Predicting Influence Probabilities using Graph Convolutional Networks

Jing Liu*, Yudi Chen*, Duanshun Li†, Noseong Park*, Kisung Lee‡, and Dongwon Lee§
*George Mason University, Fairfax, VA, USA
†University of Alberta, Edmonton, AB, Canada
‡Louisiana State University, Baton Rouge, LA, USA
§Penn State University, State College, PA, USA
{jliu30,ychen55}@gmu.edu, duanshun@ualberta.ca, npark9@gmu.edu, klee76@lsu.edu, dongwon@psu.edu

***Abstract*— As one of the fundamental tasks in data analytics, *Influence Maximization* methods have been widely used in many real-world applications. For instance, in social network analysis, after building a directed graph, where edges are weighted with influence probabilities, influence maximization methods can be used to find a set of users who can maximize the spread of information under certain cascade models. Despite their successes, however, one critical weakness of existing influence maximization methods lies in the fact that edges are weighted with *historical* probabilities. As such, influence maximization methods perform sub-optimal if there occur non-trivial changes in future. In response to this challenge, in this work, we propose a novel prediction-driven influence maximization method that accurately predicts *future* influence probabilities using graph convolutional networks and find seed users based on the predicted probabilities. The experiments with five real-world datasets show that our prediction accuracy is accurate (e.g., mean absolute percentage error less than 0.1) in many cases, and our prediction-driven influence maximization is very close to the optimal.**

***Index Terms*—influence maximization, graph convolutional network, regression**

1 Introduction

Viral marketing via *Influence Maximization* has become a popular research topic as social networks have become part of our daily life. Many people connect to and acquire information from social networks on a daily basis. Information diffusion over such social networks is sometimes more effective than that over conventional media such as newspapers.

Influence maximization is to find a certain number of seed users who can maximize the spread of information over social networks, which is an NP-hard problem. There exist several real-world applications where the influence maximization played a key role, such as 2010 U.S. congressional elections, an attempt to raise awareness about HIV among homeless youth, and so on [1], [2].

Most influence maximization algorithms require an influence probability graph as an input, where a vertex means a user and each directed edge (u, v) is weighted with a probability that user u influences other user v. One limitation of existing approaches is that they typically use past historical influence probabilities. Given past tweet and retweet logs, for instance, they calculate influence probabilities and find seeds [3]–[6].

However, we think this is sub-optimal for unpredictable nature of human behaviors. People's interests quickly change, and past historical influence probabilities may not approximate future within a reasonable bound. In one of our datasets, for instance, we queried one-year tweets for few sensational topics that let people tweet and retweet a lot[1]. We divide the entire period into several equally sized time windows and found that there are non-trivial variations on their influence probabilities and as a result, on their optimal seed sets as well. The R^2 measure of influence probabilities between two consecutive time windows is 0.61 in average which is far less than our prediction accuracy around 0.99 (see the use case in Section 6.4 for more concrete examples).

One effective method against the future uncertainty is to use *robust optimization*-based influence maximization algorithm [7]. Robust optimization is a sub-field of optimization and maximizes rewards in worst-case scenarios — it has a very long history and readers are referred to [8] for a comprehensive survey. However, the robust influence maximization can be sometimes conservative for mainly pursuing stability.

Therefore, one can run an influence maximization algorithm with predicted influence probabilities instead. In order to find optimal and reliable seed users, however, the prediction accuracy should be good enough (e.g., mean absolute percentage error ≤ 0.1).

Graph convolutional networks (GCNs) are yet another neural network model specialized in dealing with graphs. It shows state-of-the-art performance in many tasks on graphs [9]. Each GCN model claims its superiority over other models so we did an extensive literature survey and design our own model customized for our purposes. To our knowledge, there are just a few GCNs predicting for edges [10], [11] and one of them considers categorical edge labels whereas we consider real-valued edge labels. Our novelty lies in designing a flexible framework predicting three popular definitions of influence probabilities. In addition, we train our GCN with randomly sampled subgraphs to reduce the training time

This paper is based upon work supported by the National Science Foundation under Grant No. 1742702, 1820609, 1915801, and 1909702. Noseong Park is the corresponding author.

[1]We used the GNIP commercial APIs that allow us to query without any restrictions.

and GPU memory requirement. However, a certain influence probability definition requires complete neighbor information for prediction (see the LT definition in Section 2.3). We prove that our subgraph training that uses random partial neighbor information in a mini-batch is safe to predict the probability definition (see Thm. (5.1)).

In our experiments, the proposed model is able to achieve a high accuracy of the mean absolute error less than 0.01 and R^2 close to 1 in almost all cases. We also show a couple of case studies for our prediction-driven influence maximization. Our contributions can be summarized as below:

1) Our proposed GCN is optimized to predict influence probabilities: i) Both vertex and edge features are considered during graph convolutional process, ii) Our graph convolutional process is defined considering information cascade process, and iii) Training with subgraphs is necessary to increase scalability whereas some influence probability prediction requires complete neighborhood information — there exists a trade-off issue of training speed and GPU memory requirement vs. prediction accuracy. However, our method theoretically guarantees appropriate training only with randomly sampled subgraphs (see Thm. (5.1)).
2) We test with five real-world datasets in various environments and show that our model outperforms many baseline methods w.r.t. various evaluation metrics.
3) We demonstrate that our prediction-driven influence maximization can infect more users than the robust influence maximization due to the high prediction accuracy.

2 Preliminaries & Related Work

2.1 Graph Convolutional Networks

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a vertex feature vector $\mathbf{f}_v$ for each vertex $v \in \mathcal{V}$, GCNs perform a classification or regression task for each vertex v — we use boldface to denote vectors.

There have been proposed several GCN methods and each GCN method can be described by the following equation with a specific choice for each of the aggregation function agr and the combination function cmb:

$$\mathbf{a}_v^i = agr(\{\mathbf{h}_u^{i-1} | u \in Nei(v)\}),$$
$$\mathbf{h}_v^i = cmb(\mathbf{a}_v^i, \mathbf{h}_v^{i-1}),$$

where i means i-th convolutional layer and $Nei(v)$ is a set of neighbors of v. After k convolutional layers, one predicts for each vertex using $\mathbf{h}_v^k$. For doing this, one more activation layer $activation(\mathbf{h}_v^k)$ is needed at the end.

The aggregation and combination functions of some selected major GCN methods are as follows: First, LGCN [12] uses l-Max_Pooling to aggregate neighbor features, which selects the l-largest values in each feature dimension from neighbor vertices. A simple concatenation and convolutional layer are used for the combination function.

Second, GraphSAGE [13] introduced three aggregators: Mean_Pooling, LSTM, and Max_Pooling. A simple concatenation and one linear layer are used to combine $\mathbf{a}_v^i$ and $\mathbf{h}_v^{i-1}$. In those perspectives, GraphSAGE has more options for the aggregation function but a simpler combination function than LGCN.

Third, Graph Attention Network (GAT) attend neighbors during its aggregation. It uses linear layers for the aggregation and combination.

Two GCNs in [10], [11] can predict missing links. However, their link prediction is different from our influence probability prediction in two aspects: i) Edges with low and high influence probabilities are all considered in the same way in link prediction, and ii) the most importantly, link prediction assumes that partial current network information is known to predict missing edges whereas in our task, we know only past information — i.e., we predict influence probabilities at time $t+1$ with the information up to time t.

2.2 Influence Maximization

Influence maximization to find a certain number of seeds is an NP-hard problem. There are two major issues regarding influence maximization: i) how to model the information cascade, and ii) how to solve the NP-hard problem.

Independent cascade (IC) and *linear threshold* (LT) are two popular information cascade models. In the IC model, a user v is activated (influenced) by its neighbor u by the probability $p_{(u,v)}$ — i.e., a coin toss model whose head (influence) probability is $p_{(u,v)}$. In the LT model, a user v is activated if a sufficient number of neighbors (larger than a threshold) are active. In our paper, we focus on these two models.

There are many techniques to solve the seed set finding problem. Because it is an NP-hard problem, all these methods try to approximate optimal solutions. Those methods can be categorized into the following three types: i) simulation-based, ii) proxy-based, and iii) sketch-based methods.

In the simulation-based methods, Monte Carlo simulations are explicitly executed to measure the performance of a seed set [14], [15]. These methods try to prune unnecessary (redundant) simulations to minimize the simulation runtime overheads.

In the proxy-based methods, a proxy that can approximate the user-level importance in the IC or LT model is used. For instance, PageRank is an effective metric to evaluate the importance of each user in a graph and choosing the top users with the highest PageRank scores is sometimes a very good approximation to the influence maximization. There are many other proxies [16]–[18]. For many others (including sketch-based methods), see a comprehensive survey introducing a wide variety of techniques in [19].

2.3 Influence Probability

We use interaction logs such as retweets among users (vertices) to construct the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The edge weight on a certain directional edge $e = (u, v)$ represents the influence probability from the start vertex u to the end vertex v. There are several methods to define the influence probability of an edge $e = (u, v)$, denoted as p_e or $p_{(u,v)}$, as follows:

1) Bernoulli Trial (BT): $p_{(u,v)} = \frac{|actions(u) \cap actions(v)|}{|actions(u)|}$,

2) Jaccard Index (JI): $p_{(u,v)} = \frac{|actions(u) \cap actions(v)|}{|actions(u) \cup actions(v)|}$,

3) Linear Threshold (LT): $p_{(u,v)} = \frac{|actions(u) \cap actions(v)|}{|actions(v)|}$,

where $actions(x)$ means a set of actions done by user x, e.g., a set of online postings retweeted or replied by user x. Note that $\sum_{x \in Nei(v)} p(x,v) = 1$ in the last LT definition — for other definitions, this is not the case.

Each probability definition has its own meaning. The first BT definition corresponds to the maximum likelihood of the Bernoulli trial of u attempting to infect v [3]. The JI definition use the Jaccard index which was originally defined to compare the similarity of two sets [3]. The LT definition was first introduced by Kempe et al. in their seminal paper introducing the linear threshold cascade model [4].

3 Motivation

Many works calculate influence probabilities based on past behavioral logs. However, influence probabilities in the future may be different from past historical probabilities due to unpredictable nature of human beings.

One countermeasure to consider the (future) uncertainty of influence probability is *robust optimization*. However, it is sometimes too conservative because it optimizes rewards in worst-case scenarios. Instead, we propose to predict future influence probabilities and remove the uncertainty.

4 Problem Definition

We consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each vertex $v \in \mathcal{V}$ and each edge $e \in \mathcal{E}$ have features $\mathbf{f}_v$ and $\mathbf{f}_e$ describing past behavioral patterns. The influence probability prediction problem can be defined as follows:

Definition 4.1. *Given a directed graph $\mathcal{G}$ whose vertex and edge features contain information up to time t, the influence probability prediction is to predict the influence probability p_e at time $t+1$ for each edge $e \in \mathcal{E}$.*

After predicting p_e at $t+1$, we use existing influence maximization algorithms to find seed users. We show some use cases where the prediction-driven influence maximization outperforms other methods.

5 Proposed Method

Our model supports many aggregation and combination selections. We also use many final activations that produce influence probabilities in [0,1].

5.1 Activation Functions

As mentioned in Section 2.1, an activation function $activation(\mathbf{h}_e^k)$ after k graph convolutional layers is required to have a final influence probability prediction for each edge $e \in \mathcal{E}$. We test the following various activation functions to predict the influence probability from u to its neighbor v:

1) Sigmoid Activation (SA): $p_{(u,v)} = sigmoid(\mathbf{h}_{(u,v)}^k \cdot \mathbf{W}^\intercal)$,
2) Tanh Activation (TA): $p_{(u,v)} = |tanh(\mathbf{h}_{(u,v)}^k \cdot \mathbf{W}^\intercal)|$,
3) Multi-logit Regression (MR): $p_{(u,v)} = \frac{\exp(\mathbf{h}_{(u,v)}^k \cdot \mathbf{W}^\intercal)}{\sum_{x \in Nei(v)} \exp(\mathbf{h}_{(x,v)}^k \cdot \mathbf{W}^\intercal)} = \frac{logit_{(u,v)}}{\sum_x logit_{(x,v)}}$,

where $\mathbf{h}_{(x,y)}^k$ is a hidden vector of the edge from x to its one neighbor y after k graph convolutional layers and $\mathbf{W}$ is a weight vector that should be trained.

Because there are several different definitions of the influence probability (see Section 2.3), we test all the above activation (prediction) functions. For instance, the last MR function is suitable to predict the LT probability definition because $\sum_{x \in Nei(v)} p(x,v) = 1$.

5.2 Graph Convolutional Network

Our proposed graph convolutional network differs from others in two perspectives: i) We consider both vertex and edge features whereas existing methods consider only vertex features and ii) We update hidden vectors following the information cascade process whereas existing methods consider all available information (see Fig. 1 (a)). Our architecture is customized toward the task of predicting the influence probability. Recall that the influence probability is defined between two connected vertices. To predict the probability, therefore, we need to consider the edge feature that contains information specific only to the two connected vertices.

5.2.1 Initial Embedding: In our paper, a vertex v (resp. an edge e) has an input feature vector $\mathbf{f}_v$ (resp. $\mathbf{f}_e$). We consider the following items in the input feature vector:

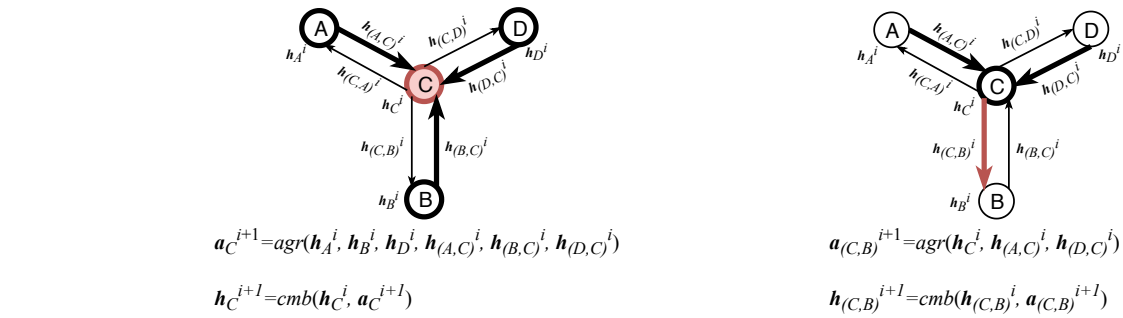
1) Each vertex v has a feature $\mathbf{f}_v$, which contains two types of features: network features and content features. Note, for our datasets which have online posting texts to derive content features, we use both the network and content features as $\mathbf{f}_v$, while for other datasets where we do not have online posting contents, $\mathbf{f}_v$ includes the network features only. The network features include i) Local clustering coefficient, ii) Out-degree/node strength, iii) Weighted PageRank, iv) The number of friends, and v) The number of followers. The content features have i) The number of original postings, and ii) From multiple postings of a user (vertex), we calculate his/her average sentiment score in [-4,4] using the method in [20][2].
2) Each directional edge $e = (u,v)$ has a feature $\mathbf{f}_e$ as follows: i) The number of replies/retweets on edge e in previous time periods, ii) Influence probability in previous time periods, and iii) Adamic-Adar index: $A(u,v) = \sum_{x \in Nei(u) \cap Nei(v)} \frac{1}{log(|Nei(x)|)}$.

Note that vertex and edge features contain different information and they are not compatible to each other during the graph convolution process. To address the issue, we embed them into one unified vector space (see Fig. 1 (b)) as follows:

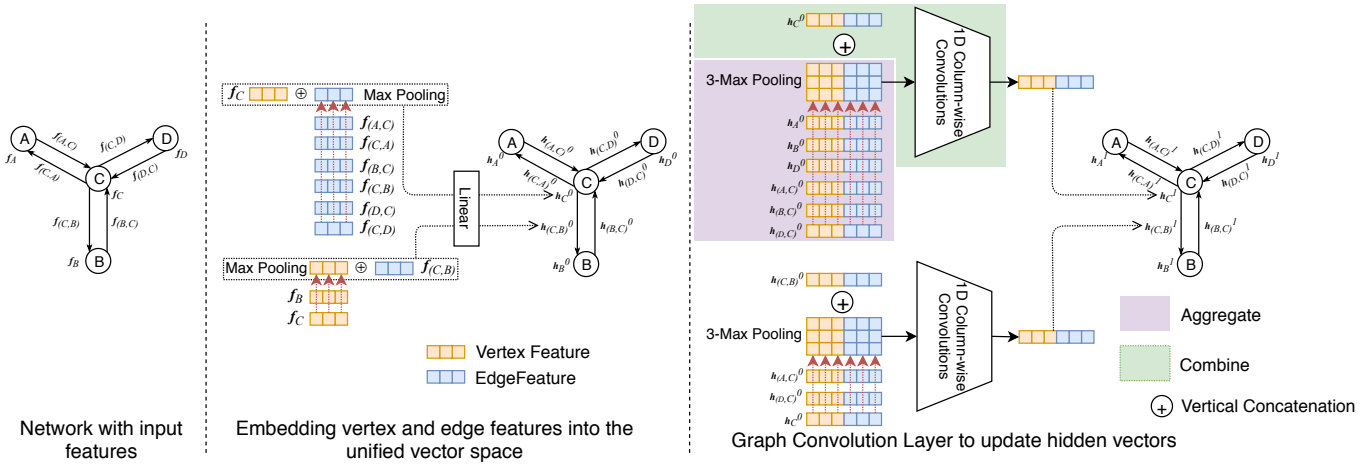
$$\begin{aligned} \mathbf{h}_v^0 &= linear\big(\mathbf{f}_v \oplus agr(\{\mathbf{f}_e | e \text{ is an incident edge to } v\})\big), \\ \mathbf{h}_e^0 &= linear\big(agr(\{\mathbf{f}_v | e \text{ is an incident edge to } v\}) \oplus \mathbf{f}_e\big), \end{aligned} \tag{1}$$

where $\oplus$ means a concatenation of two vectors. $\mathbf{h}_v^0$ and $\mathbf{h}_e^0$ mean initial hidden vectors. Note that after this initial embedding, all hidden vectors are compatible and reside in

[2]We received their JAVA source codes and appreciate their supports.



(a) Convolve hidden vectors following the information flow direction. Information flows from A/B/D to C and from (A,C)/(D,C) to (C,B). We use (x, y) to denote a directed edge from vertex x to y.



(b) Proposed initial embedding and graph convolution definitions

Fig. 1: (a) An example of how to update vertex and edge hidden vectors. (b) An example of the embedding and graph convolutional layers used in our paper. We use 1-Max_Pooling for the embedding layer and 3-Max_Pooling for the graph convolutional layer in this example. In addition to them, we also test many other options for the aggregation and combination.

the same vector space. After some preliminary experiments, we use the dimension of 8 for $\mathbf{h}_v^0$ and $\mathbf{h}_e^0$.

5.2.2 Update Hidden Vectors following Information Cascade Process: We update hidden vectors considering the information cascade process as follows:

$$\begin{aligned}
\mathbf{a}_v^{i+1} &= agr\big(\mathbf{h}_{x_1}^i, \mathbf{h}_{(x_1,v)}^i, \mathbf{h}_{x_2}^i, \mathbf{h}_{(x_2,v)}^i, \cdots, \mathbf{h}_{x_k}^i, \mathbf{h}_{(x_k,v)}^i\big),\\
\mathbf{h}_v^{i+1} &= cmb\big(\mathbf{a}_v^{i+1}, \mathbf{h}_v^i\big),\\
\mathbf{a}_{(v,x_j)}^{i+1} &= agr\big(\mathbf{h}_{(x_1,v)}^i, \cdots, \mathbf{h}_{(x_{j-1},v)}^i, \mathbf{h}_{(x_{j+1},v)}^i, \cdots, \mathbf{h}_v^i\big),\\
\mathbf{h}_{(v,x_j)}^{i+1} &= cmb\big(\mathbf{a}_{(v,x_j)}^{i+1}, \mathbf{h}_{(v,x_j)}^i\big),
\end{aligned}$$

where $x_j \in Nei(v)$ means j-th neighbor of vertex v.

Our proposed method can also be described by the general GCN equation based on the aggregation and combination functions. However, we selectively use information cascade process to update a hidden vector. In Fig. 1 (a), for instance, the hidden vector of the vertex C is updated from the hidden vectors of the three neighbors A, B, D, and their three incident edges in the left-side example, which is based on the intuition that C's infection is decided by the neighbors and the incoming edges. The hidden vector of the edge (C, B) in the right-side example can also be updated from the hidden vectors of two other incident edges to C and the hidden vector of C itself. In the figure, we highlighted those hidden vectors in bold lines. Note that they are well aligned with the information cascade process in the graph. We found that updating hidden vectors in such ways is a reasonable design choice to predict the influence probability because the influence maximization is formulated on top of the information cascade — in our experiments, other baselines that do not consider the directivity show sub-optimal performance.

5.2.3 Aggregation and Combination Functions: Fig. 1 (b) shows an example of our graph convolutional layer definitions that consists of l-Max_Pooling, where $l = 3$, and 1D column-wise convolution, i.e., the aggregation function is 3-Max_Pooling, and the combination function is 1D column-wise convolution after the vertical concatenation of $\mathbf{a}_v^{i+1}$ and $\mathbf{h}_v^i$ (or $\mathbf{a}_e^{i+1}$ and $\mathbf{h}_e^i$).

We have several other options for the aggregation and combination functions (as described in the related work section). In our experiments, we will test all those combinations.

5.2.4 Training Method: For faster training, we train the proposed GCN with randomly sampled subgraphs that consist of $m+1$ vertices. GCNs are sometimes too expensive in terms of space and time to train with whole graph every epoch so we train with randomly sampled subgraphs. This will drastically increase the scalability of our method. Our training loss definition is as follows:

$$\mathcal{L} = \sum_{\mathcal{S}\in\mathbb{S}} \sum_{u\in\mathcal{S}} \sum_{v\in Nei(u)} \|p_{(u,v)} - p'_{(u,v)}\|^2,$$

where $\mathcal{S} \subseteq \mathcal{G}$ is a subgraph sampled from $\mathcal{G}$, $p'_{(u,v)}$ is a predicted probability and $p_{(u,v)}$ is a ground truth probability. For the subgraph sampling, we first choose one center vertex randomly and perform a bread-first search from the center to choose random m neighbors.

One can ask that our MR activation sees always randomly sampled neighbors of u during the training phase and how it can provide robust influence probability predictions for all neighbors during the testing phase — other activations (i.e., SA and TA) use only $\mathbf{f}_{(u,v)}$ to predict $p_{(u,v)}$ so they are free from the question. We prove the following theorem to show that our MR activation can be trained correctly even with those random subgraphs.

Theorem 5.1. *The multi-logit regression model that guarantees a mean-absolute error of at least z for the influence probability prediction from u to its any random $m \geq 2$ neighbors will also guarantee the same mean-absolute error for the prediction of $p_{(u,v)}$ for all $u \in Nei(v)$.*

Proof. Suppose a vertex v and its i-th neighbor $u_i \in Nei(v)$. Let $n = |Nei(u)|$. There are $\binom{n}{m}$ ways to construct a subset of m random neighbors and $\binom{n}{m-1}$ of them include a specific neighbor u_i. We denote each subset of m random neighbors that contains u_i as $S_j \subseteq Nei(v)$, where $j \in \{1, 2, \cdots, \binom{n}{m-1}\}$.

A multi-logit regression trained for S_j with a mean-absolute error less than or equal to z means the following inequality:

$$-z \leq p'_{(u_i,v)} - p_{(u_i,v)} \leq z \text{ for all } u_i \in S_j,$$

where $p'_{(u_i,v)}$ is a prediction and $p_{(u_i,v)}$ is a ground truth probability. This inequality can be rewritten as follows:

$$-z + p_{(u_i,v)} \leq p'_{(u_i,v)} \leq z + p_{(u_i,v)},$$

or equivalently,

$$SUM_j \cdot (-z + p_{(u_i,v)}) \leq logit_{(u_i,v)} \leq SUM_j \cdot (z + p_{(u_i,v)}), \quad (2)$$

where $SUM_j = \sum_{u\in S_j} logit_{(u,v)}$.

Let us take the sum of Eq. (2) for all such S_j that contains u_i. Because $j \in \{1, 2, \cdots, \binom{n}{m-1}\}$,

$$\sum_{j=1}^{\binom{n}{m-1}} SUM_j \cdot (-z + p_{(u_i,v)}) \leq \binom{n}{m-1} \cdot logit_{(u_i,v)} \leq \sum_{j=1}^{\binom{n}{m-1}} SUM_j \cdot (z + p_{(u_i,v)}),$$

where

$$\sum_{j=1}^{\binom{n}{m-1}} SUM_j = \binom{n}{m-1} \sum_{u\in Nei(v)} logit_{(u,v)}.$$

Therefore,

$$\begin{aligned} &-z + p_{(u_i,v)} \leq \frac{logit_{(u_i,v)}}{\sum_{u\in Nei(v)} logit_{(u,v)}} \leq z + p_{(u_i,v)} \\ &\Rightarrow -z + p_{(u_i,v)} \leq p'_{(u_i,v)} \leq z + p_{(u_i,v)} \\ &\Rightarrow -z \leq p'_{(u_i,v)} - p_{(u_i,v)} \leq z. \end{aligned}$$

After taking the sum for all j, we have the same error bound z. In particular, note that the denominator (highlighted in red) includes all the neighbors. This implies that a multi-logit model that provides a prediction error bound z for any subset with $m \geq 2$ neighbors can also provide the same error bound for the complete set $Nei(v)$.

□

The above theorem says that we do not need to train the MR activation with all neighbors. If we train the MR activation with many random subsets of neighbors, then its predictions for all neighbors can be stabilized. This can drastically increase the training scalability of our approach. We observe that our subgraph training is about 2-20 times faster and requires about 2-20 times less GPU memory than the whole graph training with comparable accuracy.

5.3 Data Preprocessing

As we intend to predict future influence probabilities based on current vertex and edge features, we split a dataset in the following way to train and test our model. Assume the total time span of dataset is T, we use a fixed time window size w to split T into multiple time windows. We slide the time window along T with a stride s. Fig. 2 shows an example with $T = 12$, $w = 4$, and $s = 2$. T is separated into 5 time windows (t_1 to t_5). The stride s herein means the frequency of creating a new time window.

In each time window, its ground truth influence probability can be calculated by the three different definitions presented in Section 2.3. Vertex and edge features can also be extracted in each time window. In the example, we use the first four windows for training and the last window for testing. For instance, given vertex and edge features in t_i we train the model to predict the influence probabilities in t_{i+1}, where $i \leq 3$. To evaluate the accuracy, we predict for t_5 given the features of t_4. While testing, we ignore vertices and edges that do not appear in all windows because features cannot be defined in such cases. Because of this, the number of vertices and edges can be different in a dataset, depending on w and s configurations.

In real environments, each user's interests will keep changing and, in many edges, their influence probabilities will also keep fluctuating (only except some edges whose ties are strong, e.g., family members and long-term friends). Therefore, the sliding window concept is practical in real-world applications to reflecting the quickly changing environment.

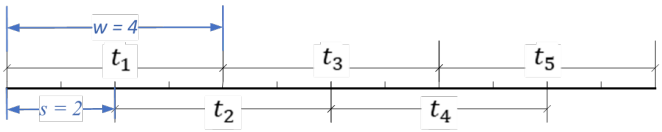


Fig. 2: An example on data preprocessing. Our task is to predict the influence probabilities in t_{i+1} given the vertex and edge features in t_i. After training with t_1, t_2, t_3, and t_4, we predict t_5.

TABLE I: Statistics on the eight experiment settings. Note that the network size varies depending on the w and s configurations (see Section 5.3).

Dataset	Total Time Span	w	s	# Vertices	# Edges	# Windows
Extended [21]	12 months	4	2	612	990	5
Extended [21]	12 months	6	2	1850	3984	4
Extended [21]	12 months	3	3	312	438	4
Ransomware	11 months	6	1	2855	7710	6
Ransomware	11 months	5	2	1619	3105	4
Celebrity	11 months	6	1	4461	12258	6
Email_EU [22]	18 months	6	4	667	5053	4
Reddit [23]	40 months	8	8	1060	1524	5

6 Experiments

We perform experiments with five online social media dataset and various w and s configurations. After showing prediction results, we will introduce a use case.

6.1 Experimental Environments

6.1.1 Datasets: We utilized five datasets. The first one is an extended version of the Twitter dataset used by Sabottke et al [21]. This dataset spans for 12 months. The second dataset is collected by us from Twitter about ransomware. At 2017, there were many ransomware attacks such as WannaCry and many people participated online discussion about them. The dataset lasts for 11 months. The third one, also collected by us, contains complete tweet and retweet history of online celebrities for 11 months. The fourth one is an email communication network from a large European research institution for 18 months, used by Leskovec et al [22]. The last dataset is about the interactions among various communities on Reddit. It has 40-month posts and was used by Kumar et al [23]. In Table I, each dataset is called *Extended*, *Ransomware*, *Celebrity*, *Email_EU*, and *Reddit*.

When splitting a dataset into windows, the time window size w and stride s need to be carefully selected to cover the total time span T. We test $w = 4, s = 2$, $w = 6, s = 2$ and $w = 3, s = 3$ for the Extended dataset; $w = 6, s = 1$ and $w = 5, s = 2$ for the Ransomware dataset; $w = 6, s = 1$ for the Celebrity dataset; $w = 6, s = 4$ for the Email_EU dataset; and $w = 8, s = 8$ for the Reddit dataset. The statistics on the eight experiment settings are summarized in Table I. After training, we predict the probabilities in the last time window. Note that the graph size varies depending on w and s configurations in a dataset (see Section 5.3).

6.1.2 Parameter Setup: For the aggregation function, we have three options: l-Max_Pooling, Mean_Pooling, and LSTM. For the combination function, we use the 1D convolutional layer. Other possible option is to use one linear (or fully-connected) layer. In our preliminary tests, the 1D convolution-based combination function outperforms in many cases the linear layer-based combination function so we choose the 1D convolution-based combination function (shown in Fig. 1). For the l-Max_Pooling, we use the average degree as l and the convolutional filter size of $\lfloor l/2 \rfloor + 1$.

We use the learning rate of 0.005. There are two possible training methods: i) training with whole graph and ii) training with subgraphs — for testing, we have to use whole graph. The subgraph training significantly reduces the training time and we found that in our datasets and task, the subgraph training is able to produce as good accuracy as the whole graph training in much lesser time. The size of subgraph is 20% of whole graph in terms of the number of vertices in our experiments. We train for 5,000 epochs. All experiments are done with GTX1080Ti, Tensorflow 1.11.0, and CUDA 10.0.

6.1.3 Baseline Methods: We compare our prediction method with the following baseline methods:

1) We execute basic and advanced regression methods, such as linear regression, Gaussian Process regression, Random Forest regression and so on. Only edge features are used to predict an influence probability from u to v. We do a parameter search[3] to find their best parameters.
2) We test state-of-the-art GCNs including LGCN, GraphSAGE, and GAT. But their official codes do not support edge features so we improve their designs to consider edge features. We use the initial embedding $\mathbf{h}_v^0$ shown in Eq. (1) to incorporate edge features and then do $\mathbf{h}_{(u,v)}^k = linear(\mathbf{h}_u^k \oplus \mathbf{h}_v^k)$ after k graph convolutional layers, followed by the same SA, TA, and MR activations. We call these modified GCNs as LGCN-edge, GraphSAGE-edge, and GAT-edge.
3) We also test the original LGCN, GraphSAGE, and GAT which consider only vertex features. To predict, we use the same method to apply the activation functions to $\mathbf{h}_{(u,v)}^k = linear(\mathbf{h}_u^k \oplus \mathbf{h}_v^k)$. However, these original models show limited performance in many cases so we show them for the Extended dataset only and omit in other experiments for space reasons.

6.1.4 Evaluation metrics: We use R^2, Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) to evaluate. Especially, the MAPE, defined as follows, is one of the strictest measures to evaluate the prediction models.

$$MAPE = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \frac{abs(p_e - p'_e)}{p_e}$$

6.2 Experimental Results

We tested all possible combinations of the parameter configurations in each dataset, but due to page limits we list only the best performing result for our method.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

TABLE II: Predication results on the Extended dataset with $w = 4, s = 2$. The up-arrow (resp. down-arrow) means higher (resp. lower) is better. The best results are indicated in boldface. We list only the top-2 among all base regression models for space reasons.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours (l-Max_Pooling, Sigmod, Whole Graph)	**0.998**	**0.002**	**0.055**
	Ours (l-Max_Pooling, Sigmod, Subgraph)	**0.998**	**0.002**	0.066
	Gradient Boosting Regression	0.808	0.040	0.724
	Gaussian Process Regression	0.767	0.044	1.023
	GAT-edge(l-Max_Pooling, Sigmoid)	0.781	0.049	2.891
	GraphSAGE-edge(l-Max_Pooling, Sigmoid)	0.694	0.064	1.170
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.746	0.045	0.785
	GAT	-0.019	0.095	6.539
	GraphSAGE	-0.014	0.093	6.139
	LGCN	0.007	0.097	6.622
JI	Ours (LSTM, Sigmod, Whole Graph)	**0.994**	**0.001**	**0.061**
	Ours (Mean_Pooling, Sigmod, Subgraph)	0.993	0.002	0.090
	Random Forest Regression	0.849	0.020	0.644
	Gradient Boosting Regression	0.823	0.021	0.705
	GAT-edge(l-Max_Pooling, Sigmoid)	0.739	0.027	3.036
	GraphSAGE-edge(l-Max_Pooling, Sigmoid)	0.602	0.032	1.561
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.761	0.030	1.427
	GAT	-0.005	0.052	4.721
	GraphSAGE	-0.004	0.051	4.498
	LGCN	-0.060	0.041	1.906
LT	Ours (l-Max_Pooling, Multi-logit, Whole Graph)	**0.999**	**0.002**	**0.017**
	Ours (l-Max_Pooling, Multi-logit, Subgraph)	**0.999**	0.003	0.032
	Random Forest Regression	0.942	0.055	0.344
	RANSAC Regression	0.923	0.065	0.354
	GAT-edge(l-Max_Pooling)	0.996	0.011	0.084
	GraphSAGE-edge(l-Max_Pooling)	0.867	0.087	0.744
	LGCN-edge (l-Max_Pooling)	0.912	0.070	0.640
	GAT	0.838	0.098	0.981
	GraphSAGE	0.838	0.098	0.980
	LGCN	0.847	0.096	0.949

6.2.1 Extended with $w = 4, s = 2$: For the Extended dataset with $w = 4, s = 2$, results are summarized in Table II. For the Bernoulli Trial (BT) probability definition, l-Max_Pooling and Sigmoid work very well. Both the whole graph and subgraph training show overwhelming performance compared to other baselines (e.g., the MAE of 0.002 in our method vs. 0.104 in LGCN).

For the Jaccard Index (JI) probability, the LSTM-based aggregation works very well for the whole graph training. For the subgraph training, Mean_Pooling is very stable. In both cases, Sigmoid produces the best results.

For the Linear Threshold (LT) probability definition, Our Multi-logit activation is the best. The combination of l-Max_Pooling and Multi-logit regression produce very stable predictions irrespective of whole graph and subgraph training.

In general, the graph convolutional networks perform very poor. All their R^2 scores are close to zero for the BT and JI probability which is considered as non-stable predictions. Although their performance for the LT probability seems acceptable, the predication results are close to each other, which indicates the good performance is due to the easy predication task for the LT probability. In the meantime, the performance of the modified graph convolutional networks (i.e., GAT-edge, GraphSAGE-edge, and LGCN-edge) improves a lot, but is still much worse than our method. This proves that our extension of graph convolutional networks to consider edge features and information cascade process is the key achieving the stable predictions.

TABLE III: Predication results on the Extended dataset with $w = 6, s = 2$. We list only the top-2 among all base regression models for space reasons.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours (l-Max_Pooling, Sigmod, Whole Graph)	**0.998**	**0.001**	**0.046**
	Ours (LSTM, Sigmod, Whole Graph)	**0.998**	**0.001**	0.060
	Ours (l-Max_Pooling, Sigmod, Subgraph)	**0.998**	**0.001**	0.054
	Random Forest Regression	0.834	0.028	1.015
	Gradient Boosting Regression	0.842	0.026	0.618
	GAT-edge(l-Max_Pooling, Sigmoid)	0.752	0.040	4.215
	GraphSAGE-edge (Mean_Pooling, Sigmoid)	0.595	0.045	1.053
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.648	0.041	0.941
	GAT	-2.4e-7	0.065	6.334
	GraphSAGE	-0.018	0.071	7.477
	LGCN	-0.042	0.073	7.122
JI	Ours (LSTM, Sigmod, Whole Graph)	**0.996**	**0.001**	0.064
	Ours (Mean_Pooling, Sigmod, Subgraph)	**0.996**	**0.001**	**0.058**
	Random Forest Regression	0.855	0.014	0.558
	Gradient Boosting Regression	0.880	0.013	0.559
	GAT-edge(l-Max_Pooling, Sigmoid)	0.332	0.032	4.513
	GraphSAGE-edge (Mean_Pooling, Sigmoid)	0.620	0.023	1.657
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.639	0.027	1.627
	GAT	-0.001	0.035	4.437
	GraphSAGE	-0.014	0.038	5.091
	LGCN	-0.158	0.031	1.014
LT	Ours (LSTM, Multi-logit, Whole Graph)	**0.999**	**8.2e-4**	**0.009**
	Ours (LSTM, Multi-logit, Subgraph)	**0.999**	0.002	0.019
	Gradient Boosting Regression	0.960	0.045	0.304
	Gaussian Process Regression	0.948	0.051	0.374
	GAT-edge(l-Max_Pooling)	0.997	0.012	0.103
	GraphSAGE-edge (Mean_Pooling)	0.885	0.079	0.786
	LGCN-edge (l-Max_Pooling)	0.914	0.067	0.578
	GAT	0.860	0.086	0.898
	GraphSAGE	0.861	0.086	0.898
	LGCN	0.866	0.085	0.887

6.2.2 Extended with $w = 6, s = 2$: For the Extended with $w = 6, s = 2$, we have similar results to the previous results as shown in Table III. All our predictions are much more stable than other baseline methods. One noticeable result is that for the JI probability, the subgraph training is better than the whole graph training. The training time of the whole graph and subgraph training is summarized in Table V. The subgraph training is up to 20 times faster than the whole graph training in this dataset.

6.2.3 Extended with $w = 3, s = 3$: In this configuration, the graph size is small so we test only the whole graph training. Our method also achieved much higher accuracy than other baseline methods in this experiment. All GCNs still perform unreasonably bad, so we omit their performance for the following datasets due to the page limits.

6.2.4 Ransomware with $w = 6, s = 1$: We test only the subgraph training for this dataset because the whole graph training sometimes takes one order of magnitude longer time as we saw in the previous Extended dataset experiments.

In this configuration, Mean_Pooling shows good performance in general, which is different from the Extended dataset results. We achieve the best MAPE score of 0.006 among all experiments for the LT probability definition by the combination of Mean_Pooling and Multi-logit prediction. Considering the difficult nature of our task and the strictness of the metric, the MAPE of 0.006 is a very outstanding prediction.

6.2.5 Ransomware with $w = 5, s = 2$: This configuration is more difficult than the previous configuration because the overlapping period of two consecutive time windows is smaller than that of the previous one.

TABLE IV: Predication results on the Extended dataset with $w = 3, s = 3$. Due to the small graph size, subgraph training is omitted for this setting. We list only the top-2 among all base regression models for space reasons.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours (Mean_Pooling, Sigmoid, Whole Graph)	**0.991**	**0.006**	**0.111**
	Ours (LSTM, Sigmod, Whole Graph)	0.982	0.008	0.140
	Random Forest Regression	0.659	0.052	1.077
	Gradient Boosting Regression	0.620	0.053	1.014
	GAT-edge (l-Max_Pooling, Sigmoid)	0.650	0.055	2.873
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.589	0.072	1.148
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.712	0.045	0.821
	GAT	-5.1e-06	0.084	4.946
	GraphSAGE	-0.025	0.096	6.095
	LGCN	-0.060	0.096	5.037
JI	Ours (Mean_Pooling, Sigmoid, Whole Graph)	**0.979**	**0.007**	**0.161**
	Random Forest Regression	0.730	0.026	0.958
	Gradient Boosting Regression	0.691	0.027	1.012
	GAT-edge (l-Max_Pooling, Sigmoid)	0.437	0.034	1.765
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.512	0.033	1.203
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.723	0.029	1.055
	GAT	-2.0e-5	0.048	3.967
	GraphSAGE	-0.010	0.053	4.691
	LGCN	-0.100	0.056	3.580
LT	Ours (Mean_Pooling, Multi-logit, Whole Graph)	**1.000**	**0.003**	**0.012**
	Ours (LSTM, Multi-logit, Whole Graph)	**1.000**	**0.003**	0.014
	Random Forest Regression	0.897	0.072	0.387
	Gradient Boosting Regression	0.892	0.073	0.386
	GAT-edge (l-Max_Pooling)	0.995	0.014	0.091
	GraphSAGE-edge (l-Max_Pooling)	0.841	0.090	0.574
	LGCN-edge (l-Max_Pooling)	0.856	0.085	0.562
	GAT	0.833	0.097	0.728
	GraphSAGE	0.832	0.097	0.728
	LGCN	0.837	0.096	0.727

TABLE V: Training time (in sec.) of the whole graph training and subgraph training for Extended, 5000 epochs.

	$w = 4, s = 2$		$w = 6, s = 2$	
	Whole	Sub	Whole	Sub
Mean_Pooling	569.68	168.31	7013.57	406.87
LSTM	983.28	613.81	7412.33	635.66
l-max_pooling	693.37	241.63	11881.55	591.14

Many baseline methods do not provide good predictions whereas our prediction accuracy is almost the same as (or slightly less than) the previous $w = 6, s = 1$ configuration. This proves that our method is more robust than others.

6.2.6 Celebrity with $w = 6, s = 1$: Our method shows the best performance again in this dataset. Mean_Pooling works well. Some other baselines also show reasonable performance for R^2 and MAE. However, their MAPE is not as good as ours. Our MAPE is 2-3 times better than that of the best baseline methods.

6.2.7 Email_EU with $w = 6, s = 4$: For this configuration, we use subgraph training. Our method with Mean_Pooling shows good performance for all probability definition (BT, JI, and LT). Regression baselines also show reasonable performance, but the modified GCN methods except GAT-edge in the LT probability definition are much worse, especially for MAPE.

6.2.8 Reddit with $w = 8, s = 8$: This configuration is more challenging due to the equal size of window and stride. The performance of many baselines deteriorates for the BT and JI probability definitions, whereas our methods still show robust performance for all probability definitions.

TABLE VI: Predication results on the Ransomware dataset with $w = 6, s = 1$. We list only the top-2 among all base regression models for space reasons. In this table and hereinafter, we test only the subgraph training for our method due to the excessively long training time of the whole graph training and exclude GAT, GraphSAGE, and LGCN for their poor performance.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours (LSTM, Sigmoid)	0.995	**0.001**	**0.082**
	Ours (l-Max_Pooling, Sigmoid)	**0.996**	**0.001**	0.088
	Gradient Boosting Regression	0.866	0.018	0.568
	Gaussian Process Regression	0.853	0.019	0.900
	GAT-edge(l-Max_Pooling, Sigmoid)	0.788	0.028	4.916
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.454	0.040	1.628
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.471	0.042	1.760
JI	Ours (Mean_Pooling, Sigmoid)	**0.992**	**6.9e-4**	**0.064**
	Ours (LSTM, Sigmoid)	0.984	9.9e-4	0.094
	Random Forest Regression	0.854	0.009	0.541
	Gradient Boosting Regression	0.864	0.009	0.564
	GAT-edge(l-Max_Pooling, Sigmoid)	0.417	0.032	1.480
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.510	0.019	1.575
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.499	0.041	1.720
LT	Ours (Mean_Pooling, Multi-logit)	**0.999**	**6.1e-4**	**0.006**
	Ours (LSTM, Multi-logit)	**0.999**	0.001	0.013
	Ours (l-Max_Pooling, Multi-logit)	**0.999**	0.001	0.014
	Random Forest Regression	0.963	0.032	0.215
	Gradient Boosting Regression	0.963	0.032	0.230
	GAT-edge(l-Max_Pooling)	0.996	0.010	0.097
	GraphSAGE-edge (l-Max_Pooling)	0.841	0.081	0.973
	LGCN-edge (l-Max_Pooling)	0.812	0.074	0.975

TABLE VII: Predication results on the Ransomware dataset with $w = 5, s = 2$. We list only the top-2 among all base regression models for space reasons.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours (Mean_Pooling, Sigmoid)	**0.994**	0.003	0.147
	Ours (LSTM, Sigmoid)	0.989	**0.002**	**0.124**
	Random Forest Regression	0.649	0.039	1.253
	RANSAC Regression	0.493	0.042	1.173
	GAT-edge(l-Max_Pooling, Sigmoid)	0.657	0.038	4.934
	GraphSAGE-edge(l-Max_Pooling, Sigmoid)	0.341	0.048	1.383
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.476	0.049	1.852
JI	Ours (Mean_Pooling, Sigmoid)	**0.947**	**0.002**	**0.162**
	Ours (LSTM, Sigmoid)	0.929	0.004	0.254
	Random Forest Regression	0.620	0.019	1.309
	RANSAC Regression	0.688	0.016	1.121
	GAT-edge(l-Max_Pooling, Sigmoid)	0.452	0.313	2.471
	GraphSAGE-edge (k-l-Max_Pooling, Sigmoid)	0.376	0.021	1.028
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.479	0.050	1.957
LT	Ours (Mean_Pooling, Multi-logit)	**0.999**	**0.003**	0.054
	Ours (l-Max_Pooling, Multi-logit)	**0.999**	**0.003**	**0.034**
	Gradient Boosting Regression	0.908	0.066	0.488
	RANSAC Regression	0.903	0.070	0.441
	GAT-edge(l-Max_Pooling)	0.996	0.012	0.101
	GraphSAGE-edge (l-Max_Pooling)	0.824	0.099	1.001
	LGCN-edge (l-Max_Pooling)	0.823	0.094	1.037

6.3 Remarks on Experimental Results

One apparent observation is that the larger the stride s, the harder the prediction. However, our model is very robust against all the large s configurations. In fact, the stride s depends on the capability of actors performing influence maximization. Some actors may be able to frequently collect updated retweet logs and recalculate influence probabilities whereas many others cannot. Recall that collecting data from social media such as Twitter costs non-trivial expenses. In any case, our model can provide very reliable predictions.

Our multi-logit regression works very well for the LT influence probability definition even with the subgraph training (by the characteristic we proved in Theorem 5.1).

TABLE VIII: Predication results on the Celebrity dataset with $w = 6, s = 1$. We list only the top-2 among all base regression models for space reasons.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours (Mean_Pooling, Sigmoid)	0.984	0.003	**0.273**
	Ours (LSTM, Sigmoid)	**0.996**	**0.002**	0.291
	Random Forest Regression	0.977	0.005	0.74
	Gradient Boosting Regression	0.978	0.005	0.882
	GAT-edge(l-Max_Pooling, Sigmoid)	0.916	0.021	5.381
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.869	0.013	1.714
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.603	0.025	1.124
JI	Ours (Mean_Pooling, Sigmoid)	**0.995**	**2.1e-6**	**0.068**
	Ours (LSTM, Sigmoid)	0.968	3.5e-5	0.175
	Linear Regression	0.940	4.5e-4	0.472
	Gradient Boosting Regression	0.940	4.3e-4	0.327
	GAT-edge(l-Max_Pooling, Sigmoid)	0.443	0.090	2.194
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.497	0.034	3.865
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.561	0.027	1.202
LT	Ours (Mean_Pooling, Multi-logit)	0.996	**3.7e-4**	0.159
	Ours (LSTM, Multi-logit)	**0.997**	0.002	**0.124**
	Random Forest Regression	0.966	0.036	0.216
	Gradient Boosting Regression	0.966	0.037	0.637
	GAT-edge(l-Max_Pooling)	0.897	0.101	1.070
	GraphSAGE-edge (l-Max_Pooling)	0.692	0.126	1.284
	LGCN-edge (l-Max_Pooling)	0.749	0.103	1.167

TABLE IX: Predication results on the Email_EU dataset with $w = 6, s = 4$. We list only the top-2 among all base regression models for space reasons.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours(Mean_Pooling, Sigmoid)	**0.952**	0.003	0.058
	Ours(LSTM, Sigmoid)	0.944	**0.002**	**0.050**
	Gradient Boosting Regression	0.911	0.006	0.450
	Random Forest Regression	0.904	0.003	0.067
	GAT-edge(l-Max_Pooling, Sigmoid)	0.743	0.020	1.344
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.524	0.026	2.121
	LGCN-edge(l-Max_Pooling, Sigmoid)	0.712	0.045	2.580
JI	Ours(l-Max_Pooling, Sigmoid)	**0.937**	0.026	0.309
	Ours(Mean_Pooling, Sigmoid)	0.926	0.028	0.402
	Random Forest Regression	0.909	**0.002**	**0.130**
	Gradient Boosting Regression	0.896	0.003	0.415
	GAT-edge(l-Max_Pooling, Sigmoid)	0.530	0.018	1.863
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.587	0.035	2.278
	LGCN-edge(l-Max_Pooling, Sigmoid)	0.745	0.033	1.597
LT	Ours(LSTM, Sigmoid)	**0.995**	**0.006**	**0.199**
	Ours(Mean_Pooling, Sigmoid)	0.994	0.007	0.203
	Gradient Boosting Regression	0.926	0.028	0.402
	Random Forest Regression	0.918	0.028	0.351
	GAT-edge(l-Max_Pooling)	0.985	0.011	0.299
	GraphSAGE-edge (l-Max_Pooling)	0.688	0.074	2.667
	LGCN-edge(l-Max_Pooling)	0.656	0.076	2.590

TABLE X: Predication results on the Reddit dataset with $w = 8, s = 8$. We list only the top-2 among all base regression models for space reasons.

	Method	$R^2 \uparrow$	$MAE \downarrow$	$MAPE \downarrow$
BT	Ours (LSTM, Sigmoid)	**0.986**	0.013	0.333
	Ours(Mean_Pooling, Sigmoid)	0.968	**0.010**	**0.057**
	Random Forest Regression	0.758	0.052	2.440
	Gradient Boosting Regression	0.637	0.071	3.664
	GAT-edge(l-Max_Pooling, Sigmoid)	0.616	0.080	4.755
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.456	0.087	2.820
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.714	0.060	0.851
JI	Ours(Mean_Pooling, Sigmoid)	**0.926**	**0.065**	0.741
	Ours (l-Max_Pooling, Sigmoid)	**0.926**	0.066	0.783
	Gradient Boosting Regression	0.881	0.009	0.581
	Random Forest Regression	0.834	0.007	**0.144**
	GAT-edge(l-Max_Pooling, Sigmoid)	0.466	0.030	5.307
	GraphSAGE-edge (l-Max_Pooling, Sigmoid)	0.559	0.023	1.276
	LGCN-edge (l-Max_Pooling, Sigmoid)	0.750	0.015	0.964
LT	Ours(Mean_Pooling, Sigmoid)	**1.000**	**0.001**	**0.006**
	Ours (l-Max_Pooling, Sigmoid)	**1.000**	**0.001**	0.008
	Gradient Boosting Regression	0.959	0.046	0.617
	Random Forest Regression	0.950	0.047	0.406
	GAT-edge(l-Max_Pooling)	0.998	0.009	0.096
	GraphSAGE-edge (l-Max_Pooling)	0.906	0.068	0.829
	LGCN-edge (l-Max_Pooling)	0.941	0.055	0.727

6.4 Use Case on Influence Maximization

In this subsection, we introduce a couple of use cases with the Extended and Reddit datasets due to the space limitation and the excessive runtime of influence maximization. Let $\sigma(S, \mathcal{G})$, where S is a set of seeds, be an influence spread value of S (i.e., the number of people influenced by S) in $\mathcal{G}$. We use the BCT algorithm [24] that performs Monte Carlo simulations to calculate $\sigma(S, \mathcal{G})$. We compare the following methods to find S:

1) Ours: $\arg\max_S \sigma(S, \mathcal{G}_{predicted})$, where $\mathcal{G}_{predicted}$ has probabilities predicted by us.
2) Robust [7]: $\arg\max_S min_i(\frac{\sigma(S,\mathcal{G}_i)}{\sigma(S_i^*,\mathcal{G}_i)})$, where $\mathcal{G}_i$ has the probabilities at i-th training window and $S_i^* = \arg\max_S \sigma(S, \mathcal{G}_i)$.

In particular, the second method is called *robust influence maximization*. It chooses a set of seeds whose worst-case performance is maximized. Thus, S_{robust} works well during the training period. It had been shown that S_{robust} is very reliable in [7]. We implemented their algorithm to find S_{robust}.

After finding the seeds, we evaluate them in the testing set, i.e., $\sigma(S, \mathcal{G}_{test})$. We expect $\sigma(S_{ours}, \mathcal{G}_{test}) \approx \sigma(S^*, \mathcal{G}_{test})$, where $S^* = \arg\max_S \sigma(S, \mathcal{G}_{test})$, i.e., the optimal seed set of $\mathcal{G}_{test}$. We perform 10,000 times Monte Carlo simulations on the testing set to obtain reliable influence spread values by S^*, S_{ours}, and S_{robust}.

Fig. 3 shows three spread examples by 5 seeds on the Extended dataset with $w = 3, s = 3$. Both vertex size and color represent the probability that a vertex is influenced by seeds during Monte Carlo simulations. A large vertex in red means that it had been consistently influenced by seeds during the simulations. In the figure, our spread visualization (in the middle) is very close to the optimal (in the left). This result shows the efficacy of our prediction-driven influence maximization — about 26 people were influenced by S^* and S_{ours} in average during the Monte Carlo simulations vs. 23 people by S_{robust} (i.e., 88% of our score). In the Reddit dataset with $w = 8, s = 8$, S^* and S_{ours} infected 21 people and S_{robust} infected 19 people. These examples show two representative cases where our prediction-driven influence maximization is very close to the optimal. In many cases, our method is as good as the optimal case. Compared to other baseline prediction models, such as LGCN-edge, GraphSAGE-edge, GAT-edge, and other regression models, our method shows up to 20% higher spread scores.

7 Conclusion

We presented our influence probability prediction model based on various graph convolutional layers and final activation layers. Our prediction model outperforms other baseline prediction models in almost all cases with small training overheads. To reduce training overheads, we rely on subgraph training which feeds partial neighbor information in a mini-batch. However, we prove that our proposed method can be properly trained only with partial neighbor information. The

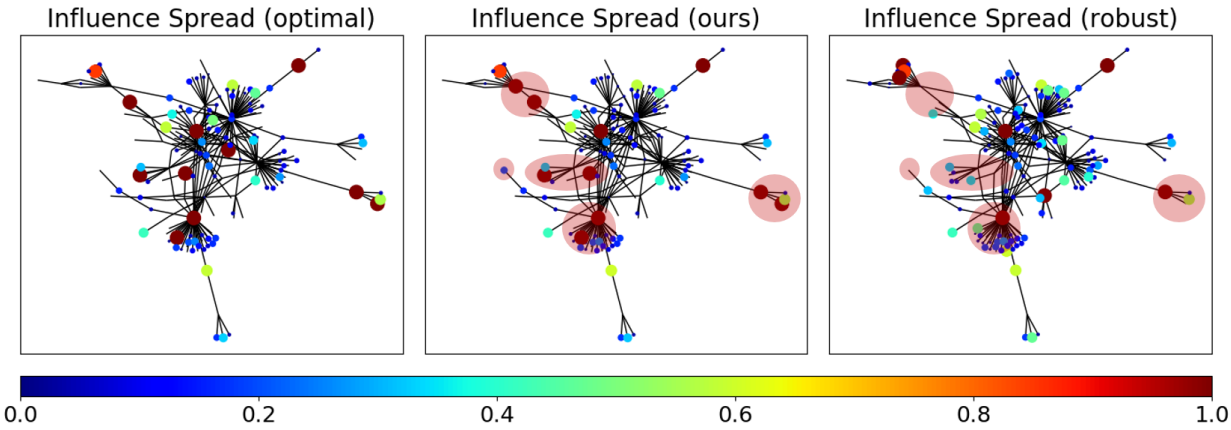


Fig. 3: Influence spread by 5 seeds in the last test window of Extended, $w = 3$, and $s = 3$ with the Independent Cascade model and the Bernoulli Trial-based influence probability. We compare three methods: an optimal solution directly calculated on the last window, our prediction-driven influence maximization, and a robust influence maximization [7]. Both vertex size and color represent the probability of being infected from seeds. About 26 people are infected in the optimal and our cases whereas 23 people are infected in the robust influence maximization case.

influence maximization based on our predictions shows better performance (i.e., more infected people) than other methods including the robust influence maximization with non-trivial margins, i.e., 10-20% higher spread scores.

As the influence maximization is used in many real-world applications, we believe that our prediction model and the prediction-driven influence maximization will benefit many people.

REFERENCES

[1] A. Yadav, H. Chan, A. Xin Jiang, H. Xu, E. Rice, and M. Tambe, "Using social networks to aid homeless shelters: Dynamic influence maximization under uncertainty," in *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, 2016.

[2] Y. Chang, C. Zhai, Y. Liu, and Y. Maarek, Eds., *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018.

[3] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, 2010.

[4] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[5] C. Lagnier, L. Denoyer, E. Gaussier, and P. Gallinari, "Predicting information diffusion in social networks using content and user's profiles," in *Proceedings of the 35th European Conference on Advances in Information Retrieval*, 2013.

[6] K. Saito, R. Nakano, and M. Kimura, "Prediction of information diffusion probabilities for independent cascade model," in *Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part III*, 2008.

[7] X. He and D. Kempe, "Robust influence maximization," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[8] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Rev.*, vol. 53, no. 3, Aug. 2011.

[9] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019.

[10] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling Relational Data with Graph Convolutional Networks," *arXiv e-prints*, 2017.

[11] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, 2018.

[12] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018.

[13] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017.

[14] A. Goyal, W. Lu, and L. V. Lakshmanan, "Celf++: Optimizing the greedy algorithm for influence maximization in social networks," in *Proceedings of the 20th International Conference Companion on World Wide Web*, 2011.

[15] C. Zhou, P. Zhang, J. Guo, X. Zhu, and L. Guo, "Ublf: An upper bound based approach to discover influential nodes in social networks," in *2013 IEEE 13th International Conference on Data Mining*, Dec 2013.

[16] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009.

[17] Q. Liu, B. Xiang, E. Chen, H. Xiong, F. Tang, and J. X. Yu, "Influence maximization over large-scale social networks: A bounded linear approach," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014.

[18] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*, 2012.

[19] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, pp. 1852–1872, 2018.

[20] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment strength detection in short informal text," *J. Am. Soc. Inf. Sci. Technol.*, vol. 61, no. 12, pp. 2544–2558, Dec. 2010.

[21] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *Proceedings of the 24th USENIX Security Symposium*, 2015.

[22] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, p. 2, 2007.

[23] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky, "Community interaction and conflict on the web," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18, 2018.

[24] H. T. Nguyen, T. N. Dinh, and M. T. Thai, "Cost-aware targeted viral marketing in billion-scale networks," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.