TBoost: Gradient Boosting Temporal Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

1 Supplementary Material

1.1 Algorithm

Algorithm 1: TBoost Training Loop

Function `Training_Loop`$(\mathcal{G}, \boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{u,v}(t), y(t))$**:**
- $\boldsymbol{x}^{gbdt}_{u,v}(t) = [\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{u,v}(t), TE(t)]$
- $y^{gbdt}(t) = y(t)$
- **for** *iter* i*=1 to* N **do**
  - `// Train` k `decision trees`
  - $f_i \overset{k}{\leftarrow} \arg\min_{f_i} \mathcal{L}^{gbdt}(f_i(\boldsymbol{x}^{gbdt}_{u,v}(t)), y(t))$
  - $f = f + f_i$;
  - `// Prepare GNN input features`
  - $\bar{\boldsymbol{x}}_{u,v}(t) = [\boldsymbol{x}^{gbdt}_{u,v}(t), \hat{y}^{gbdt}_{u,v}(t)]$
  - $\boldsymbol{x}^{gnn}_{u,v}(t) = [\boldsymbol{x}_u, \boldsymbol{x}_v, \bar{\boldsymbol{x}}_{u,v}(t)]$
  - `// Train` l `epochs of GNN`
  - $\theta, -\eta \frac{\partial \mathcal{L}_{gnn}}{\partial \boldsymbol{x}^{gnn}_{u,v}(t)} \leftarrow$ Equation 4
  - `// update the target for the next iteration of GBDT`
  - $y^{gbdt}(t) = -\eta \frac{\partial \mathcal{L}_{gnn}}{\partial \boldsymbol{x}^{gnn}_{u,v}(t)}$
- **end for**
- **Output**: GBDT f, GNN g_θ

1.2 Baselines

We evaluate TBoost against the following baselines:

- **GBDT Baselines**: CatBoost[9] and XGBoost[1] are commonly used gradient boosting algorithms. They differ in handling categorical variables, missing values, and overfitting prevention. The models are trained on the concatenated node and edge features of datasets. We also implement another version of GBDTs where the model is trained on concatenated temporal encoding features as given in Equation 3. These models are called XGBoost+ and CatBoost+ respectively.
- **Temporal GNN Baselines**: In our study, we utilized several state-of-the-art temporal graph neural network (GNN) models, including JODIE [4], DyRep [11], and TGAT [12], which are specifically designed for learning on dynamic graphs. JODIE [4] is a neural network model that learns bipartite embeddings for both nodes and edges in a dynamic graph by modeling

Submitted to Temporal Graph Learning Workshop @ NeurIPS 2023, New Orleans. Do not distribute.

the interactions between nodes as a sequence of edge snapshots over time. DyRep [11], on the other hand, uses temporal point processes to model the evolution of the graph as a sequence of events and predict user-item interactions in temporal graphs where the structure of the graph changes over time. TGAT [12] extends the basic graph attention network (GAT) to incorporate temporal information, allowing it to capture the dynamics of the graph over time. TGN [10] combines RNN-based temporal modeling with graph attention convolution. It captures temporal information using an RNN and incorporates self-attention mechanisms to model spatial and temporal dynamics jointly. To evaluate the baselines, we used public PyTorch implementation of the repositories released by the authors.

1.3 Datasets

Table 1: Dataset statistics

Dataset	Vertices	Edges	Timestamps	Classes
Wikipedia	9,227	1,57,474	1,52,757	2
Reddit	10,984	6,72,447	6,69,065	2
MOOC	7145	411,749	345,600	2
Fashion	437	3,176	344	5
Luxury	5,400	34,278	2,510	5
Office	129,466	800,357	5,348	5
Payments	367,221	1,063,398	544,724	2
BankSim	4,162	594,643	180	2

We consider real-world datasets Wikipedia [4], Reddit [4] and MOOC [4] along with simulated financial datasets Payments [2], BankSim [5] for the temporal node classification task and Amazon [7] for the temporal edge classification task. The dataset statistics are shown in Table 1:

- **Wikipedia** [4] The dataset consists of one month's worth of Wikipedia page modifications, focusing on the top 1,000 pages with the most revisions and 8,227 editors with at least five edits as users. User modifications are transformed into 172-dimensional LIWC [8] feature vectors.
- **Reddit** [4] This dataset is a temporal graph of active users and their posts under subreddits as 11,000 nodes, 700,000 temporal edges, and dynamic labels suggesting whether a user has been banned from posting. Edge feature vectors are created from the user post in the temporal setting. Here, we are trying to predict whether a user has been banned to post.
- **MOOC** [4]: This public dataset consists of actions, e.g., viewing a video, submitting an answer, etc., done by students on a MOOC online course. This dataset consists of 7,047 users interacting with 98 items (videos, answers, etc.) resulting in over 411,749 interactions. The task is to predict one of the 4,066 drop-out events.
- **Payments Data**[1] [2] The data consists of subject-centric transaction records aimed at detecting fraudulent activities, encompassing both normal and abnormal transactions with predefined probabilities. It was generated using an AI planning execution simulator, translating output planning traces into a tabular format, with parameters like the number of clients, time duration, and fraud probabilities configurable.
- **BankSim** [5] BankSim is an agent-based simulator generating synthetic data for fraud detection research, based on Spanish bank transaction data. It models merchant-customer relationships and aims to represent normal payments and known fraud patterns.
- **Amazon** [Fashion, Luxury and Office] [7] The Amazon review dataset is a vast collection of product reviews from Amazon.com, comprising reviews with text, product IDs, reviewer IDs, ratings, and timestamps. Word2Vec [6] embeddings of the reviews are utilized as features, while user ratings serve as the labels for analysis.

[1]This publication includes or references synthetic data provided by J.P. Morgan.

1.4 Model Training

Tboost is an end-to-end framework that is trained in an alternate fashion using Adam [3] optimizer. TBoost is capable of accommodating any temporal GNN model along with GBDT. We experimented with CatBoost as a GBDT module. In temporal GNN, either TGAT or TGN is employed as these models perform well over other baselines. Based on our experiments with TBoost, we found that a single layer of GNN with two attention heads is sufficient to model the data for attention-based models. We used a learning rate of $1e^{-3}$ for Wikipedia and Reddit and a learning rate of $1e^{-5}$ for other datasets. The neighborhood size was set to 10 for all models, with a batch size of 30 for Wikipedia, 200 for Reddit, and 100 for the other datasets. For tree-based models, we used a learning rate of $1e^{-3}$, a maximum depth of 7, and a maximum of 1000 iterations. We also employed early stopping with a patience of 5. Trained models are evaluated for node classification and edge classification tasks. For the node-classification task, due to the label imbalance in the datasets, we employ the area under the ROC curve (AUC) as done in prior arts [12, 10]. In the multiclass edge classification task, the number of classes is 5 for all the datasets in our experimentation. We use the macro-F1 score as the evaluation metric for these tasks. We carry out each experiment 10 times and report the mean numbers in Table 2 and Table 3 in the main material.

1.5 Ablation study

Table 2: Impact of time-encoding on GBDT for the temporal node classification task. In TE_1 temporal feature is not used, in TE_2 absolute value of timestamp is used, in TE_3 Bochner's kernel is used and in TE_4 inter arrival time is used.

Model	**CatBoost**				**XGBoost**			
	TE_1	TE_2	TE_3	TE_4	TE_1	TE_2	TE_3	TE_4
Reddit	**56.62**	56.23	54.30	56.59	53.34	**57.30**	51.45	54.78
Wiki	59.30	63.84	62.54	**65.75**	55.50	64.74	59.37	**64.91**
MOOC	59.73	60.11	**60.26**	60.00	59.82	**60.10**	57.75	59.81
Payments	52.70	50.10	52.50	**56.10**	52.20	53.07	51.11	**54.60**
BankSim	52.70	51.47	51.30	**52.90**	52.20	50.30	51.06	**52.25**
Average	56.21	56.35	56.18	**58.27**	54.61	56.70	54.10	**57.27**

Effect of temporal encoding: GBDT models, while widely used for various supervised learning tasks, are not inherently designed to handle time-series data. There is no built-in mechanism within the algorithm to handle temporal aspects. Several approaches have been explored in the literature to encode time information in GBDTs. These methods include incorporating time as a feature (TE_2) in Table 2, adopting functional encoding inspired by Bochner's Kernel (TE_3), and utilizing inter-event time (TE_4), among others(Refer Table 2). Due to the discrete nature of GBDTs, we used a non-trainable version of the temporal kernels. We report the ROC-AUC in Table 2 for the node classification task averaged over 10 runs. We conducted experiments using two extensively used GBDT algorithms, CatBoost [9] and XGBoost [1]. Our results showed consistent performance improvements by incorporating time information, but the candidate for best encoding changed across different datasets. On average, Bochner's encoding did not yield substantial improvements in GBDTs, performing similarly to models without time information. On the other hand, adding time as a feature had better results. Further, the best results were obtained with inter-event time.

Effect of pretraining: In the temporal GNN literature, one of the common ways to benchmark node classification is by pre-training the GNN with an unsupervised objective like contrastive link prediction. For downstream tasks, the GNNs are initialized with the pre-trained weights and are kept non-trainable. Only the task-specific head is trained in a supervised fashion. In this section, we tried to understand the impact of pretraining and weight fixing in TBoost for the node classification task. The results are shown in the Table 3. In the first setting ($TBoost_1$), we train Tboost without using pre-trained weights. In the second setting ($TBoost_2$), we initialize the GNN with pre-trained weights that are eventually frozen during training. Only the classifier layer is trained through backpropagation. In the third setting ($TBoost_3$), we initialize the GNN with pre-trained weights but don't freeze them. We keep the entire framework trainable. Table 3 shows that pre-training helps boost performance in most cases. We have used the same in our implementation of TBoost.

Table 3: Effect of pretraining on link classification. TBoost_1 is without pre-trained GNN model. TBoost_2 is initialized with pre-trained TGAT but the GNN weights are frozen. In TBoost_3, the pre-trained weights are kept trainable.

Model	**Reddit**	**Wikipedia**	**MOOC**	**Payments**	**BankSim**
TBoost_1	65.80	85.28	51.31	56.31	77.65
TBoost_2	66.03	87.50	**59.32**	57.37	78.10
TBoost_3	**66.09**	**88.40**	58.40	**62.10**	**78.15**

Keeping the GNN parameters trainable gives a further boost compared to the frozen parameter setting. However, for a fair comparison with baselines in Table 3, we report only the fixed weighted results.