A Partition and Interaction Combined Model for Social Event Popularity Prediction

Guandan Chen[1,2], Qingchao Kong[1,*] , Wenji Mao[1,2], Daniel Zeng[1,2]
[1]The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, China
[2]University of Chinese Academy of Sciences, China
{chenguandan2014, qingchao.kong, wenji.mao, dajun.zeng}@ia.ac.cn

***Abstract*—Social media platforms make the spread of social event information quicker and more convenient. Some of these social events may become hot topics, which highlights the importance of event popularity prediction in public management, decision making and other security related applications. Due to the complexity of social event itself, it has two unique characteristics which most previous popularity prediction work has ignored: (1) the discussion of an event itself may consist of several components, e.g. different sub-events, different stances or different user communities; (2) the popularity of an event can be influenced by other related events. To address its unique characteristics, we propose an event popularity prediction model combining partition and interaction. We employ reinforcement learning to automatically partition an event into components and recognize related events. Then we predict event popularity by modeling component information and interactions between related events. Experimental results on a real world dataset show that our proposed model can outperform the competitive baseline methods.**

Keywords—popularity prediction, information cascade, reinforcement learning

I. INTRODUCTION

The development of social media greatly facilitates information cascade, and fast information delivery has brought new challenges to security informatics, public management and business. Among many social events discussed in social media platforms, only a small number of them will become popular. Event popularity prediction aims to forecast popular events in an early time. It can help our understanding of the trends and tendency of public opinions behind user behaviors and facilitates many applications in the security related domain. For example, it can support emergency response and decision making by knowing the potential impact of natural disasters and social unrest. It can also provide valuable information in business domain.

Given the importance of popularity prediction in many applications, there are a branch of related works on this topic in recent years. The representative research on popularity prediction primarily falls into three categories: feature-engineering based models [1, 2], generative models [3, 4], and deep learning based models [5-7]. Feature-engineering based models extract features from text content, user information and time series. These models usually require domain knowledge and time-consuming feature engineering to design useful features. Generative models try to explain popularity trends using stochastic processes. Hawkes processes based model [4] is the state-of-the-art generative model for popularity prediction. It explains popularity evolutions using user influence, self-exciting mechanism and time-decay effect. However, these generative models usually have strong hypotheses, and mainly focus on information of diffusion processes, ignoring text content information. Recently, some deep learning based models demonstrate good performances in popularity prediction. The key insight of these models is to learn strong feature representations in an end-to-end manner, and do not rely on feature engineering or any strong hypothesis. For example, DeepHawkes [6] uses Gated Recurrent Units (GRU) to encode each cascade path, and employs weighted average pooling to combine features from all cascade paths.

However, there are some unique characteristics of social event popularity that previous works have not considered. Firstly, the discussion of an event itself may consist of several components, e.g. different sub-events, different stances or different user communities. Secondly, the popularity of an event may be influenced by other related events. Most previous works have ignored these event structures (e.g. different sub-events, different stances etc.), and interactions between different events. To address these unique characteristics, in this paper, we propose an **E**vent **P**opularity **P**rediction (EPP) model for social media event combining partition and interaction. EPP learns to partition an event into components and recognize related events. It then predicts popularity by modeling component information and interactions between the related events. Specifically, for partition, considering that separating a set of tweets or users into several components is a combinatorial problem and non-differential, we employ reinforcement learning technique in the model design.

Our work makes several contributions:

- We propose a novel popularity prediction model considering event components and interactions between related events.
- We are among the first to apply reinforcement learning to model different components of events for popularity prediction.
- Experimental results on a real world dataset show that our proposed model outperforms strong baseline methods.

II. RELATED WORK

A wide variety of models have been proposed for popularity prediction. These methods fall into three main categories: feature-engineering based models, generative models and deep

* Corresponding author: Qingchao Kong

978-1-5386-7848-0/18/$31.00 ©2018 IEEE

learning based models. We provide a brief discussion of the representative works here.

One branch of popularity prediction research tries to design useful features for popularity prediction. These works [1, 2, 8] have found some features about text, user, time series helpful for popularity prediction. For example, Tsur et al. [1] propose many lexicon features, and some topic, user, time series related features. Aiello et al. [2] adopt features such as topic distribution of text content, divergence of probabilistic language models etc. Weng et al. [8] have found some useful features about user network and user communities. These methods usually require careful engineering and need much domain knowledge.

Generative models try to explain popularity trends using stochastic processes [4, 9, 10]. Hawkes processes based model [4] is the state-of-the-art generative model for popularity prediction. It explains popularity evolutions using user influence, self-exciting mechanism and time-decay effect. However, these generative models usually have strong hypotheses, and mainly focus on information of diffusion processes, ignoring text content information.

Recently, some deep learning based models show good performances. These models learn to predict popularity in an end-to-end manner, and do not rely on feature engineering or any strong hypothesis. DeepCas [5] is a two-stage approach. It firstly takes random walks on the local network, then uses Gated Recurrent Units (GRU) and attention mechanism to predict popularity. DeepHawkes [6] uses Gated Recurrent Units (GRU) to encode each cascade path, and employs weighted average pooling based on time decay effect to combine features from all cascade paths. ANPP [7] is an attention-based deep neural popularity prediction model, which uses three encoders to extract features from text content, user and time series, then fuse them to predict popularity. In general, deep learning based popularity prediction models outperform other models.

The popularity of social event has two unique characteristics. Firstly, the discussion of an event itself may consist of several components, e.g. different sub-events, different stances or different user communities. Secondly, the popularity of an event may be influenced by other related events. However, most previous works have ignored these event structures (e.g. different sub-events, different stances etc.), or interactions between different events. Our model uses an end-to-end method to capture these two unique characteristics in social event popularity prediction.

III. PROPOSED MODEL

A. Problem Formulation

We define the popularity of a social event as the number of tweets discussing the event. Considering most applications do not require the accurate value of popularity, we transform the popularity prediction problem into predicting whether the future popularity of an event will exceed a given threshold. Specifically, after observing an event during a time period $[T_s, T_s + t_o]$, we make a prediction of whether the number of tweets discussing the event during $[T_s, T_s + t_p]$ will exceed a given threshold, where T_s is the start time of the event discussion, and t_o is observation time that measures how long we observe, and t_p (usually we have $t_p > t_o$) is the lifecycle of the event.

B. Structure of the Proposed Model

The proposed model consists of four parts, namely basic encoder, component encoder, related event encoder and fusion layer. The basic encoder encodes users, text content and time series to get their representations, and combines them to get a vector representation of the event. These representations will then be fed into the component encoder and the related event encoder. The component encoder learns to partition an event into several components and extracts features from components. The related event encoder recognizes related events of the target event, and models interactions between them. The fusion layer combines features from three encoders, and predicts popularity.

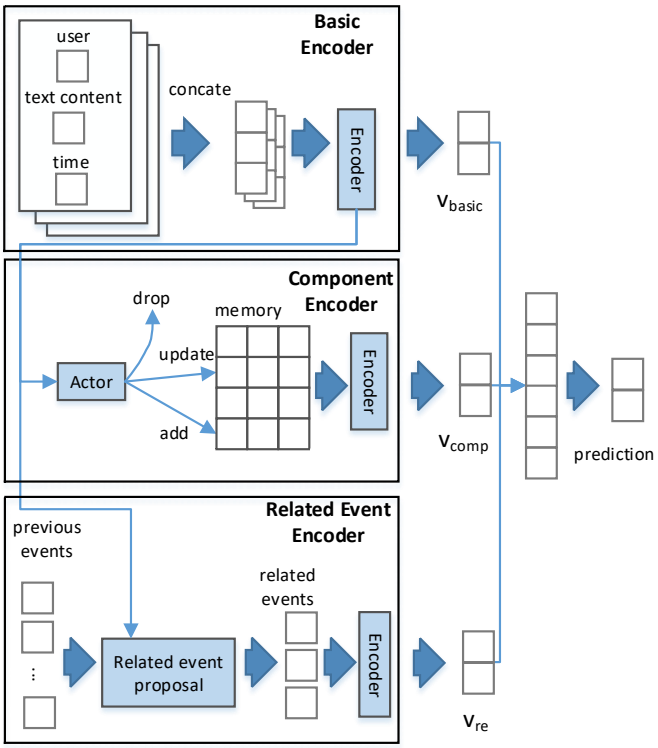

Fig. 1. Structure of the Proposed Model

C. Basic Encoder

The basic encoder encodes users, text content and time series to get their representations, and combines them to get a vector representation of the event. The structure of our basic encoder is similar to ANPP [7], and we only add a time embedding to the text and user sequence representations. Since our main contributions are the component encoder and the related event encoder, we only briefly introduce the structure of the basic encoder. For more details, please refer to ANPP.

An event can be defined as a sequence of tweets discussing it, denoted as $\{S_1, S_2, \dots, S_n\}$, where S_i consists a user (the author) u_i, a word sequence $[w_{i,1}, w_{i,2}, \dots, w_{i,m_i}]$ and the publication time T_i.

For user information, we map each user into an embedding vector using Node2vec [11], i.e.

$$v_i^{(u)} = E^{(u)} u_i \quad (1)$$

where $E^{(u)}$ is the embedding matrix for users, u_i is one-hot representation for the user of *i*-th tweet, and $v_i^{(u)}$ is its embedding vector.

For text content, since each tweet can be defined as a sequence of words, we firstly map each word into a vector using Glove [12]. Then we use a bidirectional GRU (BiGRU) and attention mechanism [13] to encode the sequence of words. BiGRU maps the input sequence into a sequence of states, where each state represents for the corresponding input and its context information. Attention mechanism maps the sequence into a vector representation by weighted summing the sequence, which can focus on the important states by assigning high attention weights to them. Specifically,

$$v_{i,j}^{(w)} = E^{(w)} w_{i,j} \quad (2)$$

$$v_i^{(text)} = Att\left(BiGRU\left(\left[v_{i,1}^{(w)}, v_{i,2}^{(w)}, \dots v_{i,m_i}^{(w)}\right]\right)\right) \quad (3)$$

where $E^{(w)}$ is the embedding matrix for words, $w_{i,j}$ is one-hot representation for *j*-th word in *i*-th tweet, $v_{i,j}^{(w)}$ is its embedding vector, m_i is the length of the word sequence, and $v_i^{(text)}$ is the vector representation for text content of the *i*-th tweet. Att is attention mechanism, which outputs a vector by weighted summing a sequence of vectors.

The representation of time information consists of three parts: (1) a vector representation for the hour of the publication time; (2) a vector representation for the weekday of the publication time; (3) results from a set of cosine function representing time interval between the time that the event starts being discussed and the publication time, i.e.

$$[cos(\beta t_i), cos(2\beta t_i), \dots, cos(K\beta t_i)] \quad (4)$$

The former two vectors are both optimized during training. Here β is a trainable parameter, and t_i is the time interval between the time that the event starts being discussed and the publication time of the *i*-th tweet, i.e. $t_i = T_i - T_s$, and K is the dimension. The vector representation of time information is the concatenation of these three vectors, denoted as $v_i^{(time)}$.

Each tweet is represented as the concatenation of the vector representations of user, text and time information, i.e.

$$v_i = \left[v_i^{(u)}, v_i^{(text)}, v_i^{(time)}\right] \quad (5)$$

Suppose there are n tweets discussing the event, we use a bidirectional GRU and attention mechanism to encode the whole sequence, i.e.

$$[h_1, h_2, \dots, h_n] = BiGRU([v_1, v_2, \dots, v_n]) \quad (6)$$

$$v_b = Att([h_1, h_2, \dots, h_n]) \quad (7)$$

where h_i is the *i*-th state of bidirectional GRU and v_b is the feature produced by the basic encoder.

D. Component Encoder

Component detection as a reinforcement learning problem. The component encoder captures the component information of an event, e.g. different sub-events, different stances or different user communities. In fact, it is a combinatorial problem to separate a set of tweets into several sets that correspond to different components. Thus, it is hard to optimize parameters directly using gradient backpropagation. To solve this issue, we cast component detection as a reinforcement learning problem. In reinforcement learning problem, an actor will take actions following a policy at every step and get a reward signal from the environment. During learning, we try to optimize the parameters of the policy to maximize the expected accumulated reward. Specifically, in the component encoder, we view the set of tweets as a sequence, and the component encoder processes tweets one by one. For each tweet, the actor follows a policy to select an action from three options to deal with it, i.e. append it to one existing component, add a new component, or view it as noise and drop it. Then, the actor receives a reward from the popularity prediction result (whether the prediction is correct) after processing all tweets. Here, the policy is a parameterized function, which takes the state (consisting of current tweet, current components) as input and outputs a probability distribution for taking actions. In this way, we can learn to detect components from popularity prediction result in an end-to-end manner, without pre-defined heuristic knowledge to detect component or time-consuming extra data annotation. The component encoder is detailed as follows.

State. The component encoder updates component representations recursively, and output a component feature at the last step. At the *i*-th step, the state includes v_i, a key k_i and existed component representations M, where k_i can be text content representation v_i^{text} or user representation v_i^u, corresponding to partition an event into components according to text content or user information respectively. Here v_i and k_i are all computed by the basic encoder. The matrix M is used to store the representations of different components, serving as a memory module [14], where each row M_j is the vector representation of one component.

Actor network. An actor network is used to select an action from following options: (1) Add a new component representation to M; (2) Select an existing component in M, and update its representation; (3) View this input as noise, and drop it. We now describe the above procedure in detail. In the actor network, we first extract features from the component representation M_j and the key k_i using

$$x_j^i = ReLU\left(Dense\left(\left[M_j + k_i, M_j \odot k_i\right]\right)\right), j \in [1, m] \quad (8)$$

where x_j^i is the feature for *j*-th component at *i*-th step. $ReLU$ represents rectified linear unit, $Dense$ is a dense connected layer, $[\cdot,\cdot]$ and $\odot$ represent for concatenation operation and element-wise multiplication respectively. Then a max pooling operation is applied,

$$f_q^i = MaxPool\left(\left[x_1^i, x_2^i, \dots, x_m^i\right]\right) \quad (9)$$

$$p_i^{act} = softmax\left(W_{act} f_q^i + b_{act}\right) \quad (10)$$

where m is the number of components in the memory, and p_i^{act} is the action distribution for three kinds of actions (i.e. add, update or drop).

Update. If the action is "update", then we select a component in the memory module according to p_i^{comp} in Eq. (12) and update its representation according to Eq. (14), i.e.

$$z_j^i = ReLU\big(Dense\big(x_j^i\big)\big), j \in [1, m] \quad (11)$$

$$p_i^{comp} = softmax\big(\big[z_1^i, z_2^i, \dots, z_m^i\big]\big) \quad (12)$$

$$j' \sim p_i^{comp} \quad (13)$$

$$M_{j'} = tanh\big(W_u\big[v_i, M_{j'}\big] + b_u\big) \quad (14)$$

Add. If the action is "add", a new component is appended to M, i.e.

$$M_{m+1} = tanh(Dense(v_i)) \quad (15)$$

Reward. During training, the actor learns to take actions so as to maximize the expected reward. Our reward signal $r = 1$ when the prediction is correct, and it is 0 otherwise. The reward is time-delayed, which is valid only after we process all the observed tweets and make a prediction. At every running time, we sample an action from the policy, and thus the expected reward is estimated from the average value of the samples.

Component feature. After processing all the n tweets, we extract features from the memory module, i.e. the component feature, which can be calculated as Eq. (16):

$$v_{comp}^k = Att(BiGRU([M_1, M_2, \dots, M_m])) \quad (16)$$

where *k* is the key in Eq. (8). Thus the final component features can be calculated as Eq (17):

$$v_{comp} = \big[v_{comp}^u, v_{comp}^{text}\big] \quad (17)$$

where v_{comp}^u and v_{comp}^{text} refers to v_{comp}^k when the key *k* is user or text content respectively.

E. Related Event Encoder

The related event encoder selects the events related to the target event from previous events, and extract features for popularity prediction from them. The selection of several related events from a set of events is also a combinatorial problem. Thus, similar to the component encoder, we learn a parameterized function that output a probability distribution corresponding to the confidence of being a related event. During training we try to optimize the expected reward of this function, where the expected reward is estimated by sampling. In this way, we can learn the related event encoder in an end-to-end manner. The selection of related events can be viewed as a special case of reinforcement learning, that only has one step in a sequence of actions. Thus, we still use reinforcement learning to optimize the related event encoder. Details of the related event encoder are as follows.

Related event proposal. For the related event encoder, we design a related event proposal network and learn to recognize related events. To reduce computational complexity, we select some candidates for related events from previous events that have at least K_{su} users or K_{sw} words in common with the target event, where K_{su} and K_{sw} are hyperparameters. Denote the representations of candidate events as $[v_1^{cand}, v_2^{cand}, \dots, v_{m_{re}}^{cand}]$, where m_{re} is the number of candidate events. For the *j*-th candidate event, the related event proposal network outputs a p_j^{re}. The higher p_j^{re} is, the more confidence we have to believe that the *j*-th candidate event is related to the target event. Specifically, the probability of a candidate being selected as a related event is computed from

$$x_j^{cand} = ReLU\big(Dense\big(\big[v_j^{cand} + v_b, v_j^{cand} \odot v_b\big]\big)\big) \quad (18)$$

$$z_j^{cand} = ReLU\big(Dense\big(ReLU\big(Dense\big(x_j^{cand}\big)\big)\big)\big) \quad (19)$$

$$p^{re} = softmax\big(\big[z_1^{cand}, z_2^{cand}, \dots, z_{m_{re}}^{cand}\big]\big) \quad (20)$$

Related event feature. During training, we randomly sample K_{re} related events from candidate events according to p^{re}, while during testing we select K_{re} related events with the largest probability in p^{re}. Let the representations of the selected related events be $[v_1^{re}, v_2^{re}, \dots, v_{K_{re}}^{re}]$. We extract the feature for popularity prediction from the related events, which is computed from:

$$v_{re} = Att\big(BiGRU\big(\big[v_1^{re}, v_2^{re}, \dots, v_{K_{re}}^{re}\big]\big)\big) \quad (21)$$

F. Fusion layer

To predict event popularity, we fuse features from the basic encoder, the component encoder, and the related event encoder. Features are concatenated as a single vector, which is passed through two dense layers to produce the prediction result. In addition, considering that features from different encoders may have different distributions, we apply batch normalization [15] to the concatenated vector.

Specifically, it is computed from:

$$z = ReLU\big(Dense\big(BatchNorm\big(\big[v_b, v_{comp}, v_{re}\big]\big)\big)\big) \quad (22)$$

$$\hat{y} = softmax(Dense(z)) \quad (23)$$

where $BatchNorm$ is batch normalization, and ỹ is the prediction result. The higher $\hat{y}_c$ is, the more confidence we have to believe the target event will have the popularity value in *c*-th range.

G. Training

We use cross entropy as the loss function, which is computed as:

$$l = \sum_c y_c log(\hat{y}_c) \quad (24)$$

where y_c is the true label, and $y_c = 1$ if the popularity is in *c*-th range, otherwise it is 0.

Auxiliary loss regularization. Considering that the related event encoder contains a sampling process with large variance, we design an auxiliary loss to produce more stable training process. The auxiliary loss tries to force features from the basic encoder useful for popularity prediction, which is computed from

$$\hat{y}^a = softmax(Dense(ReLU(Dense(v_b)))) \quad (25)$$

$$l^a = \sum_c y_c log(\hat{y}_c^a) \quad (26)$$

where $\hat{y}^a$ is the prediction result using features from the basic encoder, and l^a is cross entropy loss for this prediction.

We adopt Adagrad [16], a widely used optimization method, to train the model. For training the actor network and related event proposal network, we employ the PPO algorithm [17], a policy gradient method for reinforcement learning, which uses a clipped surrogate objective to enhance stability during training.

IV. EXPERIMENT AND DISCUSSION

A. Dataset

To evaluate the performance of our proposed popularity prediction model with baseline methods. We take Twitter as an exemplar social media platform. Twitter is a large social media platform with diverse users coming from different countries, and is an important platform for people to express opinions and discuss recent events.

The dataset was collected using Twitter API (https://developer.twitter.com) from Aug. 9, 2016 to Dec. 10, 2016. In Twitter, users usually use hashtag to mark the event or topic they discussing, which enables us to find messages about a specific event. We first drop hashtags with less than 10 tweets, then manually merge hashtags discussing the same event, and remove non-event hashtags. The observation time t_o is set to 1, 6, 12 or 24 hours in our dataset. Top 20% of events with regard to the popularity in the whole lifecycle are considered as popular events, and we randomly sample the same number unpopular events from the rest of events. Thus, each sample in our dataset is an event, which contains users, tweets and timestamps of the tweets discussing the event during the observation time, and the label is whether the event is popular considering the whole lifecycle of the event. We use 80% of the events in the first three months as the training set, the rest 20% of the events as the validation set, and the events in the last month as the test set. Table I shows the detail of the dataset.

TABLE I. STATISTICS OF THE DATASET

#events	#tweets	avg. #tweets per event	max #tweets per event
41,035	4,561,375	111	234,944

B. Baseline Methods

We compare our model with the existing deep learning based popularity prediction models and several representative feature-engineering based and generative models. Following methods are chosen as baseline methods for our comparison.

(1) *Tsur's* [1]: Adopts many lexicon features, associated with a few user and time series features.

(2) *Aiello's* [2]: Proposes many features about time series, user network and probabilistic language model.

(3) *Hawkes* [4]: Models popularity dynamics as Hawkes processes.

(4) *DeepCas* [5]: Takes random walks on the local network, then uses GRU and attention mechanism to extract features from random walk paths to predict popularity.

(5) *DeepHawkes* [6]: An extension of Hawkes model, which employs user embedding, GRU and weighted sum pooling to encode cascade paths.

(6) *ANPP* [7]: An attention-based deep neural popularity prediction model, which uses three encoders to encode text content, user and time series and fuses them to predict popularity.

We also compare to the following variations of our proposed model.

(1) *Basic encoder*: Only uses features from the basic encoder.

(2) *EPP-NCE*: Only uses the basic encoder and the related event encoder.

(3) *EPP-NRE*: Only uses the basic encoder and the component encoder.

(4) *EPP-NALR*: Removes auxiliary loss regularization from the proposed model.

We use accuracy as the evaluation metric.

C. Results

Table II shows prediction performances of our proposed model and the baseline methods. As we can see, our proposed model outperforms all baseline methods, with significant improvements of accuracies under different observation time settings. Moreover, the improvements are more prominent when observation time is short. According to the definition of popularity prediction, shorter observation time means less information for prediction. Thus, it requires more prediction ability for short observation time.

TABLE II. ACCURACIES OF DIFFERENT METHODS

Method	Observation time T_o (hours)			
	1	6	12	24
Tsur's	0.612	0.653	0.682	0.722
Aiello's	0.633	0.701	0.728	0.781
Hawkes	0.590	0.669	0.709	0.749
DeepCas	0.580	0.658	0.707	0.780
DeepHawkes	0.588	0.691	0.725	0.791
ANPP	0.651	0.723	0.764	0.818
Basic encoder	0.670	0.740	0.766	0.818
EPP-NCE	0.683	0.737	0.774	0.825
EPP-NRE	0.686	0.743	0.769	0.822
EPP-NALR	0.681	0.721	0.756	0.818
EPP	**0.721**	**0.751**	**0.783**	**0.828**

In the baseline methods, Tsur's and Aiello's are both feature-engineering based methods. Aiello's is much more effective than Tsur's. It shows that the selection of features influences the performance of feature-engineering based methods. Hawkes is the state-of-the-art generative model. Although it possesses better interpretability, its performance is lower than Aiello's. Deepcas, DeepHawkes and ANPP are all deep learning based methods. We found Deepcas can not beat Aiello's in our dataset, which shows well-designed features can also reach good performance. Both DeepHawkes and ANPP show significant performance improvement compared to the feature-engineering based methods. This indicates that well-designed deep learning based models are good at popularity prediction task. ANPP shows the best performance among the baseline methods, which uses three encoders to get representations of text content, user and time series, and then fuses them to predict popularity. It shows the advantage of utilizing more useful information.

By comparing different variations of our proposed model, we found that component encoder and related event encoder are both useful for event popularity prediction. In addition, our basic encoder shows a slight performance improvement compared to ANPP. It is because we add a time embedding which can better capture time information. Removing the component encoder or the related event encoder will lower the performance of our proposed model, which shows the effectiveness of these encoders. Without specially designed auxiliary loss regularization, our model can not reach good performance and sometimes even get worse than the basic encoder, because the high variance in random sampling may influence the stability of the training process.

To further investigate the effectiveness of the component encoder and the related event encoder, we provide a case study in Table III. We found the components and related events are reasonable. The event "#whereisNajeebAhmed" is about an Indian student Najeeb Ahmed got missing. As shown in Table III, the first component of this event is about people's appealing for help. The second component discusses that the family of Najeeb Ahmed was arrested during the protest, and people condemned the government. The third component is about a student organization named ABVP. As for the related events, most of them are about India and Indian politician Modi. These related events and the target event all have some connections to India government, and people discussing these related events may be interested in the target event.

TABLE III. AN EXAMPLE OF COMPONENTS AND RELATED EVENTS

Event	#WhereIsNajeebAhmed
Component 1	Bring back Najeeb or send this mother to him where he is. Mother of Najeeb is madly waiting to hear her sons voice.
Component 2	Shame on Delhi police for dragging The mother of Najeeb as if she's a criminal. This is utter disgust. Sad part of Government and especially police.
Component 3	Why is Delhi Police not interrogating those ABVP goons who have beaten Najeeb on the day before he has been missing? Arrest ABVP leaders of JNU.
Related Events	#SrinagarSlapsModi #NawazRattlesIndia #NoSpaceForIndiaInKashmir #ModiFailedAgain

V. CONCLUSIONS

In this paper, we propose a popularity prediction model for social events combining partition and interaction. The model learns to partition an event into components, and select related events that may influence the popularity of the target event. It then predicts event popularity by modeling component information and interactions between related events. We employ the reinforcement learning algorithm to train the model, and propose an auxiliary loss regularization to enhance the training stability. Experimental results on the Twitter dataset show that our proposed model outperforms the competitive baseline methods.

ACKNOWLEDGMENT

This work is supported in part by Ministry of Science and Technology of China (Grant No. 2016QY02D0305), National Natural Science Foundation of China (Grant No. 71702181, 71621002 and 61671450), Key Program of the Chinese Academy of Sciences (Grant No. ZDRW-XH-2017-3).

REFERENCES

[1] O. Tsur and A. Rappoport, "What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities," in *Proceedings of the fifth ACM international conference on Web search and data mining*, 2012, pp. 643-652: ACM.

[2] L. M. Aiello *et al.*, "Sensing trending topics in Twitter," *IEEE Transactions on Multimedia,* vol. 15, no. 6, pp. 1268-1282, 2013.

[3] P. Bao, H.-W. Shen, X. Jin, and X.-Q. Cheng, "Modeling and predicting popularity dynamics of microblogs using self-excited hawkes processes," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 9-10: ACM.

[4] S. Mishra, M.-A. Rizoiu, and L. Xie, "Feature driven and point process approaches for popularity prediction," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1069-1078: ACM.

[5] C. Li, J. Ma, X. Guo, and Q. Mei, "DeepCas: An end-to-end predictor of information cascades," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 577-586: International World Wide Web Conferences Steering Committee.

[6] Q. Cao, H. Shen, K. Cen, W. Ouyang, and X. Cheng, "DeepHawkes: Bridging the Gap between Prediction and Understanding of Information Cascades," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 1149-1158: ACM.

[7] G. Chen, Q. Kong, and W. Mao, "An attention-based neural popularity prediction model for social media events," in *Intelligence and Security Informatics (ISI), 2017 IEEE International Conference on*, 2017, pp. 161-163: IEEE.

[8] L. Weng, F. Menczer, and Y.-Y. Ahn, "Predicting Successful Memes Using Network and Community Structure," in *ICWSM*, 2014.

[9] Q. Zhao, M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec, "SEISMIC: A Self-Exciting Point Process Model for Predicting Tweet Popularity," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1513-1522: ACM.

[10] H. Shen, D. Wang, and C. Song, "Modeling and predicting popularity dynamics via reinforced Poisson processes," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014, pp. 291-297.

[11] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855-864: ACM.

[12] J. Pennington, R. Socher, and C. Manning, "Glove: Global Vectors for Word Representation," in *Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1532-1543.

[13] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1480-1489.

[14] Ł. Kaiser, O. Nachum, A. Roy, and S. Bengio, "Learning to Remember Rare Events," *arXiv preprint arXiv:1703.03129*, 2017.

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[16] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, no. 7, pp. 257-269, 2011.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.