EVENT DETECTION AND FORECASTING USING GNNS ON TEMPORALLY DYNAMIC GRAPHS

IMPORVED WITH KL-DIVERGENCE LOSS AND GRAPH-ODE ENCODER

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 or Reddit. 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. We used a subset of Twitter data set to implement the GNN, and obtained a better preforming result than the original KPGNN.

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.

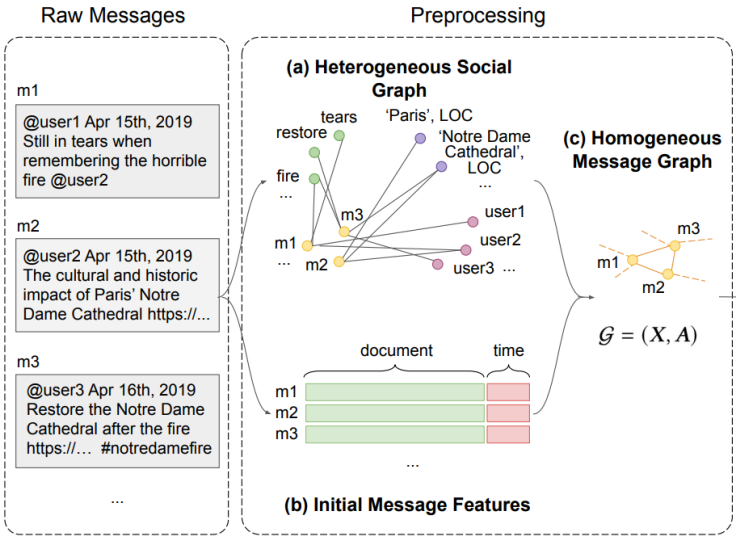

Figure 1: Construction of homogeneous graph $\mathcal{G}$ using heterogeneous message data block M_i Cao et al. (2021)

Formally, we define a **social stream** as sequence of message blocks, e.g., the message stream from twitter in a day,

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

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

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

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 begin by converting the initial message block M_0 into heterogeneous information networks (HINs) (Cao et al. (2020); Peng et al. (2019)). These HINs comprise various types of nodes, such as user accounts, message keywords, and location mentions, but the most crucial is the node representation of the message $m_1, m_2, \cdots$, which is illustrated in fig. 1. By leveraging the inherent relationships between different elements, HINs enable us to capture the complex interconnections between them. To create an HIN for a message m_i, we extract the named entities and words from it, which together with the message itself form the node set of the HIN. The edges are then formed between the tweets and the entities mentioned in them. Then we convert the HIN to a homogeneous graph sG by removing all except the message nodes m_i from the HIN, then we connect messages that shares any same attribute, such as same user, same keyword mentioned, or same location mentioned. The adjacency matrix of the resulting graph $\mathcal{G}$ is denoted by $A \in \{0, 1\}^{N \times N}$.

$$A_{ij} = \min\left\{\sum_k \left[W_{mk} W_{mk}^\top\right]_{ij}, 1\right\}$$

where W_{mk} stores the connection between nodes of type message and type $k \in \{\text{named entity}, \text{user}, \text{word}\}$. $\left[W_{mk}W_{mk}^{\top}\right]_{ij}$ 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:

Stage 0) Input: A social stream $S = \{M_0, M_1, \cdots\}$, update interval Δt, number of GNN layers L, number of mini-batches B.
Stage 1) Update the message graph $\mathcal{G}$ using message block M_t.
Stage 2) Detect event E_t from M_t using model f_t trained in last loop. (one forward pass)
Stage 3) REPEAT: Stage 1 and Stage 2, until $t \,\%\, \Delta t = 0$.
Stage 4) Remove obsolete messages from graph $\mathcal{G}$.
Stage 5) Train the next model $f_{t+\Delta t}$ using mini-batches. (elaborated in Algorithm 1)
Stage 6) 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.

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

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

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

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 and refer it to the Graph Ordinary Differential Equation (GODE) model. The embedding $\mathbf{h}_{m_i}$ for node

i is obtained by solving an initial value problem (IVP) for a coupled ODE system.

$$\mathbf{h}_{m_i}^{(t_1)} = \mathbf{h}_{m_i}^{(t_0)} + \int_{t_0}^{t_1} \underset{m_j \in \mathcal{N}(m_i)}{Aggregator}\big(Extractor(\mathbf{h}_{m_j}(t))\big)dt$$

where $\mathbf{h}_{m_i}^{(t_0)}$ is the initial condition of the hidden vector. The corresponding discrete transformation is

$$\mathbf{h}_{m_i}^{(t)} = \mathbf{h}_{m_i}^{(t-1)} + \underset{m_j \in \mathcal{N}(m_i)}{Aggregator}\Big(Extractor(\mathbf{h}_{m_i}^{(t-1)})\Big)$$

We show an instance of the extractor and aggregator as a graph convolutional layer in the following equation. The expression for the hidden state of message m_i can be written as:

$$\mathbf{h}_{m_i}^{(T)} = \mathbf{h}_{m_i}^{(0)} + \frac{1}{\sqrt{k_i}} \int_0^T \sigma\left[\sum_{m_v \in \mathcal{N}(m_i)} \frac{1}{\sqrt{k_v}}(W\mathbf{h}_{m_v}(t) + b)\right] \mathrm{d}t\,, \quad \mathbf{h}_{m_i}^{(0)} = f_{\text{enc}}(\mathbf{x}_{m_i}) \tag{1}$$

k_v denotes the degree of message m_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., $t_0 = 0$, $t_1 = 1$. Compared to the discrete update equation, the ODE-based function can explore more values in the feature space Chen et al. (2018).

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

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

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:

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

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 shuffled data set to the model. and $\mathcal{S}(\cdot, \cdot)$ is a bilinear scoring function that outputs the probability of its two operands coming from a joint distribution.

We propose to include a KL-divergence loss, also known as Kullback-Leibler divergence loss that is used to measure the distance between two distributions. The motivation for utilizing this loss is that it is tightly linked with the clustering part and we hope by including this clustering loss in the core model, the clustering quality should be improved. Finally, we obtain better clustering results by including this loss in the model.

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

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.

Here the other part, which is the definition of P is really important for the clustering loss. It is vital that P needs to be defined in a systematic way otherwise it is possible that the loss doesn't make sense at all and works as random noise in the objective function. A naive approach would be setting it to a delta distribution (to the nearest centroid) for data points above a confidence threshold and ignoring the rest. However, because q_i is soft assignment, it is more flexible to use softer probabilistic targets. In our experiments, we compute p_i by first raising q_i to the second power and then normalizing by frequency per cluster. Here p_i is defined by q_i and the intent of optimization is to match Q with P, thus we can regard it as a kind of self-training. This definition of P helps to strengthen the predictions by putting more importance on data points assigned with high confidence that eventually improves cluster purity. It also helps to normalize the loss contribution of each centroid to prevent large clusters from distorting the hidden feature space. 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 manually labeled Twitter dataset McMinn et al. (2013), including 2600 messages related to 8 events, spanning over a period of 4 weeks. The messages are split into blocks according to the timeline. The tweets from the first week form message block 0 (M_0), including 500 messages. The rest of the tweets are evenly split into 21 blocks, each containing 100 messages and the corresponding heterogeneous knowledge representation graph consist of 100 nodes. The number of edges for each message graph is given in Table 1.

The experiments are conducted on a 28-core Intel Xeon Gold 6330 CPU @2.00GHz. We evaluate the models by the similarity between predicted clusters and ground truth clusters, using normalized mutual information (NMI, Estévez et al. (2009); Liu et al. (2020); Peng et al. (2019)). The NMI ranges from 0 to 1. A value of 1 indicates perfect clustering while a value of 0 indicates completely wrong class labels. Higher is better.

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 $i = 0 \bmod$ *window size*, we train T iterations on block i. We randomly sample 70%, 20%, 10% from each message block to form the corresponding train, test, and validation set.

3.2 GODE vs KPGNN

In this section, we compare the performance of GODE and KPGNN and test model sensitivity to hyperparameters. We trained both models for 30 epochs for each block. Figure 2 shows GODE slightly outperforms KPGNN for 14 blocks. We assume the time range is normalized and integrate over the time range $[0, 1]$ using Runge-Kutta4 (Dormand & Prince (1980)). The choice of hidden dimension donimates the performance, compared to window size. The maximizing combination of hidden dimension and window size varies for different blocks. Overall, models with a larger hidden dimension and a window size between 5 and 7 detect clusters closer to the ground truth. The average runtime to train and maintain through all blocks for KPGNN and GODE is 33.75s and 37.07s, respectively. The runtime of GODE is minimally impacted by numerical integration. The overall training loss for KPGNN is 13.1244, and that for GODE is 3.7204.

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

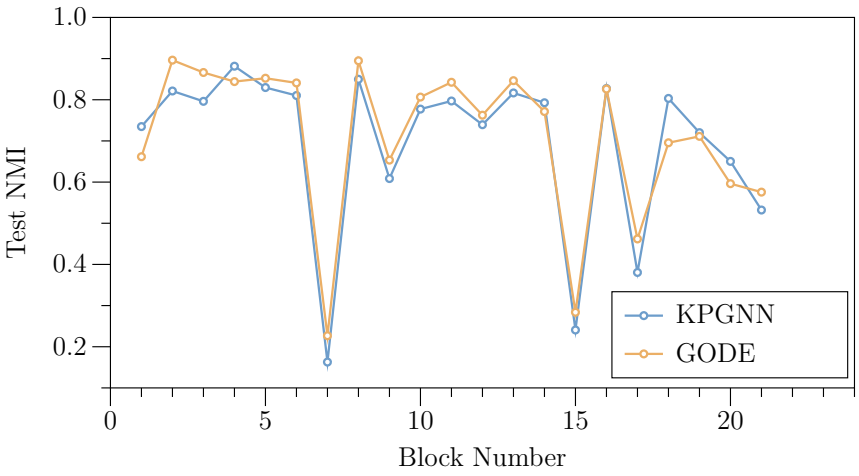

Figure 2: Test NMI for KPGNN and GODE.

3.3 KPGNN vs KPGNN with KL Divergence loss

In this section, we compare the performance of KPGNN and after adding KL-divergence loss to the KPGNN. We used the default hyper-parameter setting for KPGNN where the hidden dimension is set to 32. We run 25 epochs and from Figure 4, we can see after adding KL-divergence loss over the KPGNN, it outperforms the vanilla KPGNN model for almost all epochs at the training phase on block-0. Figure 5 represents the test NMI on each of block-1 to block-21 at the inference stage. Here, we can see KPGNN with kld loss again outperforms vanilla KPGNN for most of the blocks which supports our assumption that integrating kld loss for the clustering part will improve the clustering quality.

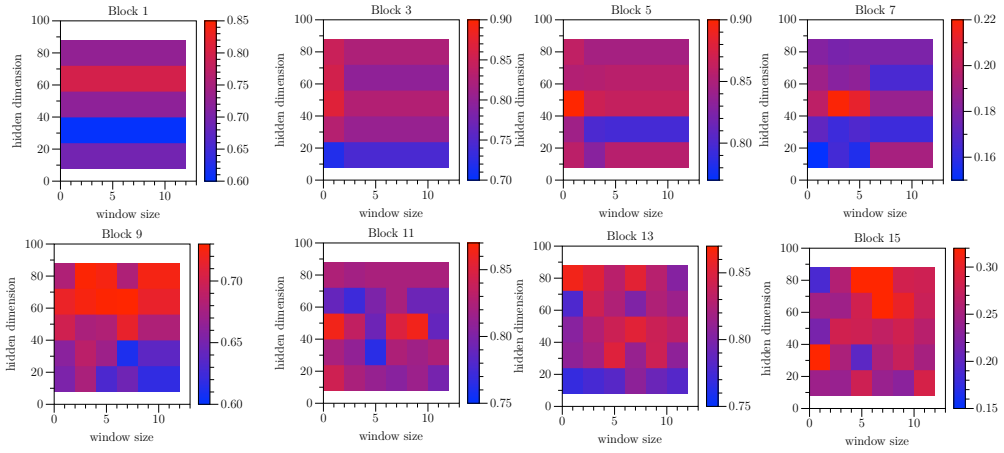

Figure 3: GODE with different hyperparameters. We fix the batch size at 150 and vary window size from $\{1, 3, 5, 7, 9\}$ and the embedding dimension from $\{16, 32, 48, 64, 80\}$.

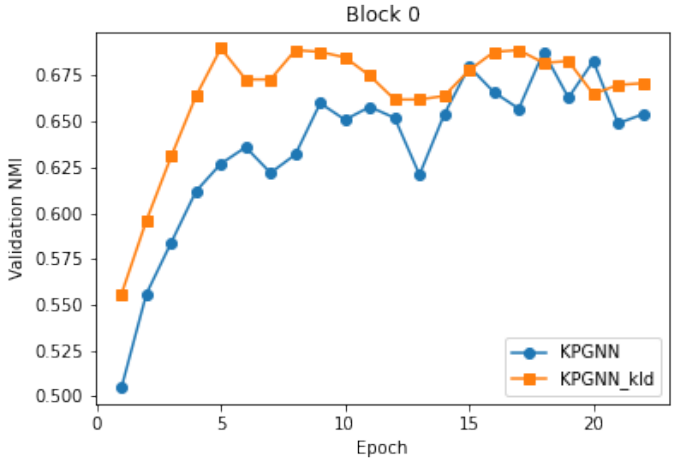

Figure 4: KPGNN vs KPGNN with kld loss at training stage.

3.4 Ablation Study

We perform ablation study and compare the following three models: (i) KPGNN, (ii) KPGNN + GODE, and (iii) KPGNN + GODE + KLD. We tune the hidden dimension, window size, and model weight initialization and report the best test NMI for the three methods in Table 2. Our models outperform KPGNN on all blocks except for M_4, M_{14}, M_{20}. Model (iii) scores the highest for 12 blocks and model (ii) 6 blocks. The predicted events are improved for block 7 and block 15, which are the two most difficult message blocks to predict in the original paper. Overall, KPGNN combined with GODE and KL-divergence loss most accurately predicts events from social media messages.

4 Conclusion

In conclusion, this project demonstrates the effectiveness of using Knowledge-Preserving Incremental Heterogeneous Graph Neural Network (KPGNN) architecture, augmented with KL-divergence loss during training and Graph Neural Ordinary Differential Equations (GODE), to detect social events in social networks such as Twitter. The ability to detect social events in real-time is essen-

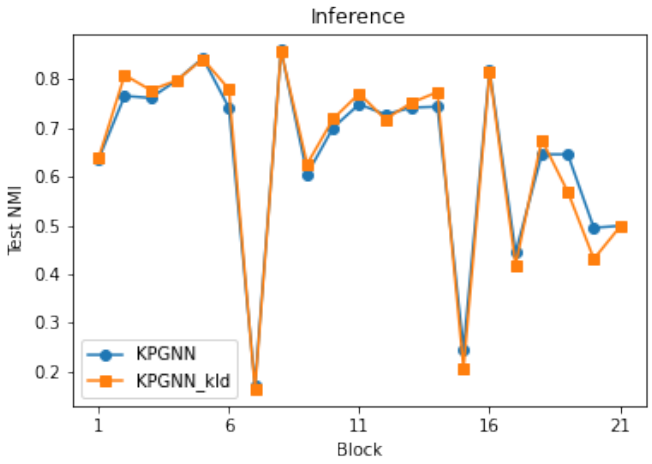

Figure 5: KPGNN vs KPGNN with kld loss at inference stage.

Table 2: Test NMI for the three methods.

Blocks	M_1	M_2	M_3	M_4	M_5	M_6	M_7
KPGNN	0.73	0.82	0.8	**0.88**	0.83	0.81	0.16
KPGNN+GODE	0.66	**0.9**	**0.87**	0.84	0.85	0.84	**0.23**
KPGNN+GODE+KLD	**0.8**	0.86	0.84	0.84	**0.85**	**0.85**	0.18
Blocks	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}
KPGNN	0.85	0.61	0.78	0.8	0.74	0.816	**0.79**
KPGNN+GODE	0.89	0.65	**0.81**	**0.84**	0.76	**0.85**	0.77
KPGNN+GODE+KLD	**0.9**	**0.67**	0.8	0.8	**0.84**	0.84	0.75
Blocks	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}	M_{21}
KPGNN	0.24	0.826	0.38	0.8	0.72	**0.65**	0.53
KPGNN+GODE	0.28	0.827	0.46	0.7	0.71	0.6	0.58
KPGNN+GODE+KLD	**0.34**	**0.833**	**0.48**	**0.81**	**0.73**	0.58	**0.61**

tial for various applications, including crisis management, product recommendation, and decision-making, and is made more challenging by the complexity of social streams. Our models outperform KPGNN on most of the blocks, with KPGNN combined with GODE and KL-divergence loss performing the best. Our findings suggest that incorporating GODE and KL-divergence loss during training improves the performance of KPGNN, and we also highlight the importance of tuning hyperparameters such as hidden dimension and window size to achieve optimal performance. Overall, our work provides valuable insights into the detection of social events in social streams and can aid in better understanding group social behaviors and public concerns.

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.

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.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. Advances in neural information processing systems, 31, 2018.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. Journal of computational and applied mathematics, 6(1):19–26, 1980.

Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. Normalized mutual information feature selection. IEEE Transactions on neural networks, 20(2):189–201, 2009.

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

Bang Liu, Fred X Han, Di Niu, Linglong Kong, Kunfeng Lai, and Yu Xu. Story forest: Extracting events and telling stories from breaking news. ACM Transactions on Knowledge Discovery from Data (TKDD), 14(3):1–28, 2020.

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.

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.