EVENT DETECTION AND FORECASTING USING GNNS ON TEMPORALLY DYNAMIC GRAPHS (PROGRESS UPDATE)

Ding, Yanna **Gao, Suyi (Stanley)** **Saha, Bishwajit**

ABSTRACT

This project utilizes the Knowledge-Preserving Incremental Heterogeneous Graph Neural Network (KPGNN) architecture to detect social events using data generated from social networks, such as Twitter. To improve the performance of KPGNN in the clustering task, we incorporated KL-divergence loss during training. Additionally, we explored the use of Graph Neural Ordinary Differential Equations (GDE) to convert the discrete updates of the GNN layer into continuous updates modeled by an ODE.

1 INTRODUCTION

Understanding group social behaviors and public concerns is crucial in various fields, and analyzing social events can aid in achieving this goal. Social networks such as Twitter and Reddit provide a continuous platform for user interaction, generating a vast amount of textual content over time. Thus, detecting social events is essential as it can provide timely and relevant information, which can aid in formulating appropriate responses. Consequently, social event detection is a crucial tool in crisis management, product recommendation, decision-making, and other fields.

Social streams, particularly those found on Twitter, are more complex than traditional news and articles for several reasons. Firstly, social streams are generated sequentially and are considerably voluminous. Moreover, they contain diverse elements, including text, time, hashtags, and implicit social network structures. Additionally, their contents are often brief and contain abbreviations that are not present in standard dictionaries. Lastly, the meaning of elements within social streams changes rapidly, further adding to their complexity.

Social event detection involves extracting clusters of interconnected messages from social streams and is closely related to clustering, as events can only be inferred from shifts in aggregate trends in the stream. This approach enables us to identify social events and obtain valuable insights into group social behaviors and public concerns. Social streams can consist of a sequence of social media messages, such as tweets or Facebook posts, generating a vast amount of textual content over time.

2 METHODOLOGY

In this project, we start with the Knowledge-Preserving Incremental Heterogeneous Graph Neural Network (KPGNN) architecture (Cao et al., 2021), which is an incremental learning model that leverage document-pivot method (Aggarwal & Subbian (2012); Hu et al. (2017); Peng et al. (2019); Zhang et al. (2007)) and classifies social messages by using their correlations.

Formally, we define a **social stream** as sequence of message blocks

$$S = \{M_0, M_1, \cdots, M_i, M_{i+1}, \cdots\},$$

where each message block M_i contains the messages m_j collected in some time interval $[t_i, t_{i+1})$. And each message m_j contains heterogeneous data such as user account, time stamp, and text document. We denote a **social event** $e = \{m_i\}$ as a set of messages that is discussing the same real-world event. The goal is to use an **incremental social event detection** algorithm to learn a sequence of models $\{f_0, \cdots, f_{t-\Delta t}, f_t, \cdots\}$ such that

$$f_t(M_i; \theta_t, \theta_{t-\Delta t}) = E_i \ ,$$

where $E_i = \{e_k\}$ is a collection of social events e_k contained in message block M_i, and Δt is the information collecting and updating interval. Note that f_t is controlled by the parameter $\theta_{t-\Delta t}$ of the previous model $f_{t-\Delta t}$ in the sequence. This means each current model preserves the knowledge obtained from the predecessor model.

To construct the graph representation $\mathcal{G}$ of social media messages, we first convert them to heterogeneous information networks (HINs) (Cao et al. (2020); Peng et al. (2019)) and map HINs to plain graphs whose nodes represent messages. HIN leverages the inherent relationships between various elements (e.g., user name, location). For a message m_i, the named entities and words are extracted from m_i. The messages and extracted information together form the node set of an HIN. Edges are formed between tweets and the entities contained in them. There are four types of nodes in an HIN: message, named entity, user, and word. Let $A \in \{0,1\}^{N \times N}$ denote the adjacency matrix of the $\mathcal{G}$. $A_{ij} = \min\{\sum_k \left[W_{mk} W_{mk}^\top\right]_{ij}, 1\}$ W_{mk} stores the connection between nodes of type message and type $k \in \{\text{named entity}, \text{user}, \text{word}\}$. $[W_{mk} W_{mk}^\top]_{ij}$ is the similarity between message m_i and m_j measured from the perspective of node type k. Two messages are connected in $\mathcal{G}$ if they are linked to the same node of type k for some $k \in \{\text{named entity}, \text{user}, \text{word}\}$ in the HIN.

2.1 Machine Learning Architecture

The main workflow of KPGNN is divided into following stages: (i) Input: A social stream $S = \{M_0, M_1, \cdots\}$, update interval Δt, number of GNN layers L, number of mini-batches B. (ii) Update the message graph $\mathcal{G}$ using message block M_t. (iii) Detect event E_t from M_t using model f_t trained in last loop. (one forward pass) (iv) REPEAT: Stage 1 and Stage 2, until $t \ \% \ \Delta t = 0$. (v) Remove obsolete messages from graph $\mathcal{G}$. (vi) Train the next model $f_{t+\Delta t}$ using mini-batches. (elaborated in Algorithm 1) (vii) Output: sequence of predicted events $\{E_0, E_1, \cdots\}$, collected from Stage 2.

2.2 Algorithm(s)

This is the subsection for algorithm(s) that is used. An example algorithm is provided for reference.

The original encoder in vanilla KPGNN is a commonly used Graph Attention Network (GAT) layer (Vaswani et al., 2017), defined as following:

$$\mathbf{h}_{m_i}^{(l)} \leftarrow \Big\|_{\text{heads}} \left(\mathbf{h}_{m_i}^{(l-1)} \oplus \underset{m_j \in \mathcal{N}(m_i)}{Aggregator} \left(Extractor \left(\mathbf{h}_{m_i}^{(l-1)} \right) \right) \right)$$

where the $Extractor$ pass the data in nodes from previous layer through an affine transformation, parameterized for learning, the $Aggregator$ collect information from adjacent nodes to update the current node, and $\|_{\text{heads}}$ denotes head-wise concatenation (Vaswani et al., 2017).

We replace the original encoder with a continuous version of layered graph neural networks (Poli et al. (2019)) and refer it to the Graph Ordinary Differential Equation (GODE) model. The embedding $\mathbf{h}_{m_i}$ for node i is obtained by solving a coupled ODE system.

$$\frac{d\mathbf{h}_{m_i}}{dt} = f_{\text{ode}}(\mathcal{G}, \mathbf{h}_{m_1}, \ldots, \mathbf{h}_{m_n})$$

Algorithm 1: Model training in mini-batches (Stage 5)

Input: Batch size β, number of batches B, number of layers L

function train($\mathcal{G}, B, \beta, L$)
  for $b = 1, 2, \cdots, B$ **do**
    $M^{(b)} = \{m_i\}_{i=1}^{\beta} \leftarrow$ sample messages from $\mathcal{G}$
    for $l = 1, 2, \cdots, L$ **do**
      $\mathbf{h}_{m_i}^{(l)} \leftarrow$ from eq. (1) `// (new) GDE encoder`
    $\mathbf{h}_{m_i} \leftarrow \mathbf{h}_{m_i}^{L}$ among $M^{(b)}$
    $T = \left\{(m_i, m_i^+, m_i^-)\right\}_{i=1}^{\beta} \leftarrow$ triplet sampling from $M^{(b)}$
    $\mathcal{L}_t \leftarrow$ from eq. (2) `// triplet loss`
    $\mathbf{s}, \tilde{\mathbf{h}}_{m_i} \leftarrow$ calculate summary and corrupted representation
    $\mathcal{L}_p \leftarrow$ from eq. (3) `// cross entropy loss`
    $\mathcal{L}_{kl} \leftarrow$ from eq. (4) `// (new) KL divergence loss`
    Back-propagation to update parameters

We show an instance of f_{ode} as a graph convolutional layer in the following equation. The expression for the hidden state of message m_i can be written as:

$$\mathbf{h}_{m_i} = \mathbf{z}_{m_i} + \int_{t_0}^{t_1} \sum_{v \in \mathcal{N}(m_i)} \sigma(W\mathbf{z}_v + b)/k_v \, dt \quad \mathbf{z}_{m_i} = f_{\text{enc}}(\mathbf{x}_{m_i}) \tag{1}$$

k_v denotes the degree of node v and $\mathcal{N}(m_i)$ are the neighbors of message i. We learn an encoder function f_{enc} and the parameters in the ODE. The encoder f_{enc} can be a generic graph attention network or graph convolutional network. The learnable parameters in the differential equations are $W \in \mathbb{R}^{d' \times d'}$ and $b \in \mathbb{R}^{d'}$, where d' is the hidden dimension. We assume the time steps are normalized, i.e., $t_0 = 0$, $t_1 = 1$.

One of the loss used in vanilla KPGNN is Triplet Loss:

$$\mathcal{L}_t = \sum_{(m_i, m_i^+, m_i^-) \in T} \max\left\{0, \mathcal{D}(\mathbf{h}_{m_i}, \mathbf{h}_{m_i^+}) - \mathcal{D}(\mathbf{h}_{m_i}, \mathbf{h}_{m_i^-}) + a\right\} \tag{2}$$

where m_i^+ is a message sampled from the same class as m_i, m_i^- is a message sampled from the different class as m_i, $\mathcal{D}(\cdot, \cdot)$ is the l_2 distance between two vectors, and $a \in \mathbb{R}$ is a hyperparameter controls the penalty between different messages. The other loss used in vanilla KPGNN is global-local pair cross-entropy Loss:

$$\mathcal{L}_p = \frac{1}{N} \sum_{i=1}^{N} \log \mathcal{S}(\mathbf{h}_{m_i}, \mathbf{s}) + \log\left(1 - \mathcal{S}(\tilde{\mathbf{h}}_{m_i}, \mathbf{s})\right) \tag{3}$$

where $\mathbf{s} \in \mathbb{R}^d$ is the summary of the message m_i, taken to be the average among nodes, $\tilde{\mathbf{h}}_{m_i}$ is a corrupted representation of m_i obtained by passing an row-wise shuffled data set to the model. and $\mathcal{S}(\cdot, \cdot)$ is a joint distribution that marginalize to its two input in each orientation.

We propose to include a KL-divergence loss (Xie et al. (2016)) that measures the quality of clustering.

$$\mathcal{L}_{kl} = \mathrm{KL}(P\|Q) = \sum_{ij} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) \tag{4}$$

Where Q is the predicted soft assignment and P is the target distribution. For computing Q, we use the Student's t-distribution as a kernel to measure the similarity between embedded point z_i and centroid μ_j.

$$q_{ij} = \frac{\left(1 + \frac{1}{\alpha}\|z_i - \mu_j\|^2\right)^{-\frac{1}{2}(\alpha+1)}}{\sum_{j'}\left(1 + \frac{1}{\alpha}\|z_i - \mu_{j'}\|^2\right)^{-\frac{1}{2}(\alpha+1)}}$$

Here, $z_i \in \mathbb{R}^{d'}$ corresponds to $x_i \in \mathbb{R}^d$ after embedding, α is the degrees of freedom of the Student's t- distribution. We can interpret q_{ij} as the probability of assigning sample i to cluster j (a soft assignment). We set $\alpha = 1$ for all experiments for simplicity.

In our experiments, we compute p_i by first raising q_i to the second power and then normalizing by frequency per cluster. We followed these general definitions of Q and P from (Xie et al., 2016).

$$p_{ij} = \frac{q_{ij}^2 \Big/ \sum_i q_{ij}}{\sum_{j'}\left(q_{ij'}^2 \Big/ \sum_i q_{ij'}\right)}$$

3 EXPERIMENTS

3.1 EXPERIMENT SETUP

We conduct experiments on a subset of the Twitter dataset McMinn et al. (2013). Message block 0 includes 500 messages. Blocks 1-21 contains 100 messages and the corresponding heterogeneous knowledge representation graph consist of 100 nodes.

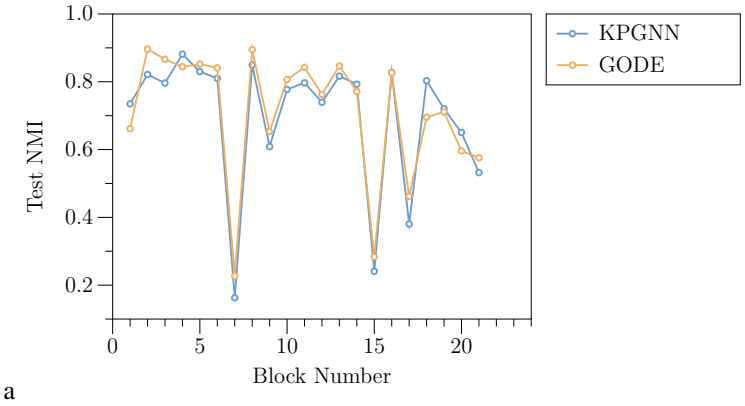
We adopt the *latest message strategy* to train the model Cao et al. (2021). In the initial stage, the model is trained on message block M_0. For blocks $i = 1, \ldots, 21$, we first perform inference to get the test NMI for block i and if $i =$ *window size*, we train T iterations on block i.

3.2 GODE VS KPGNN

In this section, we compare the performance of GODE and KPGNN and test model sensitivity to hyperparameters. Figure 1 shows GODE slightly outperforms KPGNN for 14 blocks. When training on Block 0, the choice of hidden dimension donimates the performance, compared to window size.

Table 1: Graph statistics for social stream.

Blocks	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
# of Edges	5918	409	346	233	384	520	356	652	491	658	191
Blocks	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}	M_{21}
# of Edges	492	375	269	543	210	344	178	874	341	497	349



a

Figure 1: Test NMI for KPGNN and GODE.

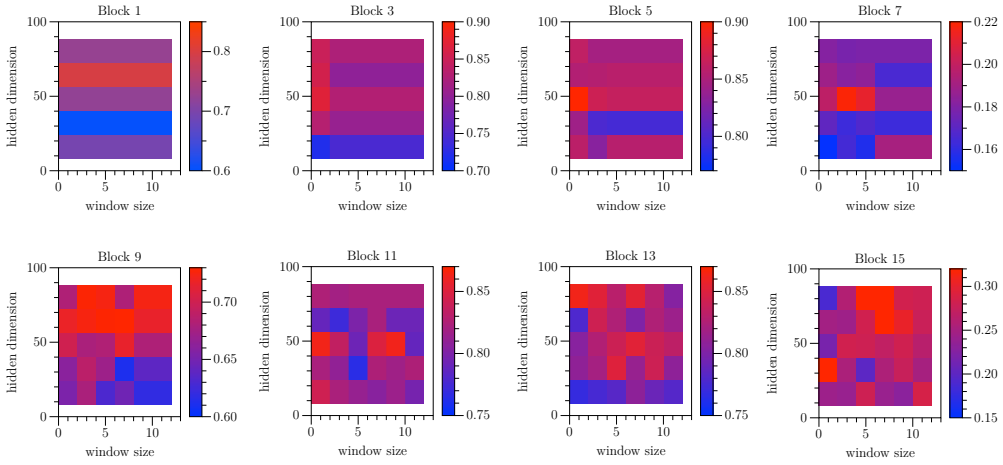


Figure 2: GODE with different hyperparameters. We fix the batch size at 150 and vary window size from $\{1, 3, 5, 7, 9\}$ and the embedding dimension from $\{16, 32, 48, 64, 80\}$.

3.3 KPGNN vs KPGNN with KL Divergence loss

Figure 3 and Figure 4 show the improvement by adding KL-divergence loss over the KPGNN on both training and inference stages respectively.

4 Further Works

These are the further works that can be done before project completion: (i) merge the two changes: GDE encoder and KL-divergence loss. (ii) modify the architecture to learn cluster centers μ_i for KL-divergence loss. (iii) fine tune GDE encoder for better performance. (iv) incorporate automorphic templates when aggregating information for each node.

References

Charu C Aggarwal and Karthik Subbian. Event detection in social streams. In Proceedings of the 2012 SIAM international conference on data mining, pp. 624–635. SIAM, 2012.

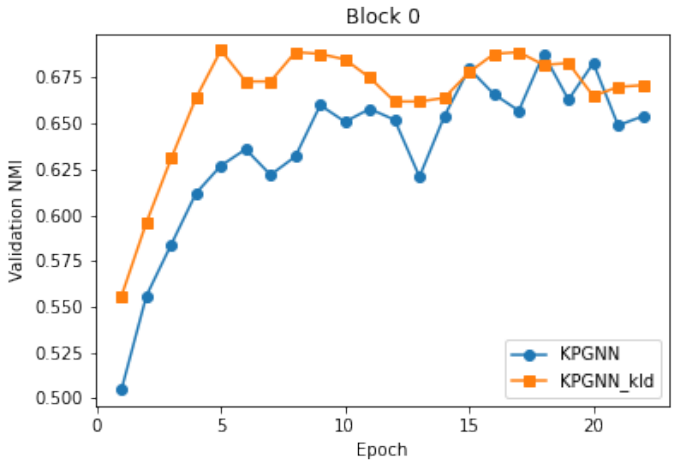


Figure 3: KPGNN vs KPGNN with kld loss at training stage.

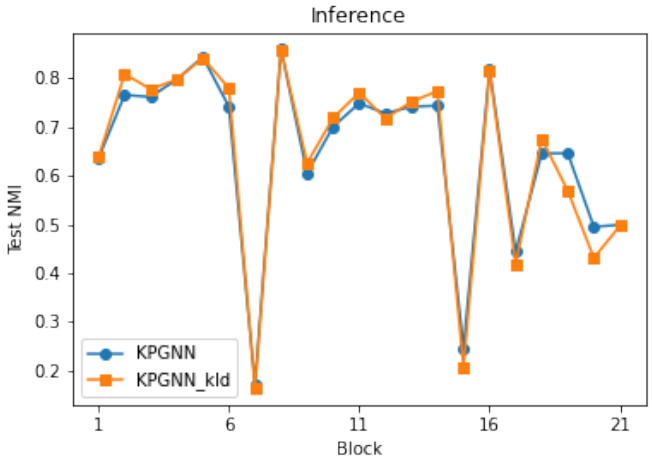


Figure 4: KPGNN vs KPGNN with kld loss at inference stage.

Yuwei Cao, Hao Peng, and Philip S Yu. Multi-information source hin for medical concept embedding. In Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part II 24, pp. 396–408. Springer, 2020.

Yuwei Cao, Hao Peng, Jia Wu, Yingtong Dou, Jianxin Li, and Philip S Yu. Knowledge-preserving incremental social event detection via heterogeneous gnns. In Proceedings of the Web Conference 2021, pp. 3383–3395, 2021.

Linmei Hu, Bin Zhang, Lei Hou, and Juanzi Li. Adaptive online event detection in news streams. Knowledge-Based Systems, 138:105–112, 2017.

Andrew J McMinn, Yashar Moshfeghi, and Joemon M Jose. Building a large-scale corpus for evaluating event detection on twitter. In Proceedings of the 22nd ACM international conference on Information & Knowledge Management, pp. 409–418, 2013.

Hao Peng, Jianxin Li, Qiran Gong, Yangqiu Song, Yuanxing Ning, Kunfeng Lai, and Philip S Yu. Fine-grained event categorization with heterogeneous graph convolutional networks. arXiv preprint arXiv:1906.04580, 2019.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. arXiv preprint arXiv:1911.07532, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In International conference on machine learning, pp. 478–487. PMLR, 2016.

Kuo Zhang, Juan Zi, and Li Gang Wu. New event detection based on indexing-tree and named entity. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 215–222, 2007.