.

Pseudo Labeling

Label propagation

iteratively propagates a node's label to neighboring nodes based on their proximity.

Self-Paced Network Representation for Few-Shot Rare Category Characterization (SPARC) and SET-GNN

incorporate pseudo-labeling into self-training to enrich minority class training samples

Pseudo Labeling

Label propagation

iteratively propagates a node's label to neighboring nodes based on their proximity.

Self-Paced Network Representation for Few-Shot Rare Category Characterization (SPARC) and SET-GNN

incorporate pseudo-labeling into self-training to enrich minority class training samples

Distance-wise Prototypical Graph Neural Network (DPGNN)

transfers knowledge from head classes to tail classes using learned class prototypes and metric learning

Pseudo Labeling

GraphSR

generates pseudo-labels for unlabeled nodes using a GNN and identifies the most reliable and informative unlabeled nodes based on their similarity to labeled nodes, adaptively enriching minority class training nodes through reinforcement learning

Model Refinement

Relaxed Similarity and Dissimilarity Network Embedding (RSDNE)

- uses DeepWalk, a method that preserves neighborhood structure based on random walks
- adds auxiliary learning objectives to ensure that the embedding space reflects both intra-class similarity and inter-class dissimilarity

Model Refinement

Relaxed Similarity and Dissimilarity Network Embedding (RSDNE)

- uses DeepWalk, a method that preserves neighborhood structure based on random walks
- adds auxiliary learning objectives to ensure that the embedding space reflects both intra-class similarity and inter-class dissimilarity

ImVerde

- aslo uses DeepWalk
- adjusts the transition probability during random walks to encourage minority nodes to stay within the same class
- also employs context and balanced-batch sampling to sample node-context pairs based on label information and network topology in a class-balanced manner

Model Refinement: Models that modifies AGGREGATE(.)

Attention and Cost-Sensitive Graph Neural Network (ACS-GNN)

- uses an attention mechanism to assign personalized weights to minority and majority samples

Model Refinement: Models that modifies AGGREGATE(.)

Attention and Cost-Sensitive Graph Neural Network (ACS-GNN)

- uses an attention mechanism to assign personalized weights to minority and majority samples

Effective-aggregation Graph Convolutional Network (EGCN)

- limits the aggregation of interclass edges from a local perspective using estimation density and focuses more on the minority class based on the imbalance ratio from a global perspective

Model Refinement

Kernel Propagation-based model for Imbalanced Node Classification in Graph Convolutional Networks (KINC-GCN)

- introduces two modules to enhance node embeddings and exploit higher-order structural features as a pre-processing step before applying GNN for classification
- first is a self-optimizing cluster analysis module that performs clustering on the obtained node embeddings
- second module, the graph reconstruction module, uses an inner product decoder to reconstruct original graphs through reconstructed embedding vectors

Loss Function Engineering

Two ways it is done

1. assigning greater weight to the loss of minority class data samples during training
2. expanding the decision boundary between minority and majority classes

Loss Function Engineering

Two ways it is done

1. assigning greater weight to the loss of minority class data samples during training
2. expanding the decision boundary between minority and majority classes

ReNode

re-weights the influence of labeled nodes based on their relative positions to class boundaries:

$\mathcal{L}_{ReNode} = \frac{1}{|\mathcal{V}^l|} \sum_{v \in \mathcal{V}^l} w_v \frac{|\overline{C}|}{C_{y_v}} \mathcal{L}(l_v, y_v)$,

where w_v is modified training weight of node v, $|\overline{C}|$ is the average number of elements in a class and $|C_{y_v}|$ is the number of labeled samples in the class that v belongs to.

Loss Function Engineering

Topology-Aware Margin (TAM)

considers the local topology of individual nodes and adaptively adjusts margins for topologically improbable nodes: $\mathcal{L}_{TAM} = \frac{1}{|\mathcal{V}^l|} \sum_{v \in \mathcal{V}^l} \mathcal{L}(l_v + \alpha m_v^{ACM} + \beta m_v^{ADM}, y_v)$, where the term $m_v^{ACM} \in \mathbf{R}^K$ adjusts the margin of each class by calibrating the deviation of the neighbor label distribution for node v, $m_v^{ADM} \in \mathbf{R}^K$ modifies the target class margin based on the relative proximity to the target class compared to the self class, and α *and* β are hyper-parameters

Post-hoc Adjustment

Approaches

1. fine-tuning,
2. recalibrating, or
3. aligning model outputs during the inference phase or late stages of training

Post-hoc Adjustment

Approaches

1. fine-tuning,
2. recalibrating, or
3. aligning model outputs during the inference phase or late stages of training

Dual-Regularized Graph Convolutional Networks (DR-GCN)

incorporates a distribution alignment module that ensures unlabeled nodes follow a similar latent distribution to labeled nodes by minimizing the distribution difference based on Kullback-Leibler divergence

Post-hoc Adjustment

Long-Tail Experts for Graph Neural Networks (LTE4G)

- employs a class prototype-based inference method to adjust predictions after the model's training is complete during the inference phase
- involves computing the prototype vector for each class by averaging the embeddings of all nodes in that class
- test node's class is then predicted by calculating its similarity with each class prototype vector and assigning the class with the highest similarity score

Datasets

Domain	Dataset	ρ	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9
Citation network	Cora [68]	5	30.21	15.73	15.43	12.96	11.00	8.01	6.65	-	-	-
	Citeseer [68]	3	21.07	20.08	17.91	17.73	15.26	7.94	-	-	-	-
	PubMed [68]	2	39.94	39.25	20.81	-	-	-	-	-	-	-
Co-purchase network	Amazon-Photo [45]	6	25.37	22.04	11.96	11.53	10.76	9.19	4.82	4.43	-	-
	Amazon-Computers [45]	18	37.51	15.68	15.58	10.28	5.95	3.94	3.54	3.17	2.24	2.12
Social network	Flickr [70]	10	40.26	25.73	9.53	7.19	5.90	5.49	3.90	-	-	-
	GitHub [41]	3	74.17	25.83	-	-	-	-	-	-	-	-
	Facebook [41]	2	30.62	28.91	25.67	14.81	-	-	-	-	-	-
Knowledge graph	Wiki-CS [30]	9	22.90	18.40	16.52	12.17	7.39	6.67	5.70	4.20	3.53	2.52

Future paths

1. work could be done other than node classification like, edge classification, graph classification, or node regression
2. beyond homophilous and homogenous data like heterophilous, heterogernous, hyper, and temporal graphs
3. beyond class and quantity imbalance like, topology imbalance.

Thank you

[Ma+23] Yihong Ma et al. *Class-Imbalanced Learning on Graphs: A Survey*. 2023. arXiv: 2304.04300 [cs.LG].

And almost all the papers in the reference of the above.