Knowledge-Preserving Incremental Social Event Detection via Heterogeneous GNNs

Yuwei Cao
University of Illinois at Chicago
Chicago, IL, USA
ycao43@uic.edu

Hao Peng*
Beihang University
Beijing, China
penghao@act.buaa.edu.cn

Jia Wu
Macquarie University
Sydney, Australia
jia.wu@mq.edu.au

Yingtong Dou
University of Illinois at Chicago
Chicago, IL, USA
ydou5@uic.edu

Jianxin Li
Beihang University
Beijing, China
lijx@act.buaa.edu.cn

Philip S. Yu
University of Illinois at Chicago
Chicago, IL, USA
psyu@uic.edu

ABSTRACT

Social events provide valuable insights into group social behaviors and public concerns and therefore have many applications in fields such as product recommendation and crisis management. The complexity and streaming nature of social messages make it appealing to address social event detection in an incremental learning setting, where acquiring, preserving, and extending knowledge are major concerns. Most existing methods, including those based on incremental clustering and community detection, learn limited amounts of knowledge as they ignore the rich semantics and structural information contained in the social data. Moreover, they cannot memorize previously acquired knowledge. In this paper, we propose a novel Knowledge-Preserving Incremental Heterogeneous Graph Neural Network (KPGNN) for incremental social event detection. To acquire more knowledge, KPGNN models complex social messages into unified social graphs to facilitate data utilization and explores the expressive power of GNNs for knowledge extraction. To continuously adapt to the incoming data, KPGNN adopts contrastive loss terms that cope with a changing number of event classes. It also leverages the inductive learning ability of GNNs to efficiently detect events and extends its knowledge from the previously unseen data. To deal with large social streams, KPGNN adopts a mini-batch subgraph sampling strategy for scalable training, and periodically removes obsolete data to maintain a dynamic embedding space. KPGNN requires no feature engineering and has few hyperparameters to tune. Extensive experimental results demonstrate the superiority of KPGNN over various baselines.

KEYWORDS

Social Event Detection, Graph Neural Networks, Incremental Learning, Contrastive Learning

1 INTRODUCTION

Social events (e.g., Twitter discussions on the Notre-Dame Cathedral fire, as shown in Figure 1) highlight significant happenings in our daily life, and generally reflect group social behaviors and widespread public concerns. Social event detection is very important since it provides valuable insights for us to make timely responses, and therefore has many applications in fields including crisis management, product recommendation, and decision making [20, 24, 25, 48]. In the last decade, social event detection has become the research hot spot in social media mining and has drawn more and more attention from both academia and the industry [8, 29].

The task of social event detection can be formalized as extracting clusters of co-related messages from social streams (i.e., sequences of social media messages) to represent events (the corresponding methods are categorized as document-pivot, i.e., DP methods [1, 16, 21, 29, 47, 48], and are discussed in more detail in Section 5). Compared to traditional news and articles, social streams such as Twitter streams are more complex, for the following reasons: they are generated in sequential order and are enormous in volume; they contain elements of various types including text, time, hashtags, and implicit social network structure; their contents are short and often contain abbreviations that are not in the dictionary; the semantics of their elements change rapidly. All these characteristics made social event detection a challenging task [32].

We argue that the complexity and streaming nature of social messages make it appealing to address the task of social event detection in an incremental learning [4, 11] setting. Incremental learning models are characterized by their abilities of 1) acquiring knowledge from data, 2) preserving previously learned knowledge, and 3) continually adapting to the incoming data [4]. Existing social event detection methods, however, cannot fully satisfy these requirements of incremental learning. Traditional methods based on incremental clustering [1, 16, 28, 47] and community detection [8, 21, 23, 44], though are capable of detecting events in an online manner, learn limited amounts of knowledge from social data. Specifically, they identify events using statistical features such as word frequencies and co-occurrences while ignoring the rich semantics and structural information contained in social streams to some extent. Moreover, these methods have very few parameters in their models. Consequently, they cannot memorize previously learned information, i.e., they forget what they have learned. Motivated by graph neural networks (GNNs)' power in aggregating structural information and semantics, recent efforts such as [29] explore GNN-based social event detection and show promising performances. Nevertheless, [29] assumes that the entire dataset is available and the output space is fixed. Extending to new data points requires retraining its model from scratch. In a word, the task of incremental social event detection is not yet solved.

In this paper, we address incremental social event detection from a knowledge-preserving perspective, i.e., we design our model to

*This is the corresponding author

continuously extend its knowledge while detecting events from the incoming social messages. Nevertheless, such knowledge-preserving incremental social event detection poses significant challenges, which we summarize as follows. Firstly, as mentioned above, the model should acquire, preserve, and extend knowledge. This requires the model to efficiently organize and process various elements in the social streams for full utilization and effectively interpret these elements to discover underlying knowledge that would help event detection. Moreover, the model needs to efficiently update its knowledge accordingly when new messages arrive. Given this, continuous training using the new messages is preferred over retraining from scratch. Secondly, the model needs to handle a changing number of events (classes) that are unknown. Unlike in the offline scenario where the total number of classes is pre-known and fixed, new events arise continuously in the online scenario. Apparently, classification techniques using softmax cross-entropy losses cannot be directly applied. Besides, predefining the total number of events as a hyperparameter is commonly done [1, 48] but undesirable as it introduces additional constraints. Thirdly, the model needs to scale to large social streams. As new messages arrive, the model needs to get rid of the obsolete social messages now and then to maintain a dynamic output space. Also, mini-batch training [31] is preferable compared to batch training [31], as it does not require having the entire training dataset in memory.

To tackle the above challenges, we propose a novel knowledge-preserving incremental social event detection model based on heterogeneous GNNs. We name our model as Knowledge-Preserving Heterogeneous Graph Neural Network or KPGNN in short. KPGNN employs a document-pivot technique and classifies social messages based on their correlations. 1) To address the first challenge, i.e., to acquire, preserve, and extend knowledge, we leverage Heterogeneous Information Networks (HINs) [34] to organize social stream elements of various types into unified social graphs. We then harness the expressive power of GNNs to acquire knowledge from the semantic and structural information contained in the social graphs. The GNN parameters, tuned for the social event detection purpose, preserve the model's knowledge about the nature of social data. As new messages arrive, the social graphs are subject to changes. To cope with that, we design a life-cycle of KPGNN (shown in Figure 2) to contain a detection stage that directly detects events from the previously unseen messages and a maintenance stage that extends the model's knowledge by resuming the training process using the new data. Such inference-maintenance design leverages GNNs' inductive learning ability, which, as pointed out by [10], is theoretically discussed [13, 36] yet less explored in real-world applications. 2) To tackle the second challenge, i.e., dynamic event classes, we introduce contrastive learning techniques into the training process. Instead of using cross-entropy loss, we design a triplet loss that contrasts positive message pairs with the corresponding negative ones. The triplets are constructed in an online manner, as inspired by computer vision studies [14, 33], to facilitate incremental learning. We also introduce an additional global-local pair loss term to better incorporate the graph structure. This term is based on contrasting global-local structural information [2, 15, 37] and does not require class labels. 3) To address the third challenge, i.e., scale to large social graphs, we periodically remove obsolete messages from the social graphs to keep an up-to-date embedding space. We also adopt a mini-batch subgraph sampling algorithm [13] for scalable and efficient training.

Table 1: Glossary of Notations.

Notation	Description
$S; M$	Social stream; Message block
m	Message or message as a node type
$e; E$	Event; Set of events
w	The window size for maintaining the model
$o; e; u$	Word; Named entity; User (node types)
$\boldsymbol{W}_{mk}$	The adjacency matrix between node type m and k
$\mathcal{G}$	Message graph
N	The total number of messages in $\mathcal{G}$
$\boldsymbol{A}$	The adjacency matrix of $\mathcal{G}$
$\boldsymbol{X}$	The initial feature vectors of the messages in $\mathcal{G}$
$\mathcal{E}(\boldsymbol{X}, \boldsymbol{A})$	GNN that embeds the messages in $\mathcal{G}$
$l; L$	GNN layer number; Total number of layers
$b; B$	Mini-batch number; Total number of mini-batches
$\{m_b\}$	A set of messages in the b-th mini-batch
$c_1, ..., c_L$	The number of neighbors sampled in each layer
$\boldsymbol{h}_{m_i}^{(l)}$	The representation of m_i at the l-th layer
$\boldsymbol{h}_{m_i}$	The final representation of m_i
m_i+	A message in the same class as m_i
m_i-	A message that is not in the same class as m_i
$\boldsymbol{s}$	The summary vector of $\mathcal{G}$
$\tilde{\boldsymbol{h}}_{m_i}$	The corrupted representation of m_i
$\mathcal{L}_t$	Triplet loss
$\mathcal{L}_p$	Global-local pair loss

We conduct extensive experiments on a large-scale Twitter corpus [26] that is publicly available. The empirical results show that KPGNN achieves better performances compared to various baselines by effectively preserving event-detection oriented knowledge. We make our code and preprocessed data publicly available [1].

We summarize our main contributions as follows:

- We formalize the task of social event detection in an incremental learning setting.
- We design a novel heterogeneous GNN-based knowledge-preserving incremental social event detection model, namely KPGNN. KPGNN continuously detects events from the incoming social streams while possessing the power of interpreting complex social data to accumulate knowledge. To the best of our knowledge, we are the first to use GNNs in incremental social event detection.
- We empirically demonstrate the effectiveness of the proposed KPGNN model.

2 NOTATIONS AND PROBLEM FORMULATION

We first summarize the main notations used in this paper in Table 1. Then we formalize *Social Stream*, *Social Event*, *Social Event Detection*, and *Incremental Social Event Detection* as follows.

[1] https://github.com/RingBDStack/KPGNN

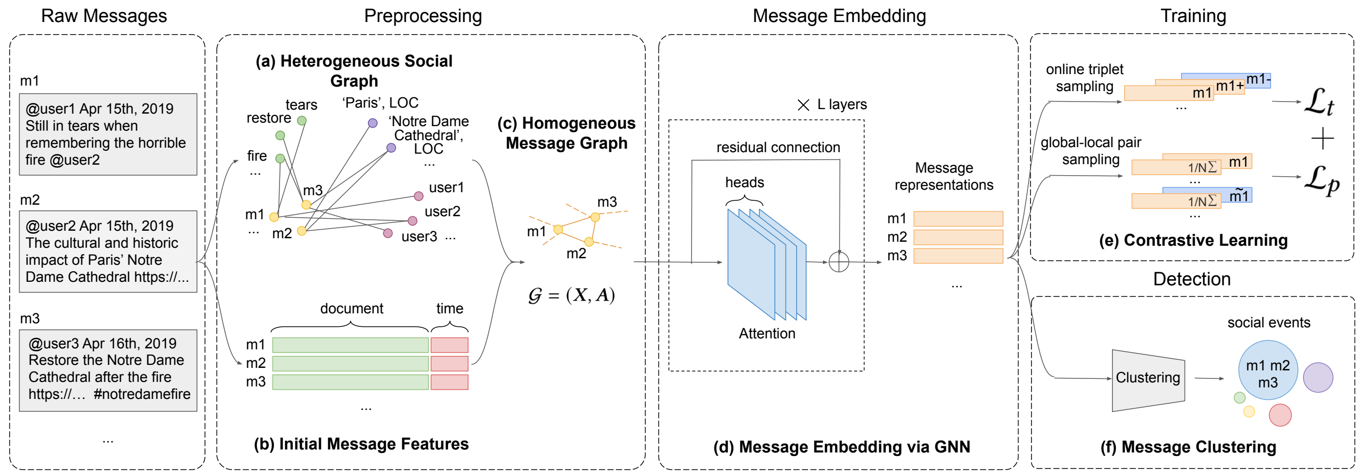

Figure 1: The architecture of the proposed KPGNN model (*best viewed in color*). (a) is a heterogeneous social graph that combines various types of elements contained in the raw messages. Different node colors denote different node types. **(b)** is the initial feature vectors of the messages. **(c)** is a homogeneous message graph that incorporates **(a)** and **(b)** (detailed in Section 3.2). **(d)** shows a GNN-based encoder that learns representations for the messages in **(c)**. **(e)** calculates triplet loss $\mathcal{L}_t$ and global-local pair loss $\mathcal{L}_p$ through contrastive learning. In **(e)**, two orange bars form a positive pair while one orange bar and one blue bar denote a negative pair. **(f)** clusters messages into social events.

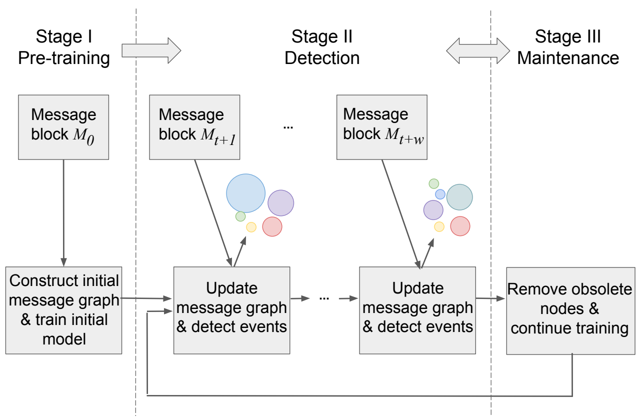

Figure 2: Incremental detection life-cycle of the proposed KPGNN model. Stage I pre-trains an initial KPGNN model. In **Stage II**, the pre-trained KPGNN model is directly used to detect social events from unseen messages. In **Stage III**, the KPGNN model is maintained by continuing training using the new messages arrived in **Stage II**. The maintained KPGNN model can then be used for the next detection stage. M_0, M_{t+1}, M_{t+w} denote the input message blocks and w is the window size for maintaining the model. The colored bubbles stand for clusters of messages, i.e., social events.

Definition 2.1. *A* ***social stream*** $S = M_0, ..., M_{i-1}, M_i, ...$ *is a continuous and temporal sequence of blocks of social messages, where* M_i *is a message block that contains all the messages arrive during time period* $[t_i, t_{i+1})$. *We denote a message block* M_i *as* $M_i = \{m_j | 1 \leq j \leq |M_i|\}$, *where* $|M_i|$ *is the total number of messages contained in* M_i, *and* m_j *is one message. We denote a social message* m_j *as* $m_j = \{d_j, u_j, t_j\}$, *where* d_j, u_j, *and* t_j *stand for the associated text document, users (sender and mentioned users), and timestamp of* m_j, *respectively.*

Definition 2.2. *A* ***social event*** $e = \{m_i | 1 \leq i \leq |e|\}$ *is a set of correlated social messages that discuss the same real-world happening. Note that we assume each social message belongs to at most one event.*

Definition 2.3. *Given a message block* M_i, *a* ***social event detection*** *algorithm learns a model* $f(M_i; \theta) = E_i$, *such that* $E_i = \{e_k | 1 \leq k \leq |E_i|\}$ *is a set of events contained in* M_i. *Here,* θ *denotes the parameter of* f.

Definition 2.4. *Given a social stream* S, *an* ***incremental social event detection*** *algorithm learns a sequence of event detection models* $f_0, ..., f_{t-w}, f_t, ...,$ *such that* $f_t(M_i; \theta_t, \theta_{t-w}) = E_i$ *for all message blocks in* $\{M_i | t+1 \leq i \leq t+w\}$. *Here,* $E_i = \{e_k | 1 \leq k \leq |E_i|\}$ *is a set of events contained in message block* M_i, w *is the window size for updating the model,* θ_t *and* θ_{t-w} *are the parameters of* f_t *and* f_{t-w}, *respectively. Note that* f_t *extends the knowledge of its predecessor* f_{t-w} *by depending on* θ_{t-w}. *Specially, we call* f_0 *which extends no previous model as the initial model.*

3 METHODOLOGY

In this section, we introduce our proposed KPGNN model. Specifically, Section 3.1 introduces the life-cycle of KPGNN to give the big picture of how KPGNN operates incrementally. Sections 3.2-3.5 zoom into the components of KPGNN, which are designed with the aims of incremental knowledge acquiring and preserving. Finally, Section 3.6 analyzes the time complexity of KPGNN.

3.1 Continuous Detection Framework

KPGNN follows Definition 2.4 and operates incrementally. Figure 2 and Algorithm 1 depict the working process of KPGNN. As shown in Figure 2, the life-cycle of KPGNN contains three stages, i.e., pre-training, detection, and maintenance. In the pre-training stage, we construct an initial message graph (detailed in Section 3.2) and train an initial model (Sections 3.3 and 3.4). In the detection stage, we

Algorithm 1: KPGNN: Knowledge-Preserving Heterogeneous Graph Neural Network

Input: A social stream $S = M_0, M_1, M_2, ...$, available labels*, window size w, the number of layers L, and the number of mini-batches B.
Output: Sets of events: $E_0, E_1, E_2, ...$.

for $t = 0, 1, 2, ...$ **do**
  if $t == 0$ **then**
    $\mathcal{G} \leftarrow$ construct initial message graph (Section 3.2)
  else
    $\mathcal{G} \leftarrow$ update M_t into message graph (Section 3.2)
  if $t! = 0$ **then** `// Detect events from` M_t
    for $l = 1, 2, ..., L$ **do**
      $\boldsymbol{h}^{(l)}_{m_i} \leftarrow$ Eq. (2), $\forall m_i \in M_t$
    $\boldsymbol{h}_{m_i} \leftarrow \boldsymbol{h}^{(L)}_{m_i}, \forall m_i \in M_t$
    $E_t \leftarrow$ message clustering (Section 3.5)
  if $t\%w == 0$ **then** `// Pre-train or maintain model`
    if $t! = 0$ **then**
      $\mathcal{G} \leftarrow$ remove obsolete messages (Section 3.2)
    for $b = 1, 2, ..., B$ **do** `// Train in mini-batches`
      $\{m_b\} \leftarrow$ sample a mini-batch of messages from $\mathcal{G}$ (Section 3.4)
      for $l = 1, 2, ..., L$ **do**
        $\boldsymbol{h}^{(l)}_{m_i} \leftarrow$ Eq. (2), $\forall m_i \in \{m_b\}$
      $\boldsymbol{h}_{m_i} \leftarrow \boldsymbol{h}^{(L)}_{m_i}, \forall m_i \in \{m_b\}$
      $T \leftarrow$ triplet sampling $\forall m_i \in \{m_b\}$(Section 3.4)
      $\mathcal{L}_t \leftarrow$ Eq. (3), $\forall m_i \in \{m_b\}$
      $\boldsymbol{s}, \tilde{\boldsymbol{h}}_{m_i} \leftarrow$ calculate summary and corrupted representations $\forall m_i \in \{m_b\}$(Section 3.4)
      $\mathcal{L}_p \leftarrow$ Eq. (4), $\forall m_i \in \{m_b\}$
      Back-propagation to update parameters;

*Labels are used for pre-training and maintenance; full labeling is not required (see Section 3.4 for details).

update the message graph with the input message block (Section 3.2) and detect events (Section 3.5). The current KPGNN model works on a continuous series of blocks before entering the maintenance stage. In the maintenance stage, we remove obsolete messages from the message graph (Section 3.2) and resume model training using data that arrived within the last window. The maintenance stage allows the model to forget obsolete knowledge (we experiment on different forgetting strategies in Section 4.4) and equips the model with the latest knowledge. The maintained model can then be used for detection in the next window. In this manner, KPGNN continuously adapts to the incoming data to detect new events and update the model's knowledge.

3.2 Heterogeneous Social Message Modeling

During preprocessing, we aim to 1) fully leverage the social data by extracting different types of informative elements from the messages, and 2) organize the extracted elements in a unified manner to facilitate further processing. We leverage heterogeneous information networks (HINs) [34] for these purposes. A HIN is a graph that contains more than one type of nodes and edges. Figure 1 (a) shows an example of HIN. Given a message m_i, we extract a set of named entities [2] and words (with very common and very rare words filtered out) from its document. The extracted elements, together with a set of users associated with m_i and m_i itself, are added as nodes into a HIN. We add edges between m_i and its elements. For example, in Figure 1 (a), from m_1, we can extract tweet node "m_1", word nodes including "fire" and "tears" (for simplicity, the figure only shows two words, while there are more to be extracted), and user nodes including "user1" and "user2". We add edges between "m_1" and the other nodes. We repeat the same process for all the messages, with repetitive nodes merged. Eventually, we get a heterogeneous social graph containing all the messages and their elements of different types. We denote the node types, i.e., message, word, named entity, and user as m, o, e, and u, respectively.

Existing heterogeneous GNNs [17, 39, 41, 45, 46] typically retain heterogeneous node types throughout their models as they care to learn the representations for all the nodes. However, KPGNN, as a document-pivot model, focuses on learning the correlations between messages and therefore we adopt a different design and map the heterogeneous social graph into a homogeneous message graph as shown in Figure 1 (c). The homogeneous message graph only contains message nodes and there are edges between messages that share some common elements. Through mapping, the homogeneous message graph preserves the message correlations encoded by the heterogeneous social graph. Specifically, the mapping process follows:

$$\boldsymbol{A}_{i,j} = min\Big\{\big[\sum_k \boldsymbol{W}_{mk} \cdot \boldsymbol{W}^\top_{mk}\big]_{i,j}, 1\Big\}, k \in \{o, e, u\}. \tag{1}$$

Here, $\boldsymbol{A} \in \{0, 1\}^{N \times N}$ stands for the adjacency matrix of the homogeneous message graph, where N is the total number of messages in the graph. $\cdot_{i,j}$ stands for the matrix element at the i-th row and the j-th column, k denotes a node type. $\boldsymbol{W}_{mk}$ stands for a submatrix of the adjacency matrix of the heterogeneous social graph that contains rows of type m and columns of type k. $\cdot^\top$ stands for matrix transpose, and $min\{,\}$ takes the smaller between its two operands. If messages m_i and m_j link to some common type k nodes, $\big[\boldsymbol{W}_{mk} \cdot \boldsymbol{W}^\top_{mk}\big]_{i,j}$ will be greater than or equal to one, and $\boldsymbol{A}_{i,j}$ will be equal to one.

To leverage the natural language semantics and temporal information in the data, we construct feature vectors of the messages, as shown in Figure 1 (b). Specifically, document features are calculated as an average of the pre-trained word embeddings [27] of all the words in the documents. Temporal features are calculated by encoding the timestamps: we convert each timestamp to OLE date (a floating-point number representing the number of days after December 30, 1899), whose fractional and integral components form a 2-d vector. We then perform a message-wise concatenation of the two. The resulting initial feature vectors, denoted as $\boldsymbol{X} = \{\boldsymbol{x}_{m_i} \in \mathbb{R}^d | 1 \leq i \leq N\}\}$, where $\boldsymbol{x}_{m_i}$ is the initial feature vector of m_i and d is the dimension, are associated with the corresponding

[2] https://spacy.io/api/annotation#section-named-entities

message nodes. We denote the homogeneous message graph as $\mathcal{G} = (\boldsymbol{X}, \boldsymbol{A})$.

Note that $\mathcal{G}$ is not static. When a new message block arrives for detection (shown in Figure 2 stage II), we update $\mathcal{G}$ by inserting the new message nodes, their linkages with the existing message nodes, and the linkages within themselves into $\mathcal{G}$. Similarly, we periodically remove obsolete message nodes and edges associated with them from $\mathcal{G}$ (shown in Figure 2 stage III). We empirically compare different update-maintenance strategies in Section 4.4.

3.3 Knowledge-Preserving Incremental Message Embedding

To study the correlations between messages in a knowledge-preserving manner, we leverage GNNs to learn message representations. Specifically, we train a GNN encoder $\mathcal{E} : \mathbb{R}^{N\times d} \times \{0,1\}^{N\times N} \rightarrow \mathbb{R}^{N\times d'}$, such that $\mathcal{E}(\boldsymbol{X}, \boldsymbol{A}) = \{\boldsymbol{h}_{m_i} \in \mathbb{R}^{d'} | 1 \le i \le N\}$, where $\boldsymbol{h}_{m_i}$ represents the high-level representation of message m_i. Figure 1 (d) illustrates this process. $\mathcal{E}$ contains L layers and the layer-wise propagation follows:

$$\boldsymbol{h}_{m_i}^{(l)} \leftarrow \overset{heads}{\|} \Big(\boldsymbol{h}_{m_i}^{(l-1)} \oplus \underset{\forall m_j \in \mathcal{N}(m_i)}{Aggregator}\big(Extractor(\boldsymbol{h}_{m_j}^{(l-1)})\big)\Big). \tag{2}$$

Here, $\boldsymbol{h}_{m_i}^{(l)}$ is the representation of m_i at the (l)-th GNN layer, and $\boldsymbol{h}_{m_i}^{(0)} = \boldsymbol{x}_{m_i}$. $\mathcal{N}(m_i)$ denotes a set of neighbors of m_i according to $\boldsymbol{A}$. $\oplus$ stands for an aggregation, e.g., summation, of the information contained in its two operands. $\overset{heads}{\|}$ represents head-wise concatenation [35]. $Extractor(\cdot)$ and $Aggregator(\cdot)$ [13] are designed differently in different GNNs. The former extracts useful information from the neighboring messages' representations while the latter summarizes the neighborhood information. We use $\boldsymbol{h}_{m_i}^{(L)}$ as the final representation of m_i, i.e., $\boldsymbol{h}_{m_i}$.

In order for KPGNN to work incrementally and embed previously unseen messages, we adopt the graph attention mechanism [36] for neighborhood information extraction and aggregation. Our $Extractor(\cdot)$ and $Aggregator(\cdot)$ do not assume a fixed graph structure as did in [12, 18, 40], instead, they consider the similarities between the representations of the source message and its neighboring messages. In this way, KPGNN handles evolving message graphs where new message nodes continuously joining in and the model generalizes to even completely unseen message graphs.

KPGNN preserves knowledge: the learned representations encode the model's knowledge about the messages, which is a fusion of natural language semantics, temporal information, and the structural information of the homogeneous message graph; the learned parameters preserve the model's cognition about the nature of social data and are especially tuned, via contrastive training (discussed in detail in Section 3.4), for the social event detection purpose.

3.4 Scalable Training via Contrastive Learning

As new messages continuously arrive, there can be new events that are previously unseen by the model. Cross-entropy loss functions, though widely adopted by various GNNs [18, 36], are no longer applicable. We instead construct a contrastive triplet loss that enables KPGNN to differentiate the events without constraining their total number. As shown in Figure 1 (e), for each message m_i (referred to as an anchor message), we sample a positive message m_i+ (i.e., a message from the same class) and a negative message m_i- (i.e., a message from a different class) to form a triplet (m_i, m_i+, m_i-). The triplet loss function pushes positive messages close to and negative messages far away from anchors and is formalized as:

$$\mathcal{L}_t = \sum_{(m_i, m_i+, m_i-)\in T} max\big\{\mathcal{D}(\boldsymbol{h}_{m_i}, \boldsymbol{h}_{m_i+}) - \mathcal{D}(\boldsymbol{h}_{m_i}, \boldsymbol{h}_{m_i-}) + a, 0\big\}. \tag{3}$$

Here, $\mathcal{D}(,)$ computes the Euclidean distance between two vectors. $a \in \mathbb{R}$ is a hyperparameter controlling how farther away should the negative messages be compared to the positive ones. $max\{,\}$ takes the larger between its two operands. T is a set of triplets sampled in an online manner [14] and we focus on the hard triplets [14], i.e., triplets that satisfy $\mathcal{D}(\boldsymbol{h}_{m_i}, \boldsymbol{h}_{m_i-}) < \mathcal{D}(\boldsymbol{h}_{m_i}, \boldsymbol{h}_{m_i+})$ for efficient training, as the usage of hard triplets results in faster convergence while helps to learn sharper boundaries between positive and negative samples [14].

To better incorporate the structural information of the message graph, we construct an additional global-local pair loss that enables KPGNN to discover and preserve the features of similar local structures, as shown in Figure 1 (e). Specifically, the global-local pair loss function, inspired by [37], takes a noise-contrastive form. It seeks to maximize the mutual information between the local message representations and the global summary of the message graph by minimizing their binary cross-entropy:

$$\mathcal{L}_p = \frac{1}{N}\sum_{i=1}^{N}\Big(\log \mathcal{S}(\boldsymbol{h}_{m_i}, \boldsymbol{s}) + \log\big(1 - \mathcal{S}(\tilde{\boldsymbol{h}}_{m_i}, \boldsymbol{s})\big)\Big). \tag{4}$$

Here, $\boldsymbol{s} \in \mathbb{R}^{d'}$ is a summary of the message graph and we simply use the average of all message representations. $\tilde{\boldsymbol{h}}_{m_i}$ is a corrupted representation of m_i and is learned by $\mathcal{E}(\tilde{\boldsymbol{X}}, \boldsymbol{A})$, where $\tilde{\boldsymbol{X}}$ is constructed by row-wise shuffle of $\boldsymbol{X}$. $\mathcal{S}(,)$ is a bilinear scoring function that outputs the probability of its two operands coming from a joint distribution (i.e., being learned from the same graph). Note $\mathcal{L}_p$ is readily applicable to dynamic message graphs and we show in Sections 4.2 and 4.3 with experiments how $\mathcal{L}_p$ helps improve the performance. The overall loss of KPGNN is simply the summation of $\mathcal{L}_t$ and $\mathcal{L}_p$.

To make KPGNN scalable to large message graphs, we adopts mini-batch subgraph sampling [13] during training. The triplets used in $\mathcal{L}_t$ are constructed from each mini-batch. $\boldsymbol{h}_{m_i}$, $\tilde{\boldsymbol{h}}_{m_i}$ and $\boldsymbol{s}$ in $\mathcal{L}_p$ are also calculated from each subgraph.

It is important to note: 1) KPGNN, as an incremental model, is not trained once and for all. Instead, we periodically resume the training to keep the model's knowledge up-to-date, as shown in Figure 2 stage III. In the maintenance stage, the training does not start from scratch, rather, it is continued based on the previous knowledge (i.e., the existing model parameters) using the new data arrived during the last time window. 2) Although the calculation of $\mathcal{L}_t$ needs labels, KPGNN does not require full labeling, as T can be sampled from the labeled messages. The unlabeled messages also contribute, as their features and structural information could be aggregated into the representations of the labeled ones through propagation (detailed in Section 3.3). The calculation of $\mathcal{L}_p$, on the other hand, does not require any labels. Such design suits the real-world scenarios where hashtags can be used as labels and the social streams can be considered as partially labeled.

3.5 Message Clustering

At the detection stage, we cluster the messages based on the learned message representations. Distance-based clustering algorithms such as K-Means and density-based ones such as DBSCAN [6] can be readily used for clustering the representations. Among them, [6] does not require specifying the total number of classes and therefore suits the need for incremental detection. This process is shown in Figure 1 (f). KPGNN outputs the resulting message clusters as social events, following Definition 2.2.

3.6 Time Complexity of KPGNN

The overall running time of KPGNN is $O(N_e)$, where N_e is the total number of edges in the message graph. Specifically, the running time of constructing the initial message graph (Algorithm 1 line 3) or updating the message graph (Algorithm 1 line 5) is $O(N + N_e) = O(N_e)$, where N is the total number of messages in the message graph. The propagation of the GNN encoder $\mathcal{E}$ (Algorithm 1 lines 7-9 and 16-18) takes $O(Ndd' + N_e d') = O(N_e)$, where d and d' are the input and output dimensions of $\mathcal{E}$. The mini-batch subgraph sampling (Algorithm 1 line 15) takes $O(\prod_{l=1}^{L} c_l)$, where $c_1, ..., c_l, ...c_L$ are L user-specified constants that define the number of neighbors sampled from the neighborhood of one message in each layer. In practice, $\prod_{l=1}^{L} c_l \ll N_e$. Triplet sampling (Algorithm 1 line 19) takes $O(\sum_{b=1}^{B} |\{m_b\}|^2)$, where $|\{m_b\}|$ is the number of messages in the b-th batch. The corruption of the message graph (Algorithm 1 line 21) takes $O(N)$.

We can tell that maintaining a light-weighted message graph would help reduce time consumption and we compare different maintenance strategies in Section 4.4.

4 EXPERIMENTS

In this section, we first introduce the experimental setups, including the dataset, baselines, experimental setting, and the evaluation metrics. We then compare KPGNN to various baselines including offline as well as incremental social event detection models. We also investigate the effects of adopting different forgetting strategies in the maintenance stage of KPGNN's life-cycle. At last, we provide sensitivity analysis for the hyperparameters of KPGNN.

4.1 Experimental Setup

4.1.1 Dataset. We conduct our experiments on a large-scale, publicly available Twitter dataset [3] collected for DP social event detection methods' evaluations from [26]. After filtering out repeated and unfetchable tweets, the dataset contains 68,841 manually labeled tweets related to 503 event classes and spread over a period of four weeks.

4.1.2 Baselines. We compare KPGNN to general message representation learning and similarity measuring methods, offline social event detection methods, and the incremental ones. Our baselines include: **Word2vec** [27], which uses the average of the pre-trained Word2vec embeddings of all the words in a message as its representation; **Latent Dirichlet Allocation (LDA)** [3], which is a generative statistical model that learns the message documents' representations by modeling the underlying topic and word distributions; **Word Mover's Distance (WMD)** [19], which measures the dissimilarity between two message documents by calculating the minimum amount of distance that the word embeddings in one need to travel to reach that of the other; **Pairwise Popularity Graph Convolutional Network (PP-GCN)** [29], which is an offline fine-grained social event detection method based on GCN[18]; **EventX** [21] is a fine-grained event detection method based on community detection and is applicable to the online scenario; **KPGNN$_t$** is a variation of the proposed KPGNN model, in which the global-local pair loss term $\mathcal{L}_p$ is removed from the loss function and only the triplet loss term $\mathcal{L}_t$ is used.

4.1.3 Experimental Setting and Implementation. For LDA, we set the total number of topics to 50. For EventX, we adopt the hyperparameters as suggested in the original paper [21]. For GNN-based methods (PP-GCN, KPGNN, and KPGNN$_t$), we set the total number of heads to 4, embedding dimension d' to 32, the total number of layers L to 2, learning rate to 0.001, optimizer to Adam, and training epochs to 15 with a patience of 5 for early stopping. For KPGNN, and KPGNN$_t$, we set the maintenance window size w to 3, mini-batch size $|\{m_b\}|$ to 2000, triplet margin a to 3, and the number of neighbors sampled for each message in the first layer c_1 and that of the second layer c_2 to 800. We observe the effects of changing w and $|\{m_b\}|$ in Section 4.5. In incremental evaluation (Section 4.3), we adopt the *latest message strategy* (different update-maintenance strategies are detailed and studied in Section 4.4). We repeat all experiments for 5 times and report the mean and standard variance of the results. Note that although KPGNN does not require pre-defining the total number of event classes, some baselines (Word2vec, LDA, and WMD) do. For a fair comparison, after we obtain the message similarity matrix from WMD and message representations from the other models except EventX (EventX does not pre-define its total number of detected classes), we leverage Spectral [43] and K-Means clustering, respectively, and set the total number of classes to the number of ground-truth classes. Otherwise, DBSCAN [6] can be used if the total number of classes is unknown as often in the case of incremental detection.

For Word2vec, we use the pre-trained 300-d GloVe[30] vectors [4]. For LDA and WMD, we use the open-source implementations [5,6]. We implement EventX with Python 3.7.3 and PP-GCN, KPGNN, and KPGNN$_t$ with Pytorch 1.6.0. All experiments are conducted on a 64 core Intel Xeon CPU E5-2680 v4@2.40GHz with 512GB RAM and 1×NVIDIA Tesla P100-PICE GPU.

4.1.4 Evaluation Metrics. To evaluate the performances of all models, we measure the similarities between their detected message clusters and the ground-truth clusters. We utilize Normalized Mutual Information (NMI) [7], Adjusted Mutual Information (AMI) [38], and Adjusted Rand Index (ARI) [38]. NMI measures the amount of information one can extract from the distribution of the predictions

[3] http://mir.dcs.gla.ac.uk/resources/

[4] https://spacy.io/models/en-starters#en_vectors_web_lg

[5] https://radimrehurek.com/gensim/models/ldamodel.html

[6] https://tedboy.github.io/nlps/generated/generated/gensim.similarities.WmdSimilarity.html

Table 2: Offline evaluation results. The best results are marked in bold and second-best in italic.

Metrics	Word2vec [27]	LDA [3]	WMD [19]	PP-GCN [29]	EventX [21]	KPGNN$_t$	KPGNN
NMI	.44±.00	.29±.00	.65±.00	.68±.02	**.72±.00**	.69±.01	*.70±.01*
AMI	.13±.00	.04±.00	.50±.00	.50±.02	.19±.00	*.51±.00*	**.52±.01**
ARI	.02±.00	.01±.00	.06±.00	.20±.01	.05±.00	*.21±.01*	**.22±.01**

Table 3: The statistics of the 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 messages	20, 254	8, 722	1, 491	1, 835	2, 010	1, 834	1, 276	5, 278	1, 560	1, 363	1, 096
Blocks	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}	M_{21}
# of messages	1, 232	3, 237	1, 972	2, 956	2, 549	910	2, 676	1, 887	1, 399	893	2, 410

Table 4: Incremental evaluation NMIs. The best results are marked in bold and second-best in italic.

Blocks	M_1	M_2	M_3	M_4	M_5	M_6	M_7
Word2vec [27]	.19±.00	.50±.00	.39±.00	.34±.00	.41±.00	.53±.00	.25±.00
LDA [3]	.11±.00	.27±.01	.28±.00	.25±.00	.26±.00	.32±.00	.18±.01
WMD [19]	.32±.00	.71±.00	.67±.00	.50±.00	.61±.00	.61±.00	.46±.00
PP-GCN [29]	.23±.00	.57±.02	.55±.01	.46±.01	.48±.01	.57±.01	.37±.00
EventX [21]	.36±.00	.68±.00	.63±.00	.63±.00	.59±.00	.70±.00	.51±.00
KPGNN$_t$	*.38±.01*	*.78±.01*	**.77±.00**	**.68±.01**	**.73±.01**	*.81±.00*	*.54±.01*
KPGNN	**.39±.00**	**.79±.01**	*.76±.00*	*.67±.00*	**.73±.01**	**.82±.01**	**.55±.01**
Blocks	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}
Word2vec [27]	.46±.00	.35±.00	.51±.00	.37±.00	.30±.00	.37±.00	.36±.00
LDA [3]	.37±.01	.34±.00	.44±.01	.33±.01	.22±.01	.27±.00	.21±.00
WMD [19]	.67±.00	.55±.00	.61±.00	.50±	.00.60±.00	.54±.00	.66±.00
PP-GCN [29]	.55±.02	.51±.02	.55±.02	.50±.01	.45±.01	.47±.01	.44±.01
EventX [21]	.71±.00	.67±.00	.68±.00	.65±.00	.61±.00	.58±.00	.57±.00
KPGNN$_t$	*.79±.01*	*.74±.01*	*.79±.01*	*.73±.00*	**.69±.01**	*.68±.01*	*.68±.01*
KPGNN	**.80±.00**	**.74±.02**	**.80±.01**	**.74±.01**	*.68±.01*	**.69±.01**	**.69±.00**
Blocks	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}	M_{21}
Word2vec [27]	.27±.00	.49±.00	.33±.00	.29±.00	.37±.00	.38±.00	.31±.00
LDA [3]	.21±.00	.35±.01	.19±.00	.18±.00	.29±.01	.35±.00	.19±.00
WMD [19]	.51±.00	.60±.00	.55±.00	.63±.00	.54±.00	.58±.00	.58±.00
PP-GCN [29]	.39±.01	.55±.01	.48±.00	.47±.01	.51±.02	.51±.01	.41±.02
EventX [21]	.49±.00	.62±.00	.58±.00	.59±.00	.60±.00	.67±.00	.53±.00
KPGNN$_t$	*.57±.01*	*.78±.01*	*.69±.01*	*.68±.01*	*.73±.00*	**.73±.00**	*.59±.01*
KPGNN	**.58±.00**	**.79±.01**	**.70±.01**	**.68±.02**	**.73±.01**	*.72±.02*	**.60±.00**

regarding the distribution of the ground-truth labels and is broadly adopted in social event detection method evaluations [21, 29]. AMI, similar to NMI, also measures the mutual information between two clusterings but is adjusted to account for chance [38]. ARI considers all prediction-label pairs and counts pairs that are assigned in the same or different clusters, and ARI also accounts for chance [38].

4.2 Offline Evaluation

This subsection compares KPGNN to the baselines in an offline scenario. We randomly sample 20% of the dataset for testing, 10% for validation, and use the rest 70% for training.

We summarize the results in Table 2. KPGNN outperforms general message embedding and similarity measuring methods by large margins in all metrics (8−141% in NMI, 4−1200% in AMI, and 267−2100% in ARI). This is due to the fact that these methods rely either on measuring the distributions of messages' elements (LDA) or on message embeddings (Word2vec and WMD), and they all ignore the underlying graph structure of the social data to some extent. Different from these methods, KPGNN simultaneously leverages the semantics and structural information in the social messages and therefore acquires more knowledge. KPGNN also outperforms both PP-GCN and KPGNN$_t$. This implies that introducing the global-local pair loss term $\mathcal{L}_p$ helps the model learn more knowledge from the graph structure. Note that although PP-GCN shows strong performance, it assumes a stationary graph structure and cannot adapt to dynamic social streams. The proposed KPGNN, on the contrary,

Table 5: Incremental evaluation AMIs. The best results are marked in bold and second-best in italic.

Blocks	M_1	M_2	M_3	M_4	M_5	M_6	M_7
Word2vec [27]	.08±.00	.41±.00	.31±.00	.24±.00	.33±.00	.40±.00	.13±.00
LDA [3]	.08±.00	.20±.01	.22±.01	.17±.00	.21±.00	.20±.00	.12±.01
WMD [19]	.30±.00	.69±.00	.63±.00	.45±.00	.57±.00	.57±.00	.46±.00
PP-GCN [29]	.21±.00	.55±.02	.52±.01	.42±.01	.46±.01	.52±.02	.34±.00
EventX [21]	.06±.00	.29±.00	.18±.00	.19±.00	.14±.00	.27±.00	.13±.00
KPGNN$_t$	*.36±.01*	*.77±.01*	**.75±.00**	**.65±.01**	**.71±.01**	*.78±.00*	*.50±.01*
KPGNN	**.37±.00**	**.78±.01**	*.74±.00*	*.64±.01*	**.71±.01**	**.79±.01**	**.51±.01**

Blocks	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}
Word2vec [27]	.33±.00	.24±.00	.39±.00	.26±.00	.23±.00	.23±.00	.26±.00
LDA [3]	24±.01	.24±.00	.36±.01	.25±.01	.16±.01	.19±.00	.15±.00
WMD [19]	.63±.00	.46±.00	.57±.00	.42±.00	.58±.00	.50±.00	.64±.00
PP-GCN [29]	.49±.02	.46±.02	.51±.02	.46±.01	.42±.01	.43±.01	.41±.01
EventX [21]	.21±.00	.19±.00	.24±.00	.24±.00	.16±.00	.16±.00	.14±.00
KPGNN$_t$	*.75±.01*	*.70±.01*	*.76±.01*	*.70±.00*	**.66±.01**	*.65±.01*	**.65±.01**
KPGNN	**.76±.01**	**.71±.02**	**.78±.01**	**.71±.01**	**.66±.01**	**.67±.01**	*.65±.00*

Blocks	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}	M_{21}
Word2vec [27]	.15±.00	.36±.00	.24±.00	.21±.00	.28±.00	.24±.00	.21±.00
LDA [3]	.13±.00	.27±.01	.13±.00	.12±.00	.22±.01	.23±.00	.13±.00
WMD [19]	.47±.00	.59±.00	.57±.00	.60±.00	.49±.00	.55±.00	.52±.00
PP-GCN [29]	.35±.01	.52±.01	.45±.00	.45±.01	.48±.02	.45±.02	.38±.02
EventX [21]	.07±.00	.19±.00	.18±.00	.16±.00	.16±.00	.18±.00	.10±.00
KPGNN$_t$	*.53±.01*	*.75±.01*	*.67±.01*	*.66±.01*	*.70±.00*	*.68±.00*	**.57±.01**
KPGNN	**.54±.00**	**.77±.01**	**.68±.01**	**.66±.02**	**.71±.01**	**.68±.02**	*.57±.00*

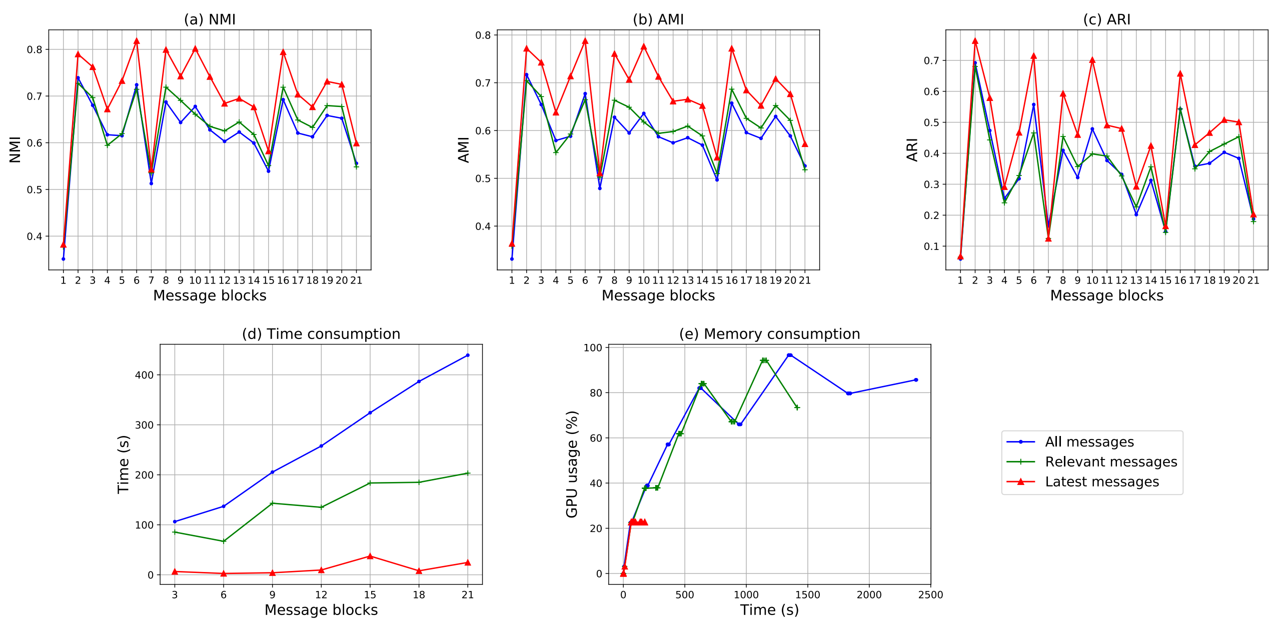

Figure 3: KPGNN with different update-maintenance strategies. (a), **(b)**, and **(c)** show the NMI, AMI, and ARI performances of KPGNN when adopting different update-maintenance strategies. In **(d)** and **(e)**, we train KPGNN for one mini-batch in the maintenance stages and measure time and memory consumption. **(d)** shows the time (in seconds) used for training KPGNN for one mini-batch. **(e)** shows the GPU% used over time through out the training.

Table 6: Incremental evaluation ARIs. The best results are marked in bold and second-best in italic.

Blocks	M_1	M_2	M_3	M_4	M_5	M_6	M_7
Word2vec [27]	01±.00	.49±.00	.16±.00	.07±.00	.17±.00	.25±.00	.02±.00
LDA [3]	.00±.00	.08±.00	.02±.01	.07±.00	.06±.00	.07±.01	.00±.00
WMD [19]	.04±.00	.48±.00	.28±.00	.11±.00	.26±.00	.16±.00	.08±.00
PP-GCN [29]	.05±.00	.67±.03	.47±.01	.24±.01	.34±.00	.55±.03	*.11±.02*
EventX [21]	.01±.00	.45±.00	.09±.00	.07±.00	.04±.00	.14±.00	.02±.00
KPGNN$_t$	*.06±.01*	*.76±.01*	**.60±.02**	**.30±.01**	**.48±.01**	*.67±.05*	.11±.01
KPGNN	**.07±.01**	**.76±.02**	*.58±.01*	*.29±.01*	*.47±.03*	**.72±.03**	**.12±.00**
Blocks	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}
Word2vec [27]	.17±.00	.08±.00	.23±.00	.09±.00	.09±.00	.06±.00	.10±.00
LDA [3]	.03±.00	.03±.01	.09±.02	.03±.01	.02±.00	.00±.00	.02±.00
WMD [19]	.22±.00	.12±.00	.20±.00	.12±.00	.27±.00	.13±.00	.33±.00
PP-GCN [29]	.43±.04	.31±.02	.50±.07	.38±.02	.34±.03	.19±.01	.29±.01
EventX [21]	.09±.00	.07±.00	.13±.00	.16±.00	.07±.00	.04±.00	.10±.00
KPGNN$_t$	*.59±.02*	*.45±.02*	*.64±.01*	*.48±.01*	**.50±.03**	*.28±.01*	**.43±.02**
KPGNN	**.60±.01**	**.46±.02**	**.70±.06**	**.49±.03**	*.48±.01*	**.29±.03**	*.42±.02*
Blocks	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}	M_{21}
Word2vec [27]	.03±.00	.19±.00	.10±.00	.07±.00	.14±.00	.10±.00	.06±.00
LDA [3]	.00±.00	.11±.01	.02±.00	.02±.00	.03±.00	.02±.01	.00±.01
WMD [19]	.16±.00	.32±.00	.26±.00	.35±.00	.12±.00	.19±.00	.19±.00
PP-GCN [29]	.15±.00	.51±.03	.35±.03	.39±.03	.41±.02	.41±.01	*.20±.03*
EventX [21]	.01±.00	.08±.00	.12±.00	.08±.00	.07±.00	.11±.00	.01±.00
KPGNN$_t$	*.16±.02*	*.62±.03*	*.41±.03*	*.46±.02*	*.50±.01*	*.51±.01*	**.23±.02**
KPGNN	**.17±.00**	**.66±.05**	**.43±.05**	**.47±.04**	**.51±.03**	**.51±.04**	.20±.01

is capable of continuously adapting to and extending its knowledge from the incoming messages (empirically testified in Section 4.3). EventX shows higher NMI but much lower AMI and ARI compared to KPGNN. This suggests that EventX tends to generate a larger number of clusters, regardless of whether there is actually more information captured, while KPGNN is stronger in general, as it scores the highest or the second-highest in all three metrics.

4.3 Incremental Evaluation

This subsection compares KPGNN with baselines in an incremental detection scenario. We split the dataset by dates to construct a social stream. Specifically, we use the messages of the first week to form an initial message block M_0 and the messages of the rest days to form the following message blocks $M_1, M_2, ..., M_{21}$. Table 3 shows the statistics of the resulting social stream. Note that PP-GCN, as an offline baseline, cannot be directly applied to the dynamic social streams and we mitigate that by retraining a new PP-GCN model from scratch for each message block: we train PP-GCN using the previous blocks as the training set and predict on the current block.

Tables 4, 5 and 6 summarize the incremental social event detection results in NMI, AMI, and ARI, respectively. The proposed KPGNN significantly and consistently outperforms the baselines for all message blocks. KPGNN achieves relative performance gains over EventX by 6−27% (16% on average) in NMI, 164−676% (319% on average) in AMI, and 68−1782% (589% on average) in ARI. The reason behind this is, EventX relies solely on community detection, while KPGNN incorporates the rich semantics of the social messages. KPGNN achieves performance gains over WMD by 3−55% (22% on average) in NMI, 3−48% (23% on average) in NMI, and 3−68% (26% on average) in ARI. This is because KPGNN leverages the structural information of the social stream, which is ignored by WMD. KPGNN also outperforms PP-GCN by 38−67% (47% on average) in NMI, 41−73% (53% on average) in AMI, and up to 58% (27% on average) in ARI. This suggests that KPGNN effectively preserves up-to-date knowledge, while PP-GCN can be distracted by obsolete information as the messages accumulate. KPGNN generally outperforms KPGNN$_t$, which testifies the positive effect of incorporating more structural information through introducing the global-local pair loss term $\mathcal{L}_p$. To conclude, KPGNN achieves performances superior to the baselines for it acquires and preserves more knowledge from the social messages.

4.4 Study on update-maintenance strategies

Recall that KPGNN updates new messages into the message graph $\mathcal{G}$ in the detection stage (Figure 2 stage II). It also periodically removes obsolete messages from $\mathcal{G}$ and continues training to adapt to the new messages in the maintenance stage (Figure 2 stage III). The manner of updating and maintaining KPGNN affects its time complexity (discussed in Section 3.6), the knowledge it preserves, and, eventually, its performance. In this subsection, we compare *three* different update-maintenance strategies, including:

1) All message strategy, keeping all the messages. In the detection stage, we simply insert the newly arrived message block

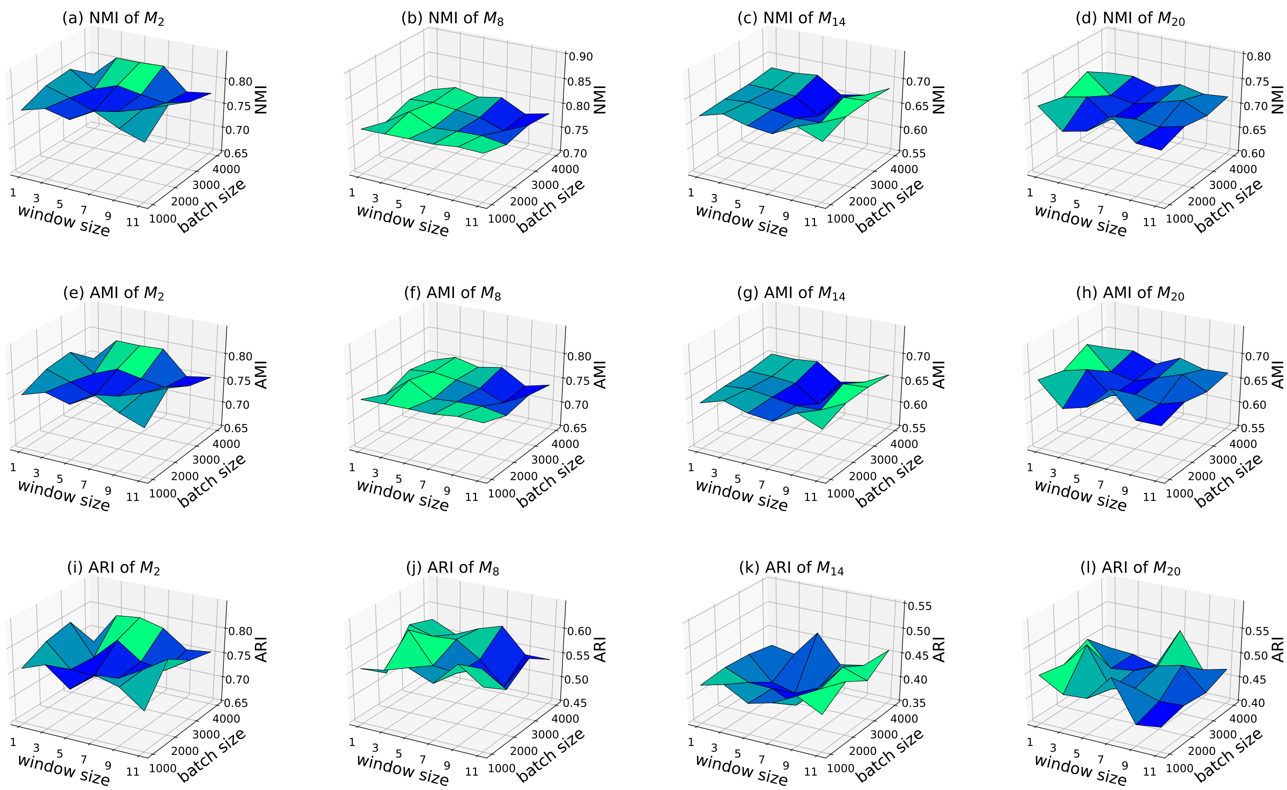

Figure 4: KPGNN with different hyperparameters. We show the performances of KPGNN on message blocks M_2, M_8, M_{14}, and M_{20} when adopting different window sizes and mini-batch sizes. **(a)-(d)** show the NMIs, **(e)-(h)** show the AMIs, and **(i)-(l)** show the ARIs. The colors indicate the fluctuations in values: the sunken areas are colored in blue and the convex areas in green.

into $\mathcal{G}$. In the maintenance stage, we continue the training process using all the messages in $\mathcal{G}$. In other words, we let KPGNN memorize all the messages it ever received. This strategy is impractical (the messages accumulated in $\mathcal{G}$ can gradually slow down the model and will eventually exceed the embedding space capacity of the message encoder $\mathcal{E}$) and we implement it just for the comparison purpose. **2) Relevant message strategy, keeping messages that are related to the newly arrived ones.** In the detection stage, we insert the newly arrived message block into $\mathcal{G}$. In the maintenance stage, we first remove messages that are not connected to any messages that arrived during the last time window and then continue training using all the messages in $\mathcal{G}$. In other words, we let KPGNN forget the messages that are both old (i.e., arrived beyond the window) and unrelated (to the new messages that arrived within the window). Note that the knowledge acquired from these messages is preserved in the form of model parameters. **3) Latest message strategy, keeping the latest message block.** In the detection stage, we use only the newly arrived message block to reconstruct $\mathcal{G}$. In the maintenance stage, we continue training with all the messages in $\mathcal{G}$, which only involves the latest message block. In other words, we let KPGNN forget all the messages except those in the latest message block. The knowledge learned from the removed messages is memorized in the form of model parameters.

Figures 3 (a)-(c) summarize the performances of KPGNN in NMI, AMI, and ARI when adopting the above three strategies in incremental social event detection experiments. We can tell that the *latest message strategy* achieves the strongest performance among all strategies by discarding all messages in the past blocks while solely keeping the knowledge learned from those messages. Figures 3 (d) and (e) show the time and memory consumption of KPGNN when adopting these strategies. As expected, the *latest message strategy* requires significantly less time and GPU memory as compared to the others, for it keeps a light-weighted message graph $\mathcal{G}$. Note that the *latest message strategy* and the *relevant message strategy* consistently while the *all message strategy* in most message blocks outperform strong baselines such as PP-GCN (the performances of PP-GCN are shown in Tables 4-6). This proves the strong performance of KPGNN despite the update-maintenance strategies.

4.5 Hyperparameter Sensitivity

This subsection studies the effects of changing w, the window size for maintaining KPGNN, and $|\{m_b\}|$, the mini-batch size. Figure 4 compares the performances of KPGNN when adopting different window sizes and mini-batch sizes. The NMI and AMI results in Figures 4 (a)-(h) have small standard deviations in the range of 0.01-0.02. This suggests that the NMIs and AMIs of KPGNN change with w and $|\{m_b\}|$, but rather insignificantly. Adopting a smaller w (1 or 3) in general gives slightly better performances. For example, the block-wise average NMIs of window sizes 1 and 3 are 0.75 and 0.75, respectively, while that of window sizes 9 and 11 are 0.74 and 0.74, respectively. The mini-batch size also has little influence on the NMIs and AMIs. For example, the block-wise average NMIs of mini-batch sizes 1000, 2000, 3000, and 4000 are 0.75, 0.75, 0.75, and 0.75, respectively. In Figures 4 (i)-(k), the ARIs of KPGNN show some fluctuations but in a manner that is not clearly related to the changes in the window size and the mini-batch size, as the block-wise average ARIs of the different window sizes range from 0.58-0.59 and that of the different mini-batch sizes range from 0.57-0.59. In a word, KPGNN is insensitive to the changes in hyperparameters.

5 RELATED WORK

Social Event Detection. Based on their objectives, social event detection methods can be categorized into document-pivot (DP) methods [1, 16, 21, 29, 47, 48] and feature-pivot (FP) ones [8, 9]. The former aim at clustering social messages based on their correlations while the latter aim at clustering social messages elements (such as words and named entities) based on their distributions. KPGNN is a DP method. Based on their application scenarios, social event detection methods can be divided into offline [29] and online [8, 16, 21, 47] ones. Though offline methods are essential in analyzing retrospective, domain-specific events such as disasters and political campaigns, online methods that continuously work on the dynamic social streams are desirable [8]. Based on techniques and mechanisms, social event detection methods can be separated into into popular classes such as methods rely on incremental clustering [1, 16, 28, 47], community detection [8, 21–23, 44] and topic models [48]. These methods, however, learn limited amounts of knowledge as they ignore the rich semantics and structural information contained in the social streams to some extent. Besides, these models have too few parameters to preserve the learned knowledge. [29] proposes a GCN-based social event detection model, however, it can only work offline. KPGNN is different from the existing methods as it effectively acquires, extends, and preserves knowledge through continuously adapting to the incoming social messages.

As a side note, our work is different from [5] since 1) [5] addresses a different task, i.e., social event prediction, 2) [5] only uses the words for graph construction, while we utilize heterogeneous element types, and 3) [5] retrains a GCN from scratch at each time step while we continuously adapt to the incoming data by resuming training periodically.

Inductive Learning with Graph Neural Networks. The past few years have witnessed the success of graph neural networks (GNNs) [13, 18, 36, 42] in graph data mining. In general, a GNN learns contextual node representations by extracting and aggregating local neighborhood information according to the input graph structure. Depending on their extraction and aggregation strategies, some GNNs [18] only conduct transductive learning [10] as they require pre-known, fixed graph structures. Others [13, 36, 42], can be used in inductive learning [10], which means that they generalize to unseen nodes. Though oftentimes discussed, inductive learning using GNNs is rarely evaluated or utilized in real application scenarios [10]. The proposed KPGNN is the first to leverage GNNs' inductive learning ability for incremental social event detection.

6 CONCLUSION

In this study, we address the task of incremental social event detection from a knowledge-preserving perspective. We design a novel KPGNN model that incorporates the rich semantics and structural information in social messages to acquire more knowledge. KPGNN continuously detects events and extends its knowledge using dynamic social streams. We empirically demonstrate the superiority of KPGNN compared to baselines through experiments. A particularly interesting future research direction would be extending the proposed model for social event analysis (including studying the evolution of events) and causal discovery in social data.

ACKNOWLEDGEMENT

The authors of this paper were supported by NSFC through grants 62002007 and U20B2053, Key Research and Development Project of Hebei Province through grant 20310101D, State Key Laboratory of Software Development Environment (SKLSDE-2020ZX-12), ARC DECRA Project through grant DE200100964, NSF under grants III-1763325, III-1909323, and SaTC-1930941.

REFERENCES

[1] Charu C Aggarwal and Karthik Subbian. 2012. Event detection in social streams. In *Proceedings of the SDM*. SIAM, 624–635.
[2] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. 2018. Mine: mutual information neural estimation. In *Proceedings of the ICML*.
[3] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *JMLR* 3, Jan (2003), 993–1022.
[4] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the ECCV*. 532–547.
[5] Songgaojun Deng, Huzefa Rangwala, and Yue Ning. 2019. Learning Dynamic Context Graphs for Predicting Social Events. In *Proceedings of the SIGKDD*. ACM, 1007–1016.
[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Proceedings of the KDD*, Vol. 96. ACM, 226–231.
[7] Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. 2009. Normalized mutual information feature selection. *TNNLS* 20, 2 (2009), 189–201.
[8] Mateusz Fedoryszak, Brent Frederick, Vijay Rajaram, and Changtao Zhong. 2019. Real-time event detection on social data streams. In *Proceedings of the SIGKDD*. ACM, 2774–2782.
[9] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S Yu, and Hongjun Lu. 2005. Parameter free bursty events detection in text streams. In *Proceedings of the VLDB*. 181–192.
[10] Lukas Galke, Iacopo Vagliano, and Ansgar Scherp. 2019. Can Graph Neural Networks Go" Online"? An Analysis of Pretraining and Inference. In *Proceedings of the ICLR*.
[11] Alexander Gepperth and Barbara Hammer. 2016. Incremental learning algorithms and applications. In *Proceedings of ESANN*.
[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the SIGKDD*. ACM, 855–864.
[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the NeurIPS*. 1024–1034.
[14] Alexander Hermans, Lucas Beyer, and Bastian Leibe. 2017. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737* (2017).
[15] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2019. Learning deep representations by mutual information estimation and maximization. In *Proceedings of the ICLR*.
[16] Linmei Hu, Bin Zhang, Lei Hou, and Juanzi Li. 2017. Adaptive online event detection in news streams. *KBS* 138 (2017), 105–112.
[17] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of the WWW*. 2704–2710.

[18] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the ICLR*.
[19] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the ICML*. PMLR, 957–966.
[20] Keqian Li, Wei Lu, Smriti Bhagat, Laks VS Lakshmanan, and Cong Yu. 2014. On social event organization. In *Proceedings of the SIGKDD*. ACM, 1206–1215.
[21] Bang Liu, Fred X Han, Di Niu, Linglong Kong, Kunfeng Lai, and Yu Xu. 2020. Story Forest: Extracting Events and Telling Stories from Breaking News. *TKDD* 14, 3 (2020), 1–28.
[22] Yaopeng Liu, Hao Peng, Jie Guo, Tao He, Xiong Li, Yangqiu Song, Jianxin Li, et al. 2018. Event detection and evolution based on knowledge base. In *Proc. KBCOM*.
[23] Yaopeng Liu, Hao Peng, Jianxin Li, Yangqiu Song, and Xiong Li. 2020. Event detection and evolution in multi-lingual social streams. *FCS* 14, 5 (2020), 1–15.
[24] Zhiwei Liu, Yang Yang, Zi Huang, Fumin Shen, Dongxiang Zhang, and Heng Tao Shen. 2017. Event early embedding: predicting event volume dynamics at early stage. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 997–1000.
[25] Augusto Q Macedo, Leandro B Marinho, and Rodrygo LT Santos. 2015. Context-aware event recommendation in event-based social networks. In *Proceedings of the RecSys*. ACM, 123–130.
[26] Andrew J McMinn, Yashar Moshfeghi, and Joemon M Jose. 2013. Building a large-scale corpus for evaluating event detection on twitter. In *Proceedings of the CIKM*. ACM, 409–418.
[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*.
[28] Ozer Ozdikis, Pinar Karagoz, and Halit Oğuztüzün. 2017. Incremental clustering with vector expansion for online event detection in microblogs. *SNAM* 7, 1 (2017), 56.
[29] Hao Peng, Jianxin Li, Qiran Gong, Yangqiu Song, Yuanxing Ning, Kunfeng Lai, and Philip S Yu. 2019. Fine-grained event categorization with heterogeneous graph convolutional networks. In *Proceedings of the IJCAI*. AAAI Press, 3238–3245.
[30] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the EMNLP*. 1532–1543.
[31] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
[32] Zafar Saeed, Rabeeh Ayaz Abbasi, Onaiza Maqbool, Abida Sadaf, Imran Razzak, Ali Daud, Naif Radi Aljohani, and Guandong Xu. 2019. What's happening around the world? A survey and framework on event detection techniques on Twitter. *Journal of Grid Computing* 17, 2 (2019), 279–312.
[33] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the CVPR*. IEEE, 815–823.
[34] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A survey of heterogeneous information network analysis. *TKDE* 29, 1 (2016), 17–37.
[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the NeurIPS*. 5998–6008.
[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of the ICLR*.
[37] Petar Velickovic, William Fedus, William L Hamilton, Pietro Lio, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *Proceedings of the ICLR*.
[38] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *JMLR* 11 (2010), 2837–2854.
[39] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *Proceedings of the WWW*. 2022–2032.
[40] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. 2017. Embedding identity and interest for social networks. In *Proceedings of the WWW*. 859–860.
[41] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous Network Representation Learning: Survey, Benchmark, Evaluation, and Beyond. *arXiv preprint arXiv:2004.00216* (2020).
[42] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the ICML*. PMLR, 40–48.
[43] Yu and Shi. 2003. Multiclass spectral clustering. In *Proceedings of the ICCV*. IEEE, 313.
[44] Weiren Yu, Jianxin Li, Md Zakirul Alam Bhuiyan, Richong Zhang, and Jinpeng Huai. 2017. Ring: Real-time emerging anomaly monitoring system over text streams. *TBD* (2017).
[45] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. In *Proceedings of the NeurIPS*. 11983–11993.
[46] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the SIGKDD*. ACM, 793–803.
[47] Kuo Zhang, Juan Zi, and Li Gang Wu. 2007. New event detection based on indexing-tree and named entity. In *Proceedings of the SIGIR*. ACM, 215–222.
[48] Xiangmin Zhou and Lei Chen. 2014. Event detection over twitter social media streams. *VLDB Journal* 23, 3 (2014), 381–400.