REFINE: Representation Learning from Diffusion Events

Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor

University of Trento, Trento, Italy
{zekarias.kefato,nasrullah.sheikh,alberto.montresor}@unitn.it

Abstract. Network representation learning has recently attracted considerable interest, because of its effectiveness in performing important network analysis tasks such as link prediction and node classification. However, most of the existing studies rely on the knowledge of the complete network structure. Very often this is not the case, unfortunately: the network is either partially or completely hidden. For example, due to privacy and competitive market advantage, the friendship and follower networks of Facebook and Twitter are hardly accessible. User activity logs (also known as cascades), instead, are usually available. In this study we propose Refine, a representation learning algorithm that does not require information about the network and simply utilizes cascades. Nodes embeddings learned through Refine are optimized for network reconstruction. Towards this end, it utilizes the global interaction patterns exposed by reaction times and co-occurrences. We present an extensive experimentation using two OSN datasets and show that our approach outperforms existing baselines. In addition, we empirically show that Refine can be used to predict cascades as well.

Keywords: Network Inference, Representation Learning, Cascade Prediction

1 Introduction

Network representation learning (NRL) has recently attracted considerable research attention. In particular, the ubiquitous success of deep learning has inspired social network scientists to exploit neural networks to automatically learn representation of nodes, that could later be used for several social analysis tasks. A number of existing studies have assumed that the network structure is completely known. Very often, however, this is not the case; instead, information about the network is either partial or completely absent. For instance, companies seeking a marketing campaign through Facebook or Twitter desire access to the structural properties of the social graph; such information, however, is usually not accessible due to privacy and competitive market advantage [1].

Some information is available, though. For example, extensive logs of events occurring on the social graph can be easily obtained, e.g. through public APIs. These logs represent the propagation of information over the latent network, for

example by recording the instant in which a user shares a meme or a piece of fake news. The process of propagation is known as a *cascade*; it is usually triggered by a few sources (*seeds*) and spreads over the graph through its edges [2–4].

In other words, we can observe who shares a meme and when this happens, but not the edge through which the meme has been transported. The goal of this study is to learn a representation of nodes optimized for reconstructing the latent network by simply using the cascades.

Related Work. Several studies [2–7] have been proposed towards the network reconstruction task. In general, we can divide them into two broad categories, which are (i) delay-aware and (ii) delay-agnostic. Some of the existing delay-aware models, such as NetInf [2], NetRate [6], InfoPath [5], and Kernel-Cascade [3], exploit infection rates based on delay patterns between infection timestamps. The main assumption is that if a pair of nodes tend to get infected right after each other, then there is a diffusion pattern that is a likely indicator of connections. Some of them [5, 6] assume a fixed parametric form (e.g exponential) of influence model or transmission rate on the edges of the network. Nonetheless, a particular study [3] has argued and empirically demonstrated that such an assumption is too strong for capturing the complex diffusion patterns and user infection dynamics in real networks.

On the other hand, some studies [4, 7] follow a delay-agnostic approach simply based on the order and/or context of infection events. Furthermore, they have argued that delay-aware models are likely to miss out several diffusion patterns, even in the presence of recurring ones, because of the delay intervals of such models that could potentially be too large or too small. This problem is normally caused by explicitly pre-defined infection rates (delay patterns) and fixed parametric forms of influence models, as argued by [3].

In the area of network representation learning, there are also quite a number of studies [8–14]. The algorithms vary from classical techniques that rely on matrix factorization to recent techniques using deep neural networks. Their goal is usually to embed nodes of the network in a low-dimensional latent space in such a way that the embedding preserves different properties of the network, for example local neighborhoods. Our work is essentially different from the above techniques, because we lack the knowledge of the network structure.

Current Work. In this study, we propose Refine, an delay-aware algorithm for network reconstruction based on representation learning. Contrary to [4, 7], we argue that delay-aware models can also perform as well as delay-agnostic models if they are properly designed. Therefore, Refine utilizes the delays between infection events; unlike some of the existing methods [2, 5, 6], however, it avoids any assumption regarding the influence model and infections rates. Instead, it directly embeds users according to the inherent interaction patterns exposed by them.

Refine is established on the premise that closely connected users, for example members of a community, expose interaction patterns that are expressed by reaction time and frequency. In terms of reaction time, given a post by a certain

member of a community, it is very likely for another member to share the post faster than non-members. In terms of frequency, it is more likely for a member of a community to co-occur with another member in cascades more frequently than with other non-members. REFINE learns a low-dimensional embedding of nodes that capture such interaction patterns and use the learned embedding to estimate pairwise edge probabilities towards reconstructing the network.

We have performed extended evaluations of our approach and compared it against strong baselines. Besides utilizing the embedding to reconstruct the latent network, we have also evaluated the capability of our representation learning approach to predict the cascades themselves.

The rest of the paper is organized as follows. Section 2 introduces the notation which is used in the rest of the paper. Section 3 describes the REFINE algorithm. Sections 4 presents the results of our experiments and we conclude the paper in Section 5.

2 Model and problem definition

We assume that cascades occur over a *hidden* graph $H = (U, E)$, where U is a set containing n vertexes, each vertex corresponding to a user, and E is a set containing m edges (connections) between users. We will use the term vertex and user as synonyms, preferring the former when referring to human being, and the latter when referring to graph-theoretic concepts. Interactions between users occur over the network; while the set of users is normally well-defined, the set of connections among them can be partially or completely unknown.

The spread of multiple contagions across the network H generates a collection $\mathcal{C}$ of cascades. A contagion can be considered as any piece of online content, such as, a tweet, meme, video, that spread through online networks as a result of resharing activities. A cascade $C \in \mathcal{C}$ is a sequence that captures both the order and the time instant in which users have been infected by a given contagion. More formally, it is defined as: $C = [(u_1, t_1), (u_2, t_2), \ldots, (u_c, t_c)]$ where t_i is the timestamp associated to user u_i. We assume that $i < j \Rightarrow t_i \leq t_j$.

We use $C(i)$ to denote the i-th user of C; and $C_t(i)$ to denote the corresponding timestamp. We also use $\mathcal{C}_u \subseteq \mathcal{C}$ to denote the subset of all cascades that user u is involved in $\mathcal{C}_u = \{C : \exists i \wedge 1 \leq i \leq |C| \wedge C(i) = u\}$ with $\mathcal{C}_u \neq \emptyset$, meaning that all users in U have been involved in at least one cascade.

Given a cascade C, we define a function $r_C : U \times U \to \mathbb{R}^+$ measuring the reaction time between the infection events of u and v, if both have been infected in C, or ∞ otherwise:

$$r_C(u, v) = \begin{cases} |C_t(i) - C_t(j)| & \exists i, j : u = C(i) \wedge v = C(j) \\ +\infty & \text{otherwise} \end{cases}$$

In addition, we define the *co-infection frequency* function $f(u, v) = |\mathcal{C}_u \cap \mathcal{C}_v|$ that computes the number of cascades that involve both u and v.

The problem we want to solve is the following: given a set of observed cascades $\mathcal{C}$ over a hidden network $H = (U, E)$, we want to infer a network $G = (U, E')$ such that E' approximates E as much as possible.

To evaluate the performance of our algorithms, similar to [8] we use the *precision-at-K* ($P@K$) metric. Our approach will produce an edge probability for every pair of vertexes; we can thus rank pairs of vertexes according to such probability. We cut this rank at different thresholds K and we compute the precision on the top-K pairs, i.e. the fraction of those pairs that are true edges on the ground-truth network.

3 The REFINE Algorithm

REFINE considers global interaction patterns expressed through users reaction time and co-occurrences in cascades. For a given user $u \in U$, REFINE computes (i) a reaction time summary between the infection time of u and all other users and (ii) the relative co-occurrence frequency between u and all other users, both measured over the entire collection $\mathcal{C}$. Our assumption is that if two users u and v exhibit a strong interaction pattern, then they are likely to be connected.

A straightforward approach towards reconstruction is to compute similarity between users according to their global interaction representation. However, this leads to poor performances as this representation is very sparse. Rather, we first learn an embedding of users in such a way that their interaction patterns in the input representation space is preserved. Finally, we estimate the pairwise edge probabilities between every pair of nodes to reconstruct the latent network.

3.1 Interaction Pattern Summarization

REFINE is a delay-aware model based on the global interaction delays (reaction-time) and frequency (co-occurrence) in cascades. We start by computing a reaction time distribution for each user. Given a cascade $C \in \mathcal{C}$ and a user u appearing in C (e.g., $\exists i : u = C(i)$), we compute, for the sake of numerical convenience, an inverted reaction time function $r_C^{-1}(u, v)$ defined as follows:

$$r_C^{-1}(u,v) = \begin{cases} 0 & r_C(u,v) = \infty \\ 1 & r_C(u,v) = 0 \\ e^{-r_C(u,v)} & \text{otherwise} \end{cases} \tag{1}$$

$r_C^{-1}(u, v)$ is a well-defined function from pairs of nodes to $[0, 1]$, given that $r_C^{-1}(u, v)$ approaches 0 when $r_C(u, v)$ grows to infinity, and $r_C^{-1}(u, v)$ approaches 1 when $r_C(u, v)$ tends to 0.

REFINE utilizes the function r_C^{-1} to compute an (inverted) *reaction time summary vector* $\boldsymbol{R}'(u)$ for each user $u \in U$, aggregated over all cascades $\mathcal{C}$, where each entry $\boldsymbol{R}'(u)[v]$, $v \in U$, is defined as follows:

$$\boldsymbol{R}'(u)[v] = \frac{\sum_{C \in \mathcal{C}_u \cap \mathcal{C}_v} r_C^{-1}(u,v)}{\sum_{C \in \mathcal{C}_u} \sum_{i=1}^{|C|} r_C^{-1}(u, C(i))} \tag{2}$$

Equation 2 computes the (inverted) average reaction time between u and v, normalized over all the cascades pertinent to u, $\mathcal{C}_u$.

One can easily notice that the reaction time summary vector $\boldsymbol{R}'(u)$ captures a reaction time distribution for each user u. Nonetheless, it fails to account for the co-infection frequency between u and every other node v, which we consider to be another strong signal for the existence of an edge between u and v. For example, let v and w be two nodes with equal values in their respective entries in the reaction time summary vector of u, i.e. $\boldsymbol{R}'(u)[v] = \boldsymbol{R}'(u)[w]$. If $f(u,v) \gg f(u,w)$, it is obvious that u and v have a stronger interaction tendency than u and w, which is not modeled by $\boldsymbol{R}'$.

To compensate for that, we first compute the relative co-infection frequency vector $\boldsymbol{F}(u)$, where each $\boldsymbol{F}(u)[v]$, $v \in U$, is defined as follows:

$$\boldsymbol{F}(u)[v] = \frac{f(u,v)}{\sum_{w \in U} f(u,w)} \tag{3}$$

Finally, we combine $\boldsymbol{R}'$ and $\boldsymbol{F}$ to obtain the *interaction pattern summary* $\boldsymbol{I}(u) = \boldsymbol{F}(u) \times \boldsymbol{R}'(u)$ for each user u. The vectors $\boldsymbol{I}(u)$ can be summarized in a matrix $\boldsymbol{I} = [\boldsymbol{I}(u_1), \ldots, \boldsymbol{I}(u_n)] \in [0,1]^{n \times n}$ that contains a row for each user.

Now, even though two users v and w have a tie for u in terms of $\boldsymbol{R}'(u)$, *i.e,* $\boldsymbol{R}'(u)[v] = \boldsymbol{R}'(u)[w]$, $\boldsymbol{F}(u)$ breaks such tie by putting more weight on the user with a stronger co-infection frequency with u.

A naïve approach towards reconstructing the hidden network could be to compute the similarity between each pair of users u, v based on $\boldsymbol{I}(u)$ and $\boldsymbol{I}(v)$, for example by computing their distance over $[0,1]^n$. This approach, however, leads to a poor performance as $\boldsymbol{I}$ is very sparse. We apply instead a learning phase to embed $\boldsymbol{I}$ in a low and dense latent embedding space, in such a way that the the patterns encoded in $\boldsymbol{I}$ are preserved. In other words, we intend to identify a mapping function $\Phi : [0,1]^{n \times n} \rightarrow \mathbb{R}^{n \times d}$, with $d \ll n$.

Finally, we utilize Φ to effectively learn the probability for an edge between a pair of nodes to exist, in order to reconstruct the hidden network.

3.2 User Embedding

The hidden network structure that we seek to reconstruct lives in a highly non-linear space [8]. Therefore, one has to identify a mapping $\Phi \in \mathbb{R}^{n \times d}$ that enables her to recover the non-linear network structure. Towards this goal, REFINE uses a deep *autoencoder*, an unsupervised neural network model.

An autoencoder enables us to embed $\boldsymbol{I}$ in a low-dimensional latent space by composing several non-linear functions (layers), as shown in Fig. 1. The input is given by the matrix $\boldsymbol{I}$. The *user embedding* module of Fig. 1 has two components, the *encoder* (blue layers) and the *decoder* (black layers). The former trans-

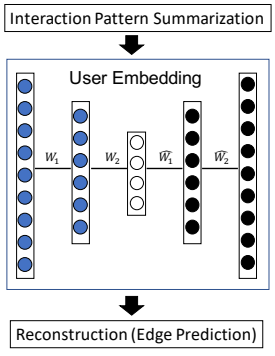

Fig. 1. The REFINE framework

forms the input into an *embedding* (white layer), while
the latter tries to regenerate and output the original input from the embedding.

Formally, the encoder $\mathcal{E} : [0,1]^{n\times n} \rightarrow \mathbb{R}^{n\times d}$ and the decoder $\mathcal{D} : \mathbb{R}^{n\times d} \rightarrow [0,1]^{n\times n}$ are a composition of non-linear functions defined as follows:

$$\mathcal{E}(\boldsymbol{I}) = e_l(\ldots e_\ell(\ldots(e_1(\boldsymbol{I}\cdot W_1)\cdot W_2)\ldots)\ldots) = \Phi \tag{4}$$

$$\mathcal{D}(\Phi) = d_l(\ldots d_\ell(\ldots(d_1(\Phi\cdot \widehat{W_1})\cdot \widehat{W_2})\ldots)\ldots) = \tilde{\boldsymbol{I}} \tag{5}$$

where e_ℓ and d_ℓ are the non-linear functions (e.g., *relu*, *tanh*) of the $\ell-th$ encoder and decoder layers, respectively. Each layer of an autoencoder is fully connected, meaning that it is a linear transformation of the output of the previous layer $\ell-1$, *i.e.* $f_{\ell-1}(\cdot)\cdot W_\ell$, and f_ℓ is either e_ℓ or d_ℓ.

Optimization: The weights are the main parameters of the model that needs to be trained. Normally this is achieved by minimizing the cost function of Eq. 6.

$$L = \arg\min_{W} \parallel \boldsymbol{I} - \tilde{\boldsymbol{I}} \parallel_F^2 \tag{6}$$

where $\boldsymbol{I}$ is the input matrix and $\tilde{\boldsymbol{I}}$ is the regenerated output matrix. The mere optimization of Eq. 6 leads to a poor performance due to $\boldsymbol{I}$'s sparsity. To deal with this, we adopt Wang's strategy [8] and reformulate Eq. 6 as

$$L = \arg\min_{W,\widehat{W}} \parallel (\boldsymbol{I} - \tilde{\boldsymbol{I}}) \oplus S \parallel_F^2 + \lambda\xi \tag{7}$$

where $\oplus$ is the Hadamard product and $S \in \mathbb{R}_+^{n\times n}$ a term to avoid the sparsity problem, is associated with $\boldsymbol{I}$, *i.e* if $\boldsymbol{I}(u,v) = 0$, then $S(u,v) = 1$ otherwise $S(u,v) = \mu > 1$ and μ is an alias for $S(u,v)$. The second term in Eq. 7, $\xi = \sum_{\ell=1}^{l} \parallel W_\ell \parallel_F^2 + \parallel \widehat{W_\ell} \parallel_F^2$, is a regularization term to avoid over-fitting and $\lambda \in (0,1)$ is the regularization constant.

Finally, Eq. 7 can be optimized using classical algorithms such as gradient descent. Then, once the optimization is solved, we obtain an embedding $\Phi(u)$ of each user $u \in U$.

Speeding-up the user embedding For a very large value of n, training an autoencoder using $\boldsymbol{I}$ could be very expensive. Thus, we propose an intermediate step of dimensionality reduction using truncated (partial) singular value decomposition (T-SVD) for very large matrices [15]. T-SVD utilizes a few of the highest or smallest eigenvalues of a large matrix. As a result, we can efficiently reduce $\boldsymbol{I}$'s dimension and feed the reduced $\boldsymbol{I}_r$ to the autoencoder. Moreover, this can be considered as an alternative solution to tackle the sparsity problem with $\boldsymbol{I}$. Note that when employing this component there is no need for the sparsity term in the loss function of Eq. 7. We have observed that including this optimization provides similar or better results, with a significant reduction in memory and computational time.

3.3 Reconstruction

Once Φ is computed as in Section 3.2, we exploit it to predict the probability that an edge exists between a pair of users. We assume that if a pair of users never co-occur in any cascade, they have a very small chance of being connected. Therefore, we discard such pairs and analyze the remaining ones.

Let $p(u,v) = 1/(1+e^{-(\Phi(u)^T \cdot \Phi(v))})$ be a function that predicts the probability that an edge exists between u and v. We build a network $G = (U, E'), E' \approx E$ by adding an edge (u,v) to E' with probability $p(u,v)$. E' can be refined by pruning edges (u,v) where $p(u,v) < \tau$ for some threshold τ.

4 Experiments and Results

Dataset Description. Our experiments are performed on the following datasets, whose characteristics are summarized in Table 1.

Twitter [16] contains a set of Twitter users with a reciprocal follower relationships, collected from March 24th to April 25th, 2012. The follower network is considered as a ground truth. Two kinds of cascades are present: (1) Hashtag (HT): Cascades collected from user activity when using/adopting hashtags; (2) Retweet (RT): Cascades collected from user retweeting tweets.

MemeTracker (MT) [5], contains users represented by a collection of news media and blog sites. Cascades are formed based on the spread of memes. A contagion occurs when a particular meme is used by a site for the first time. The sequence of all the infected sites form a cascade. The ground truth network is built based on hyper-links found in each site.

Settings. In order to tune the hyper-parameters of REFINE, we use the random grid search strategy; its weights are initialized according to [17] for uniform distribution. To implement our models, we adopted the TENSORFLOW[1] and SCIPY[2] Python-based libraries. In all the experiments, both the encoder and decoder of REFINE use the *tanh* activation function.

Results. In the first set of experiments, we have compared REFINE with two strong baselines INFOPATH [5] (delay-aware) and DEEPINFER [7] (delay-oblivious). To perform a fair comparison, we have selected four topics of the Memetracker dataset that have been evaluated in the INFOPATH original paper. The cascades derived from these topics are associated with 5000 users.

[1] https://www.tensorflow.org/
[2] https://www.scipy.org/

Dataset	$\lvert U\rvert$	$\lvert E\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert U'\rvert$
HT	595,460	14,273,311	1,345,913	34,371
RT	595,460	14,273,311	226,488	11,700
MT	3,836,314	15,540,787	71,568	52,088

Table 1. Dataset Summary. Number of users, number of edges, number of cascades, number of users after removing large cascades.

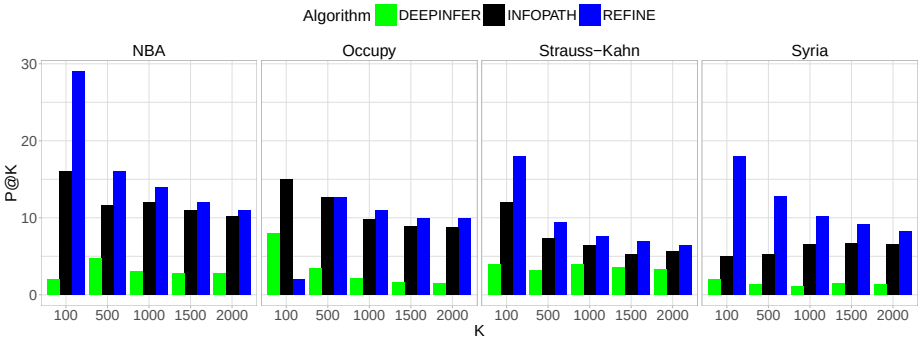

Fig. 2. Comparison of REFINE with the baselines over four topics from the Memetracker dataset for different value of K for the $P@K$ metric. For all datasets, REFINE applies T-SVD and $I_r \in \mathbb{R}^{n\times 1024}$. Cascade length: for *Syria* and *Occupy*, between 3 and 100; for *NBA* and *Strauss Kahn*, between 3 and 1000. (1) *Syria* $n = 1,207$, and $|\mathcal{C}| = 615,176$; REFINE: layer sizes $= [1024, 700, 300, 200]$, learning rate $\alpha = 0.005$, regularization constant $\lambda = 0.0005$. (2) *Occupy*: $n = 1,875$, $|\mathcal{C}| = 655,183$; REFINE: layer sizes $= [1024, 900, 400, 200]$, $\alpha = 0.001$, $\lambda = 0.009$. (3) *NBA*: $n = 2,087$, and $|\mathcal{C}| = 1,543,630$; REFINE: layer sizes $= [1024, 700, 300, 200]$, $\alpha = 0.003$, $\lambda = 0.0005$; (4) *Strauss-Kahn*: $n = 1,263$, and $|\mathcal{C}| = 204,238$; REFINE: layer sizes $= [1024, 800, 500, 200]$, $\alpha = 0.005$, $\lambda = 0.01$. For DEEPINFER: $s = 15$, and $d = 200$. For INFOPATH, we have adopted the exponential influence model, as it performs slightly better than the others.

The results are reported in Fig. 2. REFINE performs better than the baselines in almost all of the cases, by up to an order of magnitude. Apart from this, it is worthwhile to note that a single-threaded version of INFOPATH would require several days to complete. In fact, the original paper reports 4 hours of computation to infer 38 different time-varying networks for 38 different topics, in a cluster equipped with 1000 CPU cores and 6TB total RAM [5]. REFINE has been executed on a 48-core, 128GB machine and takes at most 10 minutes to reconstruct the topic-associated networks for each of the four topics.

In the same figure, it is possible to observe the poor performance of DEEPINFER; this is due to the fact that we only consider 5000 users. To detect patterns, DEEPINFER relies on frequent co-occurrence of users in close contexts; however, we do not have any guarantee that the 5000 users will occur in such manner, hence the poor performance. This would not be an issue for REFINE and INFOPATH, as they rely on reaction time and/or mere co-occurrence patterns rather than context proximity.

In all of the above experiments, the T-SVD step of REFINE has been executed. As shown in Fig. 5, handling the sparsity issue through T-SVD gives better result than the formulation in Eq. 7. REFINE with T-SVD is more robust than REFINE when K increases. However, one could ask if simply using the T-SVD method as an embedding technique could be sufficient. In the following experiment we show that a variant of REFINE, referred to as REFINE-BASIC which simply considers the T-SVD output as node embedding, is not sufficient. For this experiment, we have chosen cascades of minimum length 5 and maximum length 200. In fact, it has been argued that users belonging to large cascades are usually not

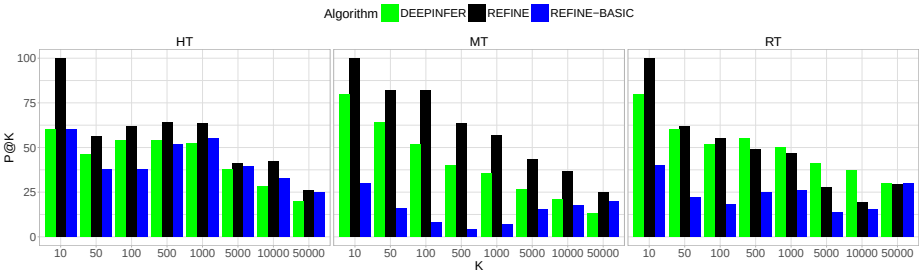

Fig. 3. REFINE vs DEEPINFER. REFINE applies T-SVD, $I_r \in \mathbb{R}^{n \times 1024}$. REFINE: HT – layer sizes $[1024, 700, 500, 300, 100]$, learning rate $\alpha = 0.0001$, regularization constant $\lambda = 0.0002$; RT & MT – layer sizes $[1024, 900, 700, 500, 200]$, $\alpha = 0.001$. RT – $\lambda = 0.004$, and MT – $\lambda = 0.001$. DEEPINFER configuration: window size $s = 10$ and $d = 200$

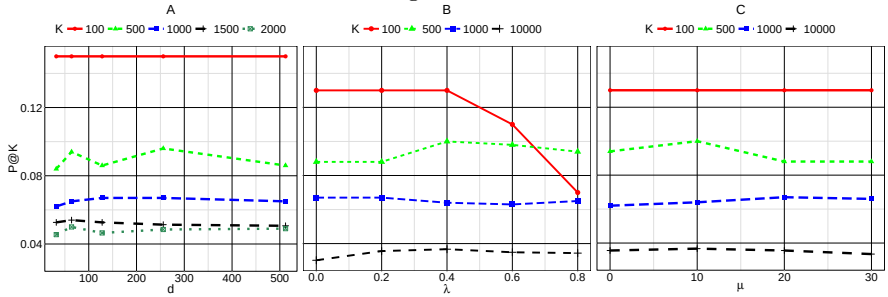

Fig. 4. Parameter sensitivity analysis with respect to (A) embedding size (#dimensions - d), (B) regularization constant (λ), and (C) sparsity penalizer (μ) using Strauss-Kahn.

similar, as such cascades tend to be viral and include almost all users [16]. By discarding cascades which are too large in order to reduce noise, the number of users decreases, as shown in column $|U'|$ of Table 1.

Fig 3 shows how poorly REFINE-BASIC performs when it is compared against REFINE and DEEPINFER. Recall that the network structure is highly non-linear and our main goal for designing the complete REFINE solution is to capture such non-linearity. REFINE-BASIC is a linear model, and hence it fails to effectively predict the edges of the latent network. One particular observation is that REFINE tends to perform well when there is a large number of training examples (i.e. the first two plots). Note that a training example in REFINE corresponds to a user. In Fig 3 we have not included the performance of INFOPATH as it fails to complete the inference on large datasets after several days.

Parameter Analysis. To complete the analysis, we investigate now how the different parameters of our models affect the performance. We start by analyzing the effect of embedding dimensionality in the network reconstruction task. As we are interested in understanding the effect of the parameters, in the following experiments we only set the minimum size of cascades to be 3, i.e. $\{C : |C| \geq 3, C \in \mathcal{C}\}$.

The first plot of Fig. 4 shows the effect of increasing the embedding dimensionality in the network reconstruction task. As one might expect, increasing this parameter up to a given threshold improves the results, because we can en-

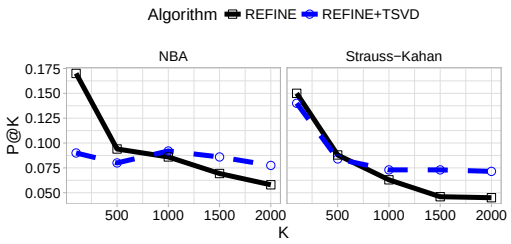

Fig. 5. The effect of using the T-SVD step in REFINE using two topics, NBA and Strauss-Kahn, from the Memetracker dataset

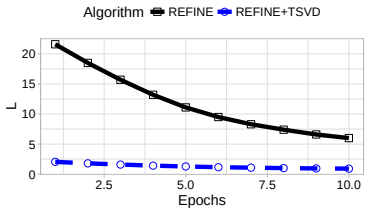

Fig. 6. The progress of the loss function at the end of 10 iterations for REFINE and REFINE with the T-SVD step

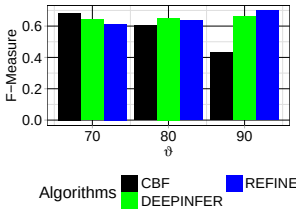

Fig. 7. Virality prediction results REFINE, DEEPINFER and (CBF)

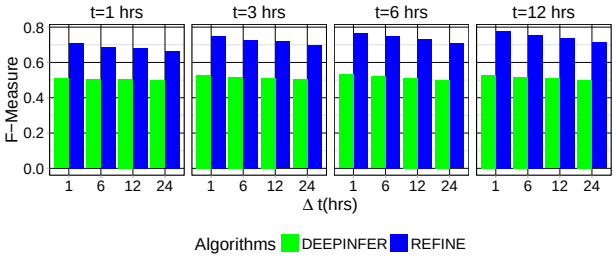

Fig. 8. Virality prediction results: REFINE vs DEEPINFER.

code more information. However, beyond a certain point the performance either reaches a plateau or decreases. Our experiments show that in most of the cases, the best results occur when the embeddings size is in the range 150-200. In the second plot of Fig. 4, the effect of the regularization constant λ (introduced in Eq. 7) is analyzed. In line with previous findings [8], our experiments show that in most of the cases, the best results are obtained when λ is between 0.0 and 0.4; after that point, the performance usually decreases. Finally, in the third plot of Fig. 4 we analyze the effect of the sparsity factor μ, introduced in Eq. 7. Our experiments show that in most of the cases, the best results are obtained when μ is between 0 and 10.

Earlier we have shown the advantage of using T-SVD in terms of the quality of the result; here, we analyze the effect from the convergence of the loss function L, Eq. 7. Fig. 6 shows that the loss function converges much faster (after a couple of iterations) for REFINE with T-SVD rather than REFINE without T-SVD.

Cascade Prediction Besides its effectiveness in network reconstruction, our approach can be extended to perform other tasks, such as *cascade prediction*: given the state of a cascade C up to a certain time t, we want to predict whether the cascade will go viral by time $t + \Delta t$. This is a practically relevant problem and a crucial challenge in social networks analysis [16, 18, 19].

In this study, we formulate the virality prediction problem similarly to Weng et al. [16]. Let $S_t(C) = \{u : u = C_t(i) \wedge i \leq t\}$ be the number of users who participated in a cascade up to a discrete time t. Let ϑ be a *virality threshold*;

we seek to predict whether the cascade will affect a number of users which is larger than $\vartheta\%$ of the recorded cascades.

We utilize the embeddings proposed in Section 3.2. We compute a feature vector $\mathbf{f} \in \mathbb{R}^d$ that encodes the current state of the cascade based on $S_t(C)$ as follows. Let $p = |S_t(C)|$, and let $\mathcal{E} \in \mathbb{R}^{p \times d}$ be an embedding matrix constructed from the set of p starting users at time t, $u \in S_t(C)$. We then compute $\mathbf{f}$ by aggregating $\mathcal{E}$, *i.e.* the $j-th$ component $\mathbf{f}_j$ for $j = 1, \ldots, d$ is computed as $\mathbf{f}_j = \frac{1}{p} \sum_{i=1}^{p} \mathcal{E}_{ij}$.

Once we automatically build the feature vectors, we assign binary labels for each cascade according to their state at $t + \Delta t$ and ϑ. That is, a cascade C is labeled as *viral* if its size at $t + \Delta t$ is greater than the size of $\vartheta\%$ of the cascades; otherwise, it is labeled *non-viral*. Finally, we follow a standard *machine learning* approach by splitting the data into training (60%) and test (40%). To make a fair comparison with community-based features (CBF) [16], we follow the same techniques and settings. As we have a rare-class classification task, we use F-Measure with $\beta = 3$ [18].

We use the same dataset as [16] (Twitter-HT). We compare REFINE with CBF and DEEPINFER; for CBF only, features are manually extracted from the underlying network.

Fig. 7 shows that REFINE is no better than the baselines for $\vartheta = \{70, 80\}$. However it is much better for $\vartheta = 90$ (REFINE = **69.7%**, DEEPINFER = 65.5%, CBF = 43%), and in virality prediction it is crucial to have an effective prediction at higher values of ϑ [18].

A vital task in this problem is to predict virality as early as possible. Therefore, in the following experiments we seek to predict virality of a cascade C at different $t + \Delta t$ based on the observation of C at different values of t with a fixed ϑ. In this experiment, we compare the two strong algorithms REFINE and DEEPINFER, and for both algorithms d is equal to 200. As shown in Fig. 8, REFINE is a clear winner for this task. In particular, note that the prediction quality for REFINE improves as we increase t, and this provides a strong case for the delay-aware approach. As it is difficult to predict far in the future, performance decreases as we increase Δt.

5 Conclusions

This study addresses the problem of network reconstruction from diffusion events through node embedding, and proposes a novel algorithm called REFINE.

One of our objectives is to argue against some existing studies [4] and show that, if carefully designed, delay-aware models are as good as or even better than delay-oblivious models in reconstructing the hidden network.

REFINE is based on user embeddings learned from cascade logs, that are leveraged to predict edge probabilities between pairs of users. Unlike some existing techniques that assume a parametric form of influence model, we make no assumption regarding the transmission rates over edges. Instead, we simply embed the interaction patterns between users in a low-dimensional space and utilize

that for reconstructing the edges. We show the effectiveness of this technique by comparing it against existing delay-aware and delay-agnostic methods.

Moreover, we have also demonstrated the technique presented in this study can be used for cascade prediction. Compared to existing manual or automatic feature extraction techniques, our algorithm shows a significant performance gain. Our study is limited to inferring the existence of edges between a pair of users, and in a future work we seek to infer the direction of edges as well.

References

1. N. Barbieri, F. Bonchi, and G. Manco, "Cascade-based community detection," in *Proc. of WSDM'13*, pp. 33–42, ACM, 2013.
2. M. Gomez Rodriguez, J. Leskovec, and A. Krause, "Inferring networks of diffusion and influence," in *Proc. of KDD'10*, ACM, 2010.
3. N. Du, L. Song, A. Smola, and M. Yuan, "Learning networks of heterogeneous influence," in *Proc. of NIPS'12*, Curran Associates Inc., 2012.
4. S. Lamprier, S. Bourigault, and P. Gallinari, "Extracting diffusion channels from real-world social data: A delay-agnostic learning of transmission probabilities," in *Proc. of ASONAM'15*, ACM, 2015.
5. M. Gomez-Rodriguez, J. Leskovec, and B. Schölkopf, "Structure and dynamics of information pathways in online media," *CoRR*, vol. abs/1212.1464, 2012.
6. M. Gomez-Rodriguez, D. Balduzzi, and B. Schölkopf, "Uncovering the temporal dynamics of diffusion networks," in *Proc. of ICML'11*, Omnipress, 2011.
7. Z. T. Kefato, N. Sheikh, and A. Montresor, "Deepinfer: Diffusion network inference through representation learning," in *Proc. of MLG'17*, ACM, Aug. 2017.
8. D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. of KDD'16*, ACM, 2016.
9. B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. of KDD'14*, ACM, 2014.
10. A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proc. of KDD'16*, ACM, 2016.
11. W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017.
12. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.
13. S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proc. of the IJCAI'16*, pp. 1895–1901, AAAI Press, 2016.
14. Z. T. Kefato, N. Sheikh, and A. Montresor, "Mineral: Multi-modal network representation learning," in *Proc. of MOD'17*, ACM, Sept. 2017.
15. J. Baglama and L. Reichel, "Augmented implicitly restarted lanczos bidiagonalization methods," *SIAM J. Sci. Comput.*, vol. 27, no. 1, pp. 19–42, 2005.
16. L. Weng, F. Menczer, and Y.-Y. Ahn, "Virality prediction and community structure in social networks," *Sci. Rep.*, vol. 3, no. 2522, 2013.
17. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," May 2010.
18. K. Subbian, B. A. Prakash, and L. Adamic, "Detecting large reshare cascades in social networks," in *Proc. of WWW'17*, 2017.
19. J. Cheng, L. Adamic, P. A. Dow, J. M. Kleinberg, and J. Leskovec, "Can cascades be predicted?," in *Proc of WWW'14*, ACM, 2014.