Structured Prediction of Network Response

Hongyu Su HONGYU.SU@AALTO.FI
Aristides Gionis ARISTIDES.GIONIS@AALTO.FI
Juho Rousu JUHO.ROUSU@AALTO.FI
Helsinki Institute for Information Technology (HIIT)
Department of Information and Computer Science, Aalto University, Finland

Abstract

We introduce the following *network response* problem: given a complex network and an action, predict the subnetwork that responds to action, that is, which nodes perform the action and which directed edges relay the action to the adjacent nodes.

We approach the problem through max-margin structured learning, in which a compatibility score is learned between the actions and their activated subnetworks. Thus, unlike the most popular influence network approaches, our method, called SPIN, is *context-sensitive*, namely, the presence, the direction and the dynamics of influences depend on the properties of the actions. The inference problems of finding the highest scoring as well as the worst margin violating networks, are proven to be $\mathbf{NP}$-hard. To solve the problems, we present an approximate inference method through a semi-definite programming relaxation (SDP), as well as a more scalable greedy heuristic algorithm.

In our experiments, we demonstrate that taking advantage of the context given by the actions and the network structure leads SPIN to a markedly better predictive performance over competing methods.

1. Introduction

With the widespread use and extensive availability of large-scale networks, an increasing amount of research has been proposed to study the structure and function of networks. In particular, network analysis has been applied to study dynamic phenomena and complex interactions, such as information propagation, opinion formation, adoption of technological innovations, viral marketing, and disease spreading (De Choudhury et al., 2010; Kempe et al., 2003; Watts & Dodds, 2007).

Influence models typically consider *actions* performed by the network nodes. Examples of such actions include buying a product or (re)posting a news story in one's social network. Often, network nodes perform such actions as a result of influence from neighbouring nodes, and a number of different models have been proposed to quantify influence in a network, most notably the independent-cascade and the linear-threshold models (Kempe et al., 2003). On the other hand, performing an action may also come as a result to an external (out of the network) stimulus, a situation that has also been subject to modeling and analysis (Anagnostopoulos et al., 2008). A typical assumption made by existing models is that influence among nodes depends only on the nodes that perform the action and not on the action itself.

A central question in the study of network influence, is to infer the latent structure that governs the influence dynamics. This question can be formulated in different ways. In one case no underlying network is available (for example, news agencies that do not link each other) and one asks to infer the hidden network structure, e.g., to discover implicit edges between the network nodes (De Choudhury et al., 2010; Du et al., 2012; Eagle et al., 2009; Gomez-Rodriguez et al., 2010; 2011). However, this problem is an unnecessarily hard one to solve in many applications. On the other hand, in many applications the network is known (e.g., "follower" links in twitter), and the research question is to estimate the hidden variables of the influence model (Goyal et al., 2010; Saito et al., 2008).

The present paper is motivated by the following observation: the influence between two nodes in the network does not depend only on the nodes and their connections, but also depends on the action under consideration. For example, if u and v represent users in twitter, v may be influenced from u regarding topics related to *science* but not

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

regarding topics related to, say, *politics*. Thus, in our view, the influence model needs to be *context-sensitive*.

We thus consider the following *network response* problem: given an action, predict which nodes in the network will perform it and along which edges the action will spread. We approach the problem via structured output learning that models the activated *response network* as a directed graph. We learn a function for mappings between action descriptions and the response subnetwork. Given an action, the model is able to predict a directed subnetwork that is most favourable to performing the action.

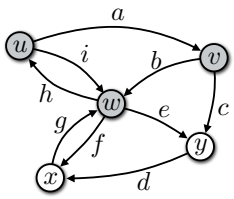
2. Preliminaries

We consider a *directed* network $G = (V, E)$ where the nodes $v \in V$ represent entities, and edges $e = (u, v) \in E$ represent relationships among entities. As discussed in the introduction, for each edge (u, v) we assume that node v can be influenced by node u. In real applications, some networks are directed (e.g., follower networks), while other networks are undirected (e.g., friendship networks). For simplicity of exposition, and without loss of generality we formulate our problem for directed networks; indeed an undirected edge can be modeled by considering pair of directed edges. In our experiments we also consider undirected networks.

In addition to other nodes, we allow the nodes to be influenced by external stimuli, modelled by a *root* node r, which is connected to all other nodes in the network, namely $(r, v) \in E$, for all $v \in V \setminus \{r\}$. Reversely, no node can influence r, so $(v, r) \notin E$, for all $v \in V \setminus \{r\}$.

The second ingredient of our model consists of the actions performed by the network nodes. We write A to denote the underlying *action space*, that is, the set of all possible actions, and we use $\mathbf{a}$ to indicate a particular action in A. We assume that actions in A are represented using a feature map $\phi : A \to \mathcal{F}_A$ to an associated inner product space $\mathcal{F}_A$. For example, $\mathcal{F}_A$ can be a vector space of dimension k, where each action $\mathbf{a}$ is represented by a k-dimensional vector $\phi(\mathbf{a})$. In the social-network application discussed in the introduction, where actions $\mathbf{a}$ correspond to news articles posted by users, $\phi(\mathbf{a})$ can be the bag-of-words representation of the news article $\mathbf{a}$.

We assume that the network gets exposed to an action $\mathbf{a} \in A$, and in response a subgraph $G_\mathbf{a} = (V_\mathbf{a}, E_\mathbf{a}) \subseteq G$, called the *response network* gets activated. The nodes $V_\mathbf{a} \subseteq V$ are the ones that get activated and $E_\mathbf{a} \subseteq E$ is the set of induced edges. We assume that the root r is always activated, i.e., $r \in V_\mathbf{a}$. Note that even though r is directly connected to each node $v \in V_\mathbf{a}$, in every response network $G_\mathbf{a}$, some nodes in $V_\mathbf{a}$ may exercise on v stronger influence than the influence that r exercises on v. The nodes that get directly activated by the root node r as a response to an action are called the *focal points* or *foci* of the response network.



$(u, \mathbf{a}, t = 1)$
$(v, \mathbf{a}, t = 2)$
$(w, \mathbf{a}, t = 5)$

$\mathbf{a} \to \phi(\mathbf{a})$

$$G_\mathbf{a} \to \psi(G_\mathbf{a}) = (a_{\mathtt{pp}}, a_{\mathtt{pn}}, a_{\mathtt{nn}}, b_{\mathtt{pp}}, b_{\mathtt{pn}}, b_{\mathtt{nn}}, c_{\mathtt{pp}}, c_{\mathtt{pn}}, c_{\mathtt{nn}}, d_{\mathtt{pp}}, d_{\mathtt{pn}}, d_{\mathtt{nn}}, \ldots)$$
$$= (\ 1,\ \ 0,\ \ 0,\ \ 1,\ \ 0,\ \ 0,\ \ 0,\ \ 1,\ \ 0,\ \ 0,\ \ 0,\ \ 1,\ \ \ldots)$$

$$\gamma(u) = (\gamma(u; a), \gamma(u; b), \gamma(u; c), \gamma(u; d), \ldots) = (1, \lambda, \lambda, \lambda^2, \ldots)$$

Figure 1. An action $\mathbf{a}$ perfromed by nodes u, v, w of a directed network at times $1, 2, 5$, respectively. Nodes x and y do not perform the action. The action $\mathbf{a}$ is represented by input feature map $\phi(\mathbf{a})$. The response network $G_\mathbf{a}$ is represented by output feature map $\psi(G_\mathbf{a})$ that encodes the propagation of the action $\mathbf{a}$ with respect to edge e (details in the text). Finally, γ is a scaling function (see Sec. 3.4). For instance, $\gamma(u)$ represents a vector of exponentially-decaying weights for node u with respect to all edges.

We assume a dataset $\{(\mathbf{a}_i, G_{\mathbf{a}_i})\}_{i=1}^m$ of m training examples, where each example $(\mathbf{a}_i, G_{\mathbf{a}_i})$ consists of an action $\mathbf{a}_i$ and the output $G_{\mathbf{a}_i}$ encoding the response network activated by $\mathbf{a}_i$. Our intention is to build a model that given a previously unobserved action $\mathbf{a}$, predicts the response network $G_\mathbf{a}$.

3. Model for network responses

3.1. Structured-prediction model

Our method is based on embedding the input and output into a joint feature space and learning in that space a linear compatibility score

$$F(\mathbf{a}, G_\mathbf{a}; \mathbf{w}) = \langle \mathbf{w}, \varphi(\mathbf{a}, G_\mathbf{a}) \rangle.$$

The score $F(\mathbf{a}, G_\mathbf{a}; \mathbf{w})$ is given by the inner product of parameters $\mathbf{w}$ and the joint feature $\varphi(\mathbf{a}, G_\mathbf{a})$. As the joint feature we will use the tensor product $\varphi(\mathbf{a}, G_\mathbf{a}) = \phi(\mathbf{a}) \otimes \psi(G_\mathbf{a})$ of the input feature map $\phi(\mathbf{a})$ of action $\mathbf{a}$, and the output feature map $\psi(G_\mathbf{a})$ that represents the response network $G_\mathbf{a}$ to the action $\mathbf{a}$. The tensor product $\varphi(\mathbf{a}, G_\mathbf{a})$ consists of all pairs of input and output features $\varphi_{ij}(\mathbf{a}, G_\mathbf{a}) = \phi_i(\mathbf{a})\psi_j(G_\mathbf{a})$.

The output features will encode the activated subgraph in the network. We use labels $\{\mathtt{p}, \mathtt{n}\}$ to indicate whether nodes perform an action (positive vs. negative). Similarly, we use edge labels $\{\mathtt{pp}, \mathtt{pn}, \mathtt{nn}\}$ to indicate the role of edges in the propagation of actions. In particular, for each edge $(u, v) = e$ of a response network $G_\mathbf{a}$ and each label $\ell \in \{\mathtt{pp}, \mathtt{pn}, \mathtt{nn}\}$ we define the feature $\psi_{e,\ell}(G_\mathbf{a})$ to

be 1 if and only if e is of type ℓ in $G_{\mathbf{a}}$ (and 0 otherwise). For example, $\psi_{(u,v),\mathtt{pp}}(G_{\mathbf{a}}) = 1$ indicates that both nodes u and v are activated in $G_{\mathbf{a}}$ and u precedes v in the partial order of activation.

An example of the model is shown in Figure 1. For the sake of brievity in the figure, we abuse notation and we use e_ℓ to denote $\psi_{e,\ell}(G_{\mathbf{a}})$. For instance, in this example we have $a_{\mathtt{pp}} = (u,v)_{\mathtt{pp}} = 1$ since both u and v are activated and u precedes v in the activation order, and thus it is possible that u has influenced v.

3.2. Maximum-margin structured learning

The feature weight parameters $\mathbf{w}$ of the compatibility score function F are learned by solving a regularized structured-output learning problem

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\,||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}\xi_i, \qquad (1)\\ \text{s.t.} \quad & F(\mathbf{a}_i, G_{\mathbf{a}_i};\mathbf{w}) > \operatorname*{\mathbf{argmax}}_{G'_{\mathbf{a}_i}\in\mathcal{H}(G)} \big(F(\mathbf{a}_i, G'_{\mathbf{a}_i};\mathbf{w}) \\ & +\ell_G(G'_{\mathbf{a}_i}, G_{\mathbf{a}_i})\big) - \xi_i, \xi_i \geq 0, \forall i = \{1,\cdots,m\}. \end{aligned}$$

The impact of the constraints on the above optimization problem is to push the compatibility score of input $\mathbf{a}_i$ with output $G_{\mathbf{a}_i}$ above the scores of all competing outputs $G'_{\mathbf{a}_i} \in \mathcal{H}(G)$ with a margin proportional to the loss $\ell_G(G'_{\mathbf{a}_i}, G_{\mathbf{a}_i})$ between the correct $G_{\mathbf{a}_i}$ and any competing subgraph $G'_{\mathbf{a}_i}$. $\mathcal{H}(G)$ is the set of directed acyclic subgraphs of G rooted at r. The slack variable ξ_i is used to relax the constraints so that a feasible solution can always be found. C is a slack parameter that controls the amount of regularization in the model. The objective minimizes an L_2-norm regularizer of the weight vector and the slack allocated to the training set. This is equivalent to maximizing the margin subject to allowing some data to be outliers. In practice, the optimization problem (Eq. 1) is tackled by marginal dual conditional gradient optimization (Rousu et al., 2007).

3.3. The inference problem

In the structured prediction model, both in training and in prediction, we need to solve the problem of finding the highest-scoring subgraph for an action. The two problems differ only in the definition of the score: in training we need to iteratively find the subgraph that violates its margins the most, whilst in prediction we need to find the subgraph with the maximum compatibility to a given action. We explain our inference algorithms for the latter problem and note that the first problem is a straightforward variant.

Given feature weights $\mathbf{w}$ and a network $G = (V,E)$, the prediction for a new input action $\mathbf{a}$ is the maximally-scoring response graph $H^* = (V^H, E^H)$

$$H^*(\mathbf{a}) = \operatorname*{\mathbf{argmax}}_{H\in\mathcal{H}(G)} F(\mathbf{a}, H;\mathbf{w}).$$

Writing this problem explicitly, in terms of the parameters and the feature maps gives

$$\begin{aligned} H^*(\mathbf{a}) &= \operatorname*{\mathbf{argmax}}_{H\in\mathcal{H}(G)} \langle \mathbf{w}, \phi(\mathbf{a}) \otimes \psi(H)\rangle \\ &= \operatorname*{\mathbf{argmax}}_{H\in\mathcal{H}(G)} \sum_{e\in E^H} s_{y_e}(e,\mathbf{a}), \qquad (2) \end{aligned}$$

where we have substituted $s_{y_e}(e,\mathbf{a}) = \sum_i w_{i,e,y_e}\phi_i(\mathbf{a})$. We will abbreviate $s_{y_e}(e,\mathbf{a})$ to $s_{y_e}(e)$, as the action $\mathbf{a}$ is fixed for an individual inference problem. The output response network H can be specified by a node label $y_v \in \{\mathtt{p},\mathtt{n}\}$, where $y_v = \mathtt{p}$ if and only if v is activated. We write H_y to emphasize the dependence of the output subgraph H from labelling y. The node labels y_v induce edge labels y_e. The score function $s(e)$ can be interpreted as a score function for the edges, given by the current input $\mathbf{a}$ and weight vector $\mathbf{w}$. The variable y_e indicates the possible labels of an edge e, and for each possible label the score function $s(e)$ assigns a different score. Depending on the values that y_e can take, the inference problem can be further diverged into two modes:

Activation mode. We assume $y_e \in \{\mathtt{pp},\mathtt{pn}\}$ where $y_e = \mathtt{pp}$ implies node v is activated by u via a directed edge $e = (u,v)$, and $y_e = \mathtt{pn}$ means that the activation cannot pass through e. In activation mode, the inference problem is transformed as finding the maximally scoring node label y_v and corresponding edge label y_e, consistent with an activated subgraph H_y given a set of edge scores $s_{y_e}(e)$.

Negative-feed mode. In addition to the setting in activation mode, we also explicitly model the inactive network by assume $y_e \in \{\mathtt{pp},\mathtt{pn},\mathtt{nn}\}$, where by $y_e = \mathtt{nn}$ we denote our belief that both u and v should be inactive given action $\mathbf{a}$. The inference problem is then to find the maximally scoring node labels and induced edge labels with regards to an activated subgraph together with the inactive counterpart given a set of edge score $s_{y_e}(e)$.

It is not difficult to show that the inference problem (Eq. 2) is **NP**-hard. The proof of the following lemma, which provides a reduction from the MAX-CUT problem, is given in the supplementary material.

Lemma 1 *Finding the graph that maximizes Eq. (2) is an* **NP***-hard problem.*

To solve the inference problem we propose two algorithms, described on the negative-feed mode. Similar techniques can be adapted to the activation-mode by setting edge score $s_{\mathtt{nn}}(e) = 0$. The first algorithm is based on a semidefinite programming (SDP) relaxation, similar to the one

used for MAX-CUT and satisfiability problems (Goemans & Williamson, 1995). The SDP algorithm offers a constant-factor approximation guarantee for the inference problem. However, it requires solving semidefinite programs. Efficient solvers do exist, but the method is not scalable to large datasets. Besides, it cannot handle the order of activations. In contrast, our second approach is a more efficient GREEDY algorithm that models activation order in a natural way, but it does not provide any quality guarantee.

The SDP inference. Recall that for each edge $(u, v) \in E$ we are given three scores: $s_{\texttt{pp}}(u, v)$, $s_{\texttt{pn}}(u, v)$, and $s_{\texttt{nn}}(u, v)$. The inference problem is to assign a label p or n for each vertex $u \in V$. If a vertex u is assigned to label p we say that u is *activated*. If both vertices u and v of an edge $(u, v) \in E$ are activated, a gain $s_{\texttt{pp}}(u, v)$ incurs. Respectively, the assignments pn and nn yield gains $s_{\texttt{pn}}(u, v)$ and $s_{\texttt{nn}}(u, v)$. The objective is to find the assignments that maximizes the total gain.

We formulate this optimization problem as a quadratic program. We introduce a variable $x_u \in \{-1, +1\}$, for each $u \in V$. We also introduce a special variable $x_0 \in \{-1, +1\}$, which is used to distinguish the activated vertices. In particular, if $x_u = x_0$ we consider that the vertex u is assigned to label p, and thus it is activated, while $x_u = -x_0$ implies that u is assigned to n and not activated. The network-response inference problem can now be written as (QP):

$$\begin{aligned}\textbf{max}\ \frac{1}{4} \sum_{(u,v)\in E} & [s_{\texttt{pn}}(u, v)(1 + x_0x_u - x_0x_v - x_ux_v) \\ & + s_{\texttt{nn}}(u, v)(1 - x_0x_u - x_0x_v + x_ux_v) \\ & + s_{\texttt{pp}}(u, v)(1 + x_0x_u + x_0x_v + x_ux_v)], \\ \text{s.t.}\quad & x_0, x_u, x_v \in \{-1, +1\}, \text{ for all } u, v \in V.\end{aligned}$$

The intuition behind the formulation of Problem (QP) is that there is gain $s_{\texttt{pn}}(u, v)$ if $x_0 = x_u = -x_v$, a gain $s_{\texttt{nn}}(u, v)$ if $x_0 = -x_u = -x_v$, and a gain $s_{\texttt{pp}}(u, v)$ if $x_0 = x_u = x_v$.

To solve the problem (QP), we use the similar technique introduced by Goemans & Williamson (1995), such that each variable x_u is *relaxed* to a *vector* $\mathbf{v}_u \in \mathbb{R}^n$. The relaxed quadratic program becomes (RQP):

$$\begin{aligned}\textbf{max}\ \frac{1}{4} \sum_{(u,v)\in E} & [s_{\texttt{pn}}(u, v)(1 + \mathbf{v}_0\mathbf{v}_u - \mathbf{v}_0\mathbf{v}_v - \mathbf{v}_u\mathbf{v}_v) \\ & + s_{\texttt{nn}}(u, v)(1 - \mathbf{v}_0\mathbf{v}_u - \mathbf{v}_0\mathbf{v}_v + \mathbf{v}_u\mathbf{v}_v) \\ & + s_{\texttt{pp}}(u, v)(1 + \mathbf{v}_0\mathbf{v}_u + \mathbf{v}_0\mathbf{v}_v + \mathbf{v}_u\mathbf{v}_v)], \\ \text{s.t.}\quad & \mathbf{v}_i \in \mathbb{R}^n, \text{for all } i = 0, \ldots, n.\end{aligned}$$

Consider an $(n+1) \times (n+1)$ matrix Y whose (u, v) entry is $y_{u,v} = \mathbf{v}_u \cdot \mathbf{v}_v$. If V is the matrix having $\mathbf{v}_u$'s as its columns, i.e., $V = [\mathbf{v}_0 \ldots \mathbf{v}_k]$, then $Y = V^TV$, implying that the matrix Y is semidefinite, a fact we denote by $Y \succeq 0$. Problem (RQP) now becomes (SDP):

$$\begin{aligned}\textbf{max}\ \frac{1}{4} \sum_{u,v=1}^{k} & [s_{\texttt{pn}}(u, v)(1 + y_{0,u} - y_{0,v} - y_{u,v}) \\ & + s_{\texttt{nn}}(u, v)(1 - y_{0,u} - y_{0,v} + y_{u,v}) \\ & + s_{\texttt{pp}}(u, v)(1 + y_{0,u} + y_{0,v} + y_{u,v})], \\ \text{s.t.}\quad & Y \succeq 0.\end{aligned}$$

Problem (SDP) asks to find a semidefinite matrix, so that a linear function on the entries of the matrix is optimized. This problem can be solved by semidefinite programming within accuracy ϵ, in time that it is polynomial on k and $\frac{1}{\epsilon}$. After solving the semidefinite program one needs to *round* each vector $\mathbf{v}_u$ to the variable $x_u \in \{-1, +1\}$ in the following way:

1. Factorize Y with Cholesky decomposition to find $V = [\mathbf{v}_0, \mathbf{v}_1 \ldots \mathbf{v}_n]$.
2. Select a random vector $\mathbf{r}$.
3. For each $u = 0, 1, \ldots, n$, if $\mathbf{v}_u \cdot \mathbf{r} \geq 0$ set $x_u = 1$, otherwise set $x_u = -1$.

Let Z be the value of the solution obtained by the above algorithm. Let Z^* be the optimal value of Problem (QP) and Z_R the optimal value of Problem (SDP). Since Problem (SDP) is a relaxation of Problem (QP) it is $Z_R \geq Z^*$. Furthermore, it can be shown that for the *expected* value of Z it holds $E[Z] \geq (\alpha - \epsilon)Z_R$, with $\alpha > 0.796$ and where expectation is taken over the choice of $\mathbf{r}$. Thus the above algorithm is a 0.796 approximation algorithm for Problem (QP).

The GREEDY inference. The inference (Eq. 2) is defined on all edges of the network, which can be expressed equivalently as a function of activated vertices (see details in supplementary)

$$H^*(\mathbf{a}) = \underset{H\in\mathcal{H}(G)}{\textbf{argmax}} \sum_{v_i \in V_{\texttt{p}}^H} F_m(v_i),$$

where $V_{\texttt{p}}^H$ is a set of activated vertices. $F_m(v_i)$ is the marginal gain on each node that is comprised partially from changing edge label from pn to pp on incoming edges $\{(v_p, v_i) \mid v_p \in \text{parents}(v_i)\}$, and partially from changing edge label from nn to pn on outgoing edges $\{(v_i, v_c) \mid v_c \in \text{parents}(v_i)\}$ defined as

$$\begin{aligned}F_m(v_i) = & \sum_{v_p \in \text{parents}(v_i)} [s_{\texttt{pp}}(v_p, v_i) - s_{\texttt{pn}}(v_p, v_i)] \\ & + \sum_{v_c \in \text{children}(v_i)} [s_{\texttt{pn}}(v_i, v_c) - s_{\texttt{nn}}(v_i, v_c)].\end{aligned}$$

It is difficult to maximize the sum of marginal gains as the activated subnetwork is unknown. One can instead compute for each vertex the maximized marginal gain $\mathbf{max}_{v_i} F_m(v_i)$ in an iterative fashion as long as $F_m(v_i) \geq 0$, which leads to a greedy algorithm described as follows. The algorithm starts with an activated vertex set $V_{\mathrm{p}}^H = \{r\}$. In each iteration, it chooses a vertex $v_i \in V/V_{\mathrm{p}}^H$ and adds to V_{p}^H such that v_i is the current maximizer of $F_m(v)$. The procedure terminates if the maximized gain is smaller than 0. E^H can be obtained by adding edges $e = (v_i, v_j) \in E$, if $v_i, v_j \in V_{\mathrm{p}}^H$ and v_i was added to V_{p}^H prior to v_j. The time complexity for greedy inference algorithm is $O(|E| \log |V|)$. See supplementary material for details of the algorithm.

We note that we have not been able to show an approximation guarantee for the quality of solutions produced by the GREEDY algorithm. A property that it is typically used to analyse greedy methods is *submodularity*. However, for this particular problem submodularity does not hold (it only holds in the special case of MAX-CUT, i.e., when $s_{\mathrm{pp}}(e) = s_{\mathrm{nn}}(e) = 0$ and $s_{\mathrm{pn}}(e) = 1$).

3.4. Loss functions

Instead of penalizing prediction mistakes uniformly on the network G, we wish to focus in the vicinity of the response network. To achieve this effect we scale the loss accrued on the nodes and edges by their distance to the children of the root of the response network.

As the loss function in (1) we use *symmetric-difference loss* (or Hamming loss), applied to the nodes and the edges of the subgraphs separately, and scaled by function $\gamma_G(v_k)$ according distance to the focal point v_k.

$$\ell_G(G_{\mathbf{a}}, G_{\mathbf{b}}) = \sum_{v \in V} \ell_v^{\Delta}(G_{\mathbf{a}}, G_{\mathbf{b}}) \gamma_G(v_k; v) + \sum_{(v,v') \in E} \ell_{v,v'}^{\Delta}(G_{\mathbf{a}}, G_{\mathbf{b}}) \gamma_G(v_k; v),$$

where $\ell_v^{\Delta}(G_{\mathbf{a}}, G_{\mathbf{b}}) = [v \in V_a \Delta V_b]$, $\ell_e^{\Delta}(G_{\mathbf{a}}, G_{\mathbf{b}}) = [e \in E_a \Delta E_b]$, $S \Delta S'$ denotes the symmetric difference of two sets S and S'. We consider the following strategies to construct the scaling function $\gamma_G(v_k)$:

Exponential scaling. Mistakes are penalized by λ and λ is weighted exponentially according to the shortest path distance to the focal point v_k. Given focal point v_k, edge (v_i, v_j), and distance matrix D between the nodes, the scaling function is defined as

$$\gamma_G(v_k; v_i, v_j) = \begin{cases} 1 & \text{if } i = 0 \\ \lambda^{D(k,i)} & \text{if } i \neq 0 \text{ and } D(k,i) \leq R \\ \lambda^{(R+1)} & \text{if } D(k,i) > R \end{cases}$$

where $\lambda > 0$ is the scaling factor and $R > 1$ is a radius parameter. Edges outside the radius have equal scalings.

Diffusion scaling. The diffusion kernel defines a distance-based function between nodes v_i and v_j (Kondor & Lafferty, 2002). The kernel value $K(i, j)$ corresponds to the probability of a random walk from node v_i to node v_j. Given the adjacency matrix L of the network G, the diffusion kernel is computed as

$$K = \lim_{s \to \infty} \left(I + \frac{\beta L}{s} \right)^s = \exp(\beta L),$$

where I is the identity matrix and β is the a parameter that controls how much the random walks deviate from the focal point. Given focal node v_k, edge (v_i, v_j), and diffusion kernel K the scaling function is defined as

$$\gamma_G(v_k; v_i, v_j) = \begin{cases} 1 & \text{if } i = 0, \\ K(v_k, v_i) & \text{otherwise.} \end{cases}$$

The scaling function keeps the loss value on the edges connecting the focal point, and scale other edges by the weights computed from diffusion kernel. Diffusion scaling has the effect of shrinking the distance to nodes that connects to the focal point by many paths.

4. Experimental evaluation

In this section, we evaluate the performance of SPIN and compare it with the state-of-the-art methods through extensive experiments. We use two real-world datasets, DBLP and Memetracker, described below. Statistics of the datasets are given in Table 1.

DBLP[1] dataset is a collection of bibliographic information on major computer science journals and proceedings. We extract a subset of original data by using "inproceedings" articles from year 2000. First, we construct an undirected DBLP network G by connecting pairs of authors who have coauthored more that p papers ($p = 5, 10, 15$). After that, we generate a set of experimental networks of different size by performing snowball sampling (Goodman, 1961). For each experimental network, we extract all the documents for which at least one of their authors is a node in the network. We apply LDA algorithm (Blei et al., 2002) on the titles of extracted documents to generated topics. Topics are associated with publications, timestamped by publication dates, and described by bag-of-word features computed from LDA. In this way, a topic can be seen as an action and we will study the influence among authors.

Memetracker[2] dataset is a set of phrases propagated over prominent online news sites in March 2009. We construct

[1] http://www.informatik.uni-trier.de/~ley/db/

[2] http://Memetracker.org

directed networks G for Memetracker dataset by connecting two websites via a directed edge if there are at least five phrases copying from one website to the other. A posted phrase corresponds to an action, which again is timestamped and represented with bag-of-word features.

4.1. Experimental setup and metrics

SPIN can be applied to predict action-specific network response (contenxt-aware) when action representation $\phi(a)$ is given as input. It is also capable of predicting edge influence scores in context-free mode when $\phi(a)$ is treated as unknown. For comparison purposes, we evaluate SPIN against the following the state-of-the-art methods:

- Support Vector Machine (SVM) is used as a single target classifier used to predict the response network via decomposing it as a bag of nodes and edges, and predicting each element in the bag.
- Max-Margin Conditional Random Field (MMCRF) (Rousu et al., 2007; Su et al., 2010) is a multi-label classifier that utilizes the structure of output graph G. The model predicts the node labels of the network.
- Expectation-Maximization for the independent cascade model (ICM-EM) (Saito et al., 2008) is a context-free model that infers the influence probability of the network given a directed network and a set of action cascades. Here we use the implementation from Mathioudakis et al. (2011) of this algorithm, which is publicly available[3].
- Netrate (Gomez-Rodriguez et al., 2011) models the network influence as temporal processes occurs at difference rate. It infers the directed edges of the global network and estimates the transmission rate of each edge.

To quantitatively evaluate the performance of the tested methods in predicting node and edge labels, we adopt two popular metrics: *accuracy* and F_1 *score*, defined as

$$F_1 = \frac{2 \cdot P \cdot R}{P + R},$$

where P is precision and R is recall. We also define *Predicted Subgraph Coverage* (PSC) as

$$\text{PSC} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{v \in V_i} |G_v|,$$

where V_i is the set of focal points given action a_i, n is the number of nodes in the network, and m is the number of actions. PSC expresses the relative size of a correctly predicted subgraph G_v in terms of node predictions that cover the focal points v.

[3] https://dl.dropboxusercontent.com/u/21620176/public_html/spine/index.html

Dataset	Training Example	Feature Space	Network $\lvert V\rvert$	Network $\lvert E\rvert$
DBLP S100	440	1190	100	204
DBLP M100	478	1127	100	151
DBLP M500	2119	3619	500	699
DBLP M700	2800	4369	699	952
DBLP M1k	3720	5281	1000	1368
DBLP M2k	6030	7183	2000	2687
DBLP L100	509	1274	100	152
DBLP L500	1869	3424	499	701
DBLP L700	2620	4300	699	960
DBLP L1k	3560	5405	1000	1368
DBLP L2k	3618	5454	1023	1402
memeS	4632	181	82	325
memeM	4804	179	182	521
memeL	4809	179	333	597

Table 1. Statistics of DBLP and Memetracker datasets.

Data	Accuracy			F_1 Score			Time (10^2s)		
	SDP	Neg	Act	SDP	Neg	Act	SDP	Neg	Act
S100	**79.9**	*77.6*	72.9	**57.2**	*56.2*	55.5	16.0	*1.5*	**0.2**
M100	**75.8**	*73.6*	68.5	51.6	*53.1*	**54.5**	15.2	*1.4*	**0.2**
L100	**75.1**	*72.0*	67.4	53.5	*56.9*	**57.2**	13.7	*1.6*	**0.3**
Geom.	**76.9**	*74.3*	69.6	52.0	*55.4*	**55.7**	15.0	*1.5*	**0.3**

Table 2. Comparison of different inference algorithms. $Geom.$ is geometric mean of rows.

Our metrics are computed both in *global context* where we pool all the nodes and edges from the background network, as well as in *local context* where we only collect the nodes and edges within certain radius R of the focal points. The experimental results are from a five-fold cross validation.

4.2. Experimental results

We examine whether our context-sensitive structure predictor can boost the performance of predicting network responses. We compare SPIN with other methods in both context-sensitive and context-free problems. We show that SPIN can perform significantly better in terms of predicting action-specific network responses.

Comparison of inference algorithms. Table 2 shows the geometric mean of node accuracy, F_1 and running time over parameter space on three DBLP datasets, where "Neg" and "Act" represent the GREEDY inference defined on the negative-feed and the activation modes. SDP is also formulated on the negative-feed mode. In general, the inference algorithm based on negative-feed mode outperforms activation mode in terms of accuracy. The difference in F_1 is smaller in comparison. SDP based inference surpasses GREEDY inference in accuracy, however, by a small margin. In addition, GREEDY inference is almost 10 times faster even on small datasets, where running time is total time used for cross validation. For the following experiments, we opted for GREEDY inference in negative-feed mode as the inference engine of SPIN.

Dataset	Node Accuracy			Node F_1 Score			Edge Acc		*PSC*			Time (10^3s)		
	SVM	MMCRF	SPIN	SVM	MMCRF	SPIN	SVM	SPIN	SVM	MMCRF	SPIN	SVM	MMCRF	SPIN
memeS	**73.4**	68.0	*72.2*	39.0	*39.8*	**47.1**	**62.7**	45.6	23.4	*25.3*	**33.6**	6.6	**2.9**	*4.1*
memeM	**82.1**	79.0	*81.5*	29.1	*30.1*	**38.0**	61.1	**68.8**	18.6	*18.8*	**28.3**	13.7	**3.2**	*7.3*
memeL	**89.9**	88.3	*89.8*	26.7	*27.1*	**35.0**	45.5	**80.0**	17.7	*18.9*	**27.6**	19.9	**5.9**	*11.8*
M100	71.2	*73.6*	**76.7**	49.3	*50.8*	**54.3**	33.3	**61.7**	33.3	**35.6**	*34.6*	**0.1**	0.2	**0.1**
M500	89.0	*91.4*	**92.0**	**18.8**	13.5	*14.6*	28.2	**92.6**	*29.3*	26.4	**29.5**	9.0	*3.8*	**3.2**
M700	91.9	**94.1**	*92.1*	*13.8*	7.3	**14.2**	26.3	**93.0**	*29.4*	23.9	**34.4**	18.5	*8.3*	**4.4**
M1k	94.1	**95.8**	*94.2*	**10.9**	3.5	*9.3*	26.6	**94.7**	*33.7*	16.6	**35.2**	42.2	*14.7*	**10.4**
M2k	*96.8*	**97.6**	96.7	**6.2**	1.4	*3.4*	25.3	**97.6**	**34.6**	9.6	*14.7*	165.0	*88.4*	**54.1**
L100	69.4	*72.2*	**75.7**	51.1	*53.1*	**57.4**	31.6	**62.3**	30.9	*31.7*	**33.4**	**0.1**	*0.2*	0.3
L500	85.9	**89.1**	*86.8*	*21.7*	15.1	**24.7**	27.9	**87.9**	*14.2*	11.2	**19.7**	6.5	*3.2*	**2.1**
L700	89.7	**92.4**	*89.7*	*16.2*	9.4	**17.3**	26.5	**90.4**	*9.5*	6.7	**12.5**	16.0	*7.8*	**5.3**
L1k	*92.4*	**94.4**	91.5	*12.4*	6.4	**13.9**	26.4	**92.3**	*6.1*	4.4	**8.4**	40.3	*13.7*	**10.4**
L2k	*92.5*	**94.5**	91.9	*12.3*	5.4	**12.7**	26.5	**93.2**	*6.0*	2.9	**7.2**	41.9	*21.9*	**13.1**
Geom.	85.5	*86.4*	**86.6**	*19.8*	12.6	**20.3**	32.6	**79.7**	*18.9*	14.2	**21.7**	9.4	*4.6*	**4.3**

Table 3. Comparison of prediction performance on global context. The best in bold-face, the second best in italic.

Context-aware prediction. We apply SPIN with exponential scaling to predict context-sensitive network responses. Comparison of prediction performance against SVM and MMCRF is listed in Table 3. We show that SPIN can dramatically boost the performance of all measures except node accuracy: MMCRF wins in node accuracy, but SPIN is the second best and the difference is small. In terms of time consumption for training, SPIN is around three times faster than SVM and two times faster than MMCRF on the largest M2k dataset.

Context-free network influence prediction. Here we compare SPIN to methods developed for influence network prediction, namely Netrate and ICM-EM, on Memetracker data. To make the comparison fair to the competition, we convert the network to undirected network and replace action features by a constant value. For SPIN, we further represent each undirected edge by two directed edges. The measure of success is *Precision*@K, where we ask for top-K percent edge predictions from each model and compute the precision. Table 4 shows *Precision*@K as function of K, where the performance of SPIN surpasses ICM-EM and Netrate in all spectrum of K with a noticeable margin. ICM-EM has the least accurate predictions of the three, but achieves by far the the best running time. SPIN and Netrate solve more complex convex optimization problems, leading more accurate predictions at the cost of more CPU time needed for training, SPIN being the more efficient on the largest dataset, memeL.

The good performance of SPIN compared to Netrate is mostly explained by the fact that Netrate solves a much harder problem in which the underlying undirected network is assumed to be unknown, while SPIN is able to leverage the known network structure. In the experiment reported, the edge predictions from Netrate are filtered against the underlying complex network, in order to excessively penalize influence predictions along non-linked nodes.

Effect of loss scaling. Figure 2 depicts the effect of parameter λ of the exponential loss scaling to prediction performance on subgraphs of different radius. SVM (dashed line) is used as the baseline. When $0 < \lambda < 1$, the node prediction accuracy (top, left) and F_1 (top, right) decrease by the increasing subgraph radius, while $\lambda \geq 1$ leads to the opposite behavior allowing larger subgraph to be learned. Predicted subgraph coverage decreases by incresing λ. Edge prediction accuracy (bottom, right) increases monotonically in λ implying that predicting the longer influence paths is a hard problem for SVM. In Table 5 we examine the performance of diffusion scaling. The numbers reported are geometric means over the different Memetracker and DBLP datasets. We observe a decreased performance when increasing the parameter β, which corresponds to smoothing the distance matrix. This indicates

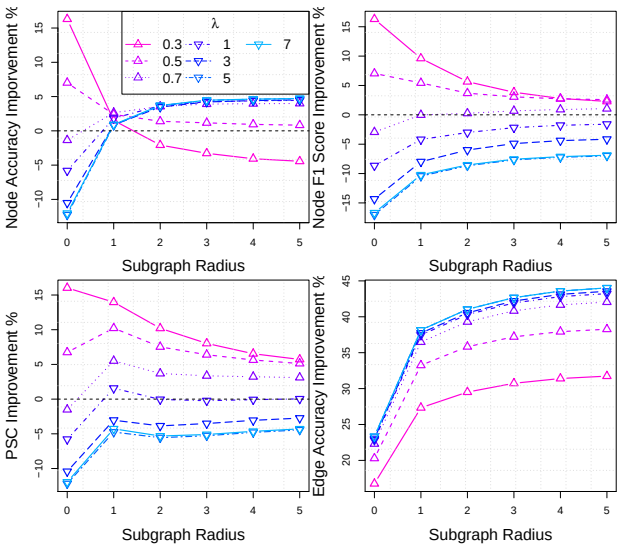


Figure 2. The improvement of prediction performance for different scaling factor λ with respect to SVM.

Dataset	Model	T (10^3s)	Precision @ K									
			10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
memeS	SPIN	*5.50*	**82.9**	**81.0**	**76.0**	**74.0**	**74.0**	**70.0**	**69.8**	**67.9**	**66.7**	**64.7**
	ICM-EM	**0.01**	60.3	63.5	65.1	62.0	62.0	61.5	62.2	60.4	60.7	*61.9*
	NETRATE	5.83	*76.2*	*73.8*	*70.4*	*68.7*	*68.7*	*66.8*	*64.9*	*63.4*	*62.9*	*61.9*
memeM	SPIN	*5.52*	**82.7**	**72.1**	**70.5**	**69.2**	**69.2**	**67.9**	**66.2**	**65.6**	**64.3**	**64.2**
	ICM-EM	**0.02**	56.3	55.3	56.8	57.4	57.4	56.3	57.5	57.8	58.3	*58.5*
	NETRATE	13.93	*61.2*	*64.6*	*62.9*	*62.5*	*62.5*	*62.4*	*61.2*	*60.1*	*58.7*	*58.5*
memeL	SPIN	*4.75*	**82.2**	**73.6**	**69.1**	**66.7**	**66.7**	**65.9**	**66.1**	**65.9**	**63.9**	**63.6**
	ICM-EM	**0.01**	52.1	55.7	54.2	56.5	56.5	56.7	57.4	58.0	*57.6*	*57.0*
	NETRATE	12.63	*56.5*	*57.8*	*60.0*	*59.3*	*59.3*	*59.4*	*58.9*	*58.4*	57.5	*57.0*

Table 4. Model performance in context-free influence network prediction.

Loss Scaling	Node Acc		Node F_1		Edge Acc		*PSC*		Time (10^3s)	
	Meme	DBLP	Meme	DBLP	Meme	DBLP	Meme	DBLP	Meme	DBLP
Dif $\beta = 0.1$	80.8	86.5	40.0	28.6	63.0	80.5	30.2	30.3	68.3	2.7
Dif $\beta = 0.5$	66.4	86.5	42.5	28.5	40.9	80.5	33.0	30.2	50.9	4.0
Dif $\beta = 0.8$	63.5	86.5	40.9	28.5	39.3	80.5	31.2	30.2	32.6	3.2
Exp $\lambda = 0.5$	80.9	83.9	39.7	28.7	63.1	77.7	29.7	24.3	71.0	10.8

Table 5. Comparison of diffusion scaling with exponential scaling.

that emphasizing connections between long-distance nodes makes prediction more difficult, a finding consistent with the results on exponential scaling. Setting $\beta = 0.1$ leads to comparable performance over exponential scaling with $\lambda = 0.5$, with slight improvement on the DBLP datasets.

5. Discussion

We have presented a novel approach, based on structured output learning, to the problem of modelling influence in networks. In contrast to previous state-of-the-art approaches, such as Netrate and ICM-EM, our proposal, named SPIN, is a context-sensitive model. SPIN does not try to force global influence parameters, but instead it incorporates the action space into the learning process and makes predictions tailored to the action under consideration. Our method can provide a useful tool in market research or other application scenarios when actions arise from a high-dimensional space, and one wants to make predictions for actions not seen before. Another benefit of our approach, compared to other state-of-the-art methods, is that our method does not make explicit assumptions regarding the underlying propagation model. Additionally, action responses are explicitly formulated as directed acyclic subgraphs, and the model is capable of predicting the complete subgraph structure. We proved that the inference problem of SPIN is $\mathbf{NP}$-hard, and we provided an approximation algorithm based on semidefinite programming (SDP). In addition, we developed a greedy heuristic algorithm for the inference problem that scales linearly in the size of the network, with time consumption in the same ballpark as Netrate. With extensive experiments we show that SPIN can dramatically boost the performance of action-based network-response prediction. SPIN can also be applied in context-free prediction where it captures the edge influence weight of the network.

References

Anagnostopoulos, Aris, Kumar, Ravi, and Mahdian, Mohammad. Influence and correlation in social networks. *KDD*, 2008.

Blei, D., Ng, A., and Jordan, M. Latent dirichlet allocation. In Dietterich, T., Becker, S., and Ghahramani, Z. (eds.), *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

De Choudhury, Munmun, Mason, Winter A, Hofman, Jake M, and Watts, Duncan J. Inferring relevant social networks from interpersonal communication. *WWW*, pp. 301–310, 2010.

Du, Nan, Song, Le, Smola, Alex, and Yuan, Ming. Learning Networks of Heterogeneous Influence. *NIPS*, 2012.

Eagle, Nathan, Pentland, Alex Sandy, and Lazer, David. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, 2009.

Goemans, Michel and Williamson, David. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *JACM*, 42 (6), 1995.

Gomez-Rodriguez, Manuel, Leskovec, Jure, and Krause, Andreas. Inferring Networks of Diffusion and Influence. *KDD*, 2010.

Gomez-Rodriguez, Manuel, Balduzzi, David, and Schölkopf, Bernhard. Uncovering the Temporal Dynamics of Diffusion Networks. *ICML*, 2011.

Goodman, Leo A. Snowball sampling. *The annals of mathematical statistics*, 32(1):148–170, 1961.

Goyal, Amit, Bonchi, Francesco, and Lakshmanan, Laks VS. Learning influence probabilities in social networks. *WSDM*, 2010.

Kempe, David, Kleinberg, Jon, and Tardos, Éva. Maximizing the spread of influence through a social network. In *KDD*, 2003.

Kondor, I.R. and Lafferty, J. D. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the ICML*, 2002.

Mathioudakis, Michael, Bonchi, Francesco, Castillo, Carlos, Gionis, Aristides, and Ukkonen, Antti. Sparsification of influence networks. *KDD*, 2011.

Rousu, J., Saunders, C., Szedmak, S., and Shawe-Taylor, J. Efficient algorithms for max-margin structured classification. *Predicting Structured Data*, pp. 105–129, 2007.

Saito, Kazumi, Nakano, Ryohei, and Kimura, Masahiro. Prediction of information diffusion probabilities for independent cascade model. In *Knowledge-Based Intelligent Information and Engineering Systems (KES)*, 2008.

Su, Hongyu, Heinonen, Markus, and Rousu, Juho. Structured output prediction of anti-cancer drug activity. In *Proceedings of the 5th IAPR international conference on Pattern recognition in bioinformatics*, PRIB'10, 2010.

Watts, Duncan J and Dodds, Peter Sheridan. Influentials, networks, and public opinion formation. *Journal of consumer research*, 34(4):441–458, 2007.