Predicting Positive and Negative Links in Signed Social Networks by Transfer Learning

Jihang Ye Hong Cheng Zhe Zhu Minghua Chen
The Chinese University of Hong Kong
{yjh010, zzhu}@alumni.ie.cuhk.edu.hk, hcheng@se.cuhk.edu.hk,
minghua@ie.cuhk.edu.hk

ABSTRACT

Different from a large body of research on social networks that has focused almost exclusively on positive relationships, we study signed social networks with both positive and negative links. Specifically, we focus on how to reliably and effectively predict the signs of links in a newly formed signed social network (called a *target network*). Since usually only a very small amount of edge sign information is available in such newly formed networks, this small quantity is not adequate to train a good classifier. To address this challenge, we need assistance from an existing, mature signed network (called a *source network*) which has abundant edge sign information. We adopt the transfer learning approach to leverage the edge sign information from the source network, which may have a different yet related joint distribution of the edge instances and their class labels.

As there is no predefined feature vector for the edge instances in a signed network, we construct generalizable features that can transfer the topological knowledge from the source network to the target. With the extracted features, we adopt an AdaBoost-like transfer learning algorithm with instance weighting to utilize more useful training instances in the source network for model learning. Experimental results on three real large signed social networks demonstrate that our transfer learning algorithm can improve the prediction accuracy by 40% over baseline methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

Keywords

Signed Social Network; Sign Prediction; Transfer Learning

1. INTRODUCTION

Online social networks such as Facebook, Twitter and LinkedIn have been gaining increasing popularity in recent years. People usually form links to indicate *friend* or *follow* relationships. But in some other social networks people can form both positive and negative links. Positive links express trust, like or approval attitudes, whereas negative links indicate distrust, dislike or disapproval attitudes. For a given directed link from user u to v in a social network, we define its *sign* to be positive (or negative) if it expresses a positive (or negative) attitude from u to v. We call such networks with both positive and negative links *signed social networks*. Examples include Epinions[1] whose users can express trust or distrust of others [18], Slashdot[2] whose participants can declare others to be either "friends" or "foes" [11], and Wikipedia[3] whose users can vote for or against the promotion of others to administrator status [2].

In some signed social networks, the attitude of a link can be easily determined based on user rating score, i.e., positive or negative. But in some other cases such as in online forums or BBS, the existence of interactions (i.e., links) between two users can be easily observed while the specific semantic attitudes of these links are not explicitly labeled, since they are usually expressed implicitly by user reviews or comments. As a more general case, hyperlinks between webpages can also indicate agreement or disagreement with the target of the link, but the lack of explicit labels makes it very difficult to determine the attitude over these hyperlinks [20]. Manually analyzing and labeling the signs of links will be very expensive and inefficient. A promising solution is to train a classifier for link sign prediction in the signed networks. However for some signed social networks, especially the newly formed ones, the paucity of available signs makes it difficult to train a good classifier to predict unknown link signs. How to reliably and efficiently predict the signs of links in signed social networks is an important and challenging problem.

Previous research [15] has shown that, the structural information is a powerful and reliable source for the purpose of link prediction in unsigned networks, when applied in a traditional machine learning framework. Some examples of such structural information include the number of common neighbors, or other local neighborhood statistics. As for the signed social networks, a recent work by Leskovec et al. [14] studies the *edge sign prediction problem* using the signed triad features and a logistic regression model. While its major contribution is the connections to theories of balance and status in social psychology, the prediction model makes a very strong assumption on the input network: *the signs of all links except the one to be predicted are known in advance.* This is not very practical in reality as it is very expensive to obtain the signs of all links except one in a large network, especially for newly formed networks.

Thus, in this work we study the edge sign prediction prob-

WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2035-1/13/05.

[1] www.epinions.com
[2] slashdot.org
[3] www.wikipedia.org

lem with a more realistic setting as follows. Given a directed signed network, we are interested in predicting the signs of edges whose signs are unknown. We call this network a *target network*. We assume there is a very small amount of edge sign information in the target network as the training data, but the quantity is inadequate to train a good classifier. This assumption holds for many newly formed and fast evolving networks. Thus we consider to leverage another more mature signed social network, called a *source network*, which has abundant edge sign information. The source network may have a different joint distribution of the edge instances and the class labels from the target network, perhaps because the source network is out-dated or is from a different application. But the source network is not completely useless. There still exists a certain degree of similarity, e.g., similar degree distributions and diameters, or common properties, e.g., structural balance and social status [14], between the source and target networks. For example, according to the structural balance theory, many signed networks follow a common principle that "the friend of my friend is my friend" and "the enemy of my friend is my enemy". Thus our task is to leverage the sign information in both the source and target networks to train a good classifier. This approach is known as *transfer learning* [19, 1, 21].

While most existing transfer learning works focus on transactional data [1], image [21] and text [22], in which the data instances are represented in a predefined d-dimensional feature space, a unique challenge in our transfer learning problem across two signed social networks is that there is no predefined feature vector for the edge instances in the networks. Therefore, the first step is to investigate how to construct generalizable features that can transfer knowledge from the source network to the target for edge sign prediction. Specifically, we propose two types of features, i.e., *explicit topological features* which express the manifest properties of edge instances such as degree and triads, and *latent topological features* which capture the common patterns between the source and target networks for knowledge transfer.

With the extracted features, a straightforward solution is to simply combine the source and target training instances and treat them equally to learn a model. However, due to the distributional difference between the two networks, some training instances in the source network are very different from the target network, thus may cause test edges in the target network to be wrongly predicted and degrade the performance. Therefore, we adopt a transfer learning algorithm with instance weighting similar to [3]. This algorithm borrows the AdaBoost learning idea which assigns and iteratively adjusts the weight of each training instance in the source and target networks. This instance weighting mechanism can effectively distinguish the more useful edge instances from the less useful ones in the source network and attach more importance to the former.

Our main contributions are summarized as follows.

- We formulate the problem of edge sign prediction in a signed social network which may have only a very small amount of labeled training instances. We consider to exploit another network, called source network, which has abundant labeled instances. The source and target networks may have different yet related joint distributions. Our task is to leverage the source network instances for feature construction and model learning. To the best of our knowledge, this is the first work on transfer learning across large signed social networks.
- We design the latent topological features which can capture the common structural patterns between the source and target networks, thus are generalizable features across domains. The latent features are obtained by nonnegative matrix tri-factorization [4].
- We adopt an AdaBoost-like transfer learning algorithm with instance weighting to distinguish the more useful training instances from the less useful ones in the source network.
- We conducted extensive experiments on three real large signed networks and demonstrated that our transfer learning algorithm can improve the prediction accuracy by 40% over baseline schemes.

The rest of our paper is organized as follows. We introduce related work in Section 2 and give the problem definition in Section 3. We describe our proposed features for the edge sign prediction problem in Section 4. In Section 5 we propose an AdaBoost-like learning algorithm with instance weighting in the transfer learning framework. Experimental results are presented in Section 6 to show the effectiveness of our features and transfer learning algorithm. Finally, we conclude our work in Section 7.

2. RELATED WORK

In this section we first introduce related studies on signed social networks, and then the state-of-the-art transfer learning research.

Signed social networks have attracted more and more attention since Guha et al. proposed their leading work on trust propagation in signed social networks [6]. Kunegis et al. did spectral analysis on signed networks [11, 12]. They revealed fundamental characteristics of signed networks by evaluating various measures in [11]. They also studied signed spectral clustering methods, signed graph kernels and network visualization methods in signed graphs [12].

For the edge sign prediction problem in signed graphs, existing studies can be categorized into two major approaches: a *matrix kernel* approach [11] and a *machine learning* approach [14]. Kunegis et al. exploited the property of multiplicative transitivity in signed graphs to realize edge sign prediction and their method utilized the node adjacency information only. Leskovec et al. used signed triads as features and constructed a logistic regression model for prediction [14]. While its major contribution is the connections to theories of balance and status in social psychology, the prediction model makes a very strong assumption on the input network: the signs of all links except the one to be predicted are known in advance, which is not very practical in reality.

Transfer learning [19] has been an important research topic and a useful technique in practice. Transfer learning can effectively transfer the information from the source domain to facilitate a different target domain's learning task, where the labeled data in the target domain is very limited [1, 16]. Most existing transfer learning works focus on transactional data [1], image [21] and text [22], in which data instances are represented in a predefined d-dimensional feature space. They typically map data instances from different origins into the same latent domain [21] with sparse coding

or other dimension transformation techniques. Some transfer learning methods also use bipartite or tripartite graph as tools to facilitate knowledge transfer [7, 8] by mapping the instances and features to bipartite or tripartite graph nodes. To the best of our knowledge, our work is the first one on transferring topological knowledge across large signed social networks. Specifically, we transfer the knowledge of a source network to the target network for both feature construction and model learning, which have been shown to be very effective.

3. PROBLEM FORMULATION

Let a directed graph $G_t = (V_t, E_t^l, E_t^u, S)$ be the *target graph* for edge sign prediction. Here V_t denotes the set of vertices, E_t^l denotes the set of directed edges with edge sign labels, $S : E_t^l \mapsto \{-1, +1\}$ is the edge sign mapping function that maps an edge $e \in E_t^l$ to a positive label $(+1)$ or a negative label (-1), and E_t^u denotes the set of directed edges whose signs are unknown and need to be predicted. We treat the labeled edges E_t^l as the training data which is an independent and identically distributed (i.i.d.) sample drawn from the target graph. Thus E_t^l has the same distribution as the test edge set E_t^u. However, in many scenarios the quantity of the training edges is inadequate to train a good classifier.

Assume that we have another directed graph $G_s = (V_s, E_s, S)$, called the *source graph*, where V_s denotes the set of vertices, E_s denotes the set of directed edges, and $S : E_s \mapsto \{-1, +1\}$ is the edge sign mapping function that maps an edge $e \in E_s$ to a positive or negative label. The labeled edges E_s are assumed to be abundant, but the distribution of E_s may differ from that of the test edge set in the target graph G_t, perhaps because G_s is out-dated, or is from a different domain. When a classifier trained on E_s is applied to the test edge set E_t^u from G_t, the performance of the classifier may substantially degrade.

However, the labeled edges E_s from the source graph G_s is not entirely useless, because there may still exist a certain degree of similarity or common properties between the source graph G_s and the target graph G_t. Considering the inadequate labeled edges E_t^l from G_t, it is important and beneficial to leverage the labeled edges E_s from G_s to help train a classifier to predict the edge signs of E_t^u in G_t.

Formally, let $T = T_s \cup T_t$ denote the training edge set. $T_s = \{(e_s, S(e_s)\}, \forall e_s \in E_s$, and $T_t = \{(e_t, S(e_t)\}, \forall e_t \in E_t^l$. We denote $|T_s| = n$ and $|T_t| = m$. E_t^u is the unlabeled test edge set. The objective is to learn a classifier $P : E_s \cup E_t^l \mapsto \{-1, +1\}$ that minimizes the prediction error on the test edge set E_t^u.

In the following, we will first study *feature construction* to create useful topological features that are generalizable from the source graph G_s to the target G_t. Then we will study *model learning* that uses an AdaBoost-like method to weigh the training edges from the source graph and the target graph differently, for the transfer learning purpose.

4. FEATURE CONSTRUCTION

In this section, we study how to construct useful features for edge sign prediction. Different from many traditional machine learning or transfer learning problems in which instances are represented in a *predefined* feature vector, there is no predefined feature vector for the edge instances in a signed social network. Therefore the problem is how to construct topological features for the edge instances that are generalizable from the source graph G_s to the target graph G_t. In this work, we propose to create a collection of features from two categories: (1) *explicit topological features* which express manifest properties of the edge instances in the source or target graph; and (2) *latent topological features* which are hidden but express the common patterns between the source graph and the target graph. Such latent topological features are generalizable across domains in principle.

4.1 Explicit Topological Features

For a directed edge $e = (u, v)$, we begin by defining a number of explicit topological features including node degree, betweenness centrality, triad count and edge embeddedness. These features express the connectivity pattern of the edge e and its two end nodes u and v. It is noteworthy that we *do not* include any edge sign information in the features, because we only have a very small amount of edge signs in the target graph G_t. When predicting the sign of an edge $e \in E_t^u$, it is practically infeasible to use the signs of edges in the local neighborhood of e as features, as those signs can be unknown.

Node Degree. For a directed edge $e = (u, v)$, we use $deg_{out}(u)$ and $deg_{in}(v)$ to denote the number of outgoing edges from u and incoming edges to v, respectively. The node degree measures the aggregate connection strength of a node to the rest of a graph.

Betweenness Centrality. Betweenness centrality measures a node's centrality in a graph. For a node $v \in V$, the betweenness centrality $f_{bc}(v)$ is defined as

$$f_{bc}(v) = \sum_{i \neq v \neq j} \frac{\sigma_{i,j}(v)}{\sigma_{i,j}} \tag{1}$$

where $\sigma_{i,j}$ is the number of shortest paths from node i to j and $\sigma_{i,j}(v)$ is the number of shortest paths from node i to j through node v. Here all possible node pairs $i, j \in V$ such that $i \neq v \neq j$ are considered. For a directed edge $e = (u, v)$, we use $f_{bc}(u)$ and $f_{bc}(v)$ as two betweenness centrality features.

Triad Count. Following [14], we also use the triad counts as features. For a directed edge $e = (u, v)$, we consider each triad involving the edge (u, v), consisting of a node w such that w has an edge either to or from u and also an edge either to or from v. Considering that the edge between u (or v) and w can be in either direction, there are $2 \times 2 = 4$ types of triads. For an edge between u and w, we call it a *forward edge* (F) if it points from u to w, or a *backward edge* (B) otherwise. Similarly, for an edge between v and w, we call it a *forward edge* if it points from w to v, or a *backward edge* otherwise. Figure 1 shows the four types of triads involving (u, v). We use four features f_{FF}, f_{FB}, f_{BF} and f_{BB} to record the number of triads of each type that the edge (u, v) is involved in.

Edge Embeddedness. For a directed edge $e = (u, v)$, the edge embeddedness [14] $f_{eb}(e)$ is defined as the number of common neighbors of nodes u and v. $f_{eb}(e)$ is also used as one topological feature.

4.2 Latent Topological Feature

The above explicit topological features, though very intuitive, may not be generalizable from the source graph G_s to the target graph G_t, especially when G_s and G_t have dif-

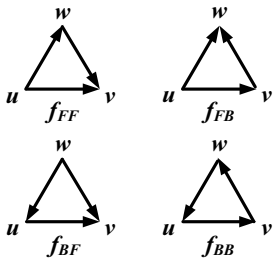


Figure 1: Four Types of Triads

ferent distributions. To better leverage the labeled edges in the source graph, we propose to construct *latent topological features* which can capture the common patterns between G_s and G_t, and thus are generalizable features. Again, it is worth noting that only a very small amount of edge signs are available in the target graph G_t, so it is not practical to include the edge signs in the latent topological features. Therefore, we only use the connectivity information of both graphs without edge signs to construct the latent features.

Specifically, denote the source graph without edge signs as $\overline{G}_s = (V_s, E_s)$ with $|V_s| = M$. Let $A_s \in \{0,1\}^{M\times M}$ denote the $M \times M$ adjacency matrix of $\overline{G}_s$. For a pair of vertices u, v, $A_{s(u,v)} = 1$ iff $(u, v) \in E_s$, and $A_{s(u,v)} = 0$ otherwise. A_s is asymmetric since $\overline{G}_s$ is a directed graph. Similarly, let $A_t \in \{0,1\}^{N\times N}$ denote the $N \times N$ adjacency matrix of the unsigned target graph $\overline{G}_t$ where $|V_t| = N$.

Given A_s and A_t, we propose to use *Nonnegative Matrix Tri-Factorization* (**NMTF**) [4], which originates from Nonnegative Matrix Factorization (**NMF**) [13] and has been widely applied and extended in machine learning [4, 17], to construct latent topological features through factorizing A_s and A_t under the same space. The latent feature space can capture the principal common factors of the original link structures in both G_s and G_t. We formulate the problem of finding the latent feature space as follows:

$$\begin{aligned}\min \mathcal{J} &= \|A_s - U_s\Sigma_k V_s^T\|_F^2 + \|A_t - U_t\Sigma_k V_t^T\|_F^2 \\ &+ \alpha\|\Sigma_k\|_F^2, \\ \text{s.t.} \quad & \sum_{j=1}^{k} U_{s(\cdot j)} = 1, \ \sum_{j=1}^{k} V_{s(\cdot j)} = 1, \\ & \sum_{j=1}^{k} U_{t(\cdot j)} = 1, \ \sum_{j=1}^{k} V_{t(\cdot j)} = 1, \\ & U_s, V_s \in \mathbb{R}_+^{M\times k}, \ U_t, V_t \in \mathbb{R}_+^{N\times k} \end{aligned} \tag{2}$$

where $\|\cdot\|_F$ is the Frobenius norm. In this formulation, we consider the nonnegative matrix tri-factor decomposition $A_s \approx U_s\Sigma_k V_s^T$ and $A_t \approx U_t\Sigma_k V_t^T$. Matrix $\Sigma_k \in \mathbb{R}_+^{k\times k}$ is the common latent space for both graphs, which ensures that the extracted topological features of both graphs are expressed in the same space. U_s, V_s, U_t, and V_t are four latent topological feature matrices. The i^{th} row of matrix U_s represents the outgoing linkage features of a source graph node i ($i \in V_s$) in the latent space, while the i^{th} row of matrix V_s represents the incoming linkage features of node i in the latent space. Similar interpretations can be derived for matrices U_t and V_t in the target graph G_t. α is the trade-off regularization parameter to weigh the term $\|\Sigma_k\|_F^2$. Since all of our variables are nonnegative, excessively large values in Σ_k will make many entries in U_s, U_t, V_s and V_t approach 0. As a result, this will cause each node to have nearly indistinguishable latent topological features. Thus it is necessary to constraint the values in Σ_k through a regularization term. In addition, we enforce the constraints that each row in the topological feature matrices is positive and normalized.

By solving the joint NMTF problem in Eq. 2, we obtain four feature matrices U_s, V_s, U_t and V_t for expressing the latent topological (both outgoing and incoming) features for nodes in G_s and G_t in terms of the same latent space Σ_k.

4.2.1 Optimization Algorithm

We develop an iterative update algorithm to minimize $\mathcal{J}$ in Eq. 2. Specifically, we optimize the objective function in Eq. 2 by updating one variable while fixing the other variables. We can rewrite the objective function in Eq. 2 as follows.

$$\begin{aligned}\mathcal{J} &= \mathbf{tr}(A_s^T A_s - 2A_s^T U_s\Sigma_k V_s^T + V_s\Sigma_k^T U_s^T U_s\Sigma_k V_s^T) \\ &+ \mathbf{tr}(A_t^T A_t - 2A_t^T U_t\Sigma_k V_t^T + V_t\Sigma_k^T U_t^T U_t\Sigma_k V_t^T) \\ &+ \alpha\mathbf{tr}(\Sigma_k^T\Sigma_k) \end{aligned} \tag{3}$$

Update Rule of U_s: Since we have the constraint $U_s \geq 0$, following the standard constrained optimization theory, we introduce the Lagrangian multiplier $\mathcal{L}_{U_s} \in \mathbb{R}^{M\times k}$ and minimize the Lagrangian function

$$L(U_s) = \mathcal{J} - \mathbf{tr}(\mathcal{L}_{U_s}U_s) \tag{4}$$

We set $\frac{\partial L(U_s)}{\partial U_s} = 0$. Then considering the KKT condition $\mathcal{L}_{U_s(i,j)}U_{s(i,j)} = 0$, we can get

$$(-2A_sV_s\Sigma_k^T + 2U_s\Sigma_k V_s^T V_s\Sigma_k^T)_{(i,j)}U_{s(i,j)} = 0 \tag{5}$$

Based on Eq. 5, following a similar approach as in [13], we have the following multiplicative update rule

$$U_{s(i,j)} \leftarrow U_{s(i,j)}\sqrt{\frac{(A_sV_s\Sigma_k^T)_{(i,j)}}{(U_s\Sigma_k V_s^T V_s\Sigma_k^T)_{(i,j)}}} \tag{6}$$

Update Rules of Other Matrices: Similar to the update rule of U_s, the update rules of V_s, U_t, V_t, and Σ_k are

$$V_{s(i,j)} \leftarrow V_{s(i,j)}\sqrt{\frac{(A_s^T U_s\Sigma_k)_{(i,j)}}{(V_s\Sigma_k^T U_s^T U_s\Sigma_k)_{(i,j)}}} \tag{7}$$

$$U_{t(i,j)} \leftarrow U_{t(i,j)}\sqrt{\frac{(A_tV_t\Sigma_k^T)_{(i,j)}}{(U_t\Sigma_k V_t^T V_t\Sigma_k^T)_{(i,j)}}} \tag{8}$$

$$V_{t(i,j)} \leftarrow V_{t(i,j)}\sqrt{\frac{(A_t^T U_t\Sigma_k)_{(i,j)}}{(V_t\Sigma_k^T U_t^T U_t\Sigma_k)_{(i,j)}}} \tag{9}$$

$$\Sigma_{k(i,j)} \leftarrow \Sigma_{k(i,j)}\sqrt{\frac{(U_s^T A_sV_s + U_t^T A_tV_t)_{(i,j)}}{(U_s^T U_s\Sigma_k V_s^T V_s + U_t^T U_t\Sigma_k V_t^T V_t + \alpha\Sigma_k)_{(i,j)}}} \tag{10}$$

Algorithm 1 is the iterative update algorithm that uses the above multiplicative rules for updating each variable matrix to optimize Eq. 2. The convergence criterion is, the gap between any two consecutive objective function values of Eq. 2 is less than a certain threshold.

4.2.2 Convergence Analysis

We now study the convergence property of Algorithm 1. First, we give the following two lemmas from [13, 4].

Algorithm 1: Iterative Update Algorithm for Eq. 2

Data: [Adjacency Matrices A_s, A_t; Regularizer α]
Result: [Latent feature matrices U_s, V_s, U_t, V_t; Latent space Σ_k]
begin
  Initialize U_s, V_s, U_t, V_t, Σ_k following [23]
  while *beyond convergence* **do**
    1. Update U_s according to Eq. 6
    2. Update V_s according to Eq. 7
    3. Update U_t according to Eq. 8
    4. Update V_t according to Eq. 9
    5. Update Σ_k according to Eq. 10
    6. Normalize each row of U_s, V_s, U_t, V_t
end

LEMMA 1. *[13] $Aux(h, h')$ is an auxiliary function for $F(h)$ if the conditions $Aux(h, h') \geq F(h)$, $Aux(h, h) = F(h)$ are satisfied. If Aux is an auxiliary function for F, then F is non-increasing under the update*

$$h^{t+1} = \arg\min_h Aux(h, h^t)$$

where h^t is the value of variable h in the t^{th} iteration.

LEMMA 2. *[4] For any matrices $P \in \mathbb{R}_+^{N\times N}$, $Q \in \mathbb{R}_+^{k\times k}$, $S \in \mathbb{R}_+^{N\times k}$, $S' \in \mathbb{R}_+^{N\times k}$, and P and Q are symmetric, the following inequality holds*

$$\sum_{i,j} \frac{(PS'Q)_{i,j} S_{i,j}^2}{S'_{i,j}} \geq \mathbf{tr}(S^T PSQ) \tag{11}$$

THEOREM 1. *Define $\mathcal{J}(U_s)$ according to Eq. 2 as*

$$\mathcal{J}(U_s) = \mathbf{tr}(-2A_s^T U_s \Sigma_k V_s^T + V_s \Sigma_k^T U_s^T U_s \Sigma_k V_s^T) \tag{12}$$

then the following function

$$\begin{aligned} Aux(U_s, U'_s) &= -2\sum_{i,j} (A_s V_s \Sigma_k^T)_{(i,j)} U'_{s(i,j)} (1 + \ln \frac{U_{s(i,j)}}{U'_{s(i,j)}}) \\ &+ \sum_{i,j} (U'_s \Sigma_k V_s^T V_s \Sigma_k^T)_{(i,j)} \frac{U_{s(i,j)}^2}{U'_{s(i,j)}} \end{aligned} \tag{13}$$

is an auxiliary function for $\mathcal{J}(U_s)$. Furthermore, $Aux(U_s, U'_s)$ is convex for U_s and its global minimum can be achieved at

$$U_{s(i,j)} = U'_{s(i,j)} \sqrt{\frac{(A_s V_s \Sigma_k^T)_{(i,j)}}{(U'_s \Sigma_k V_s^T V_s \Sigma_k^T)_{(i,j)}}} \tag{14}$$

PROOF. First, in Lemma 2, let $P = I$, $Q = \Sigma_k V_s^T V_s \Sigma_k^T$, $S = U_s$ and $S' = U'_s$. Then we have

$$\begin{aligned} \mathbf{tr}(S^T PSQ) &= \mathbf{tr}(U_s^T \cdot I \cdot U_s \cdot \Sigma_k V_s^T V_s \Sigma_k^T) \\ &= \mathbf{tr}(U_s^T U_s \Sigma_k V_s^T \cdot V_s \Sigma_k^T) \\ &\leq \sum_{i,j} (U'_s \Sigma_k V_s^T V_s \Sigma_k^T)_{(i,j)} \frac{U_{s(i,j)}^2}{U'_{s(i,j)}} \end{aligned}$$

Based on the property of trace, we have

$$\mathbf{tr}(U_s^T U_s \Sigma_k V_s^T \cdot V_s \Sigma_k^T) = \mathbf{tr}(V_s \Sigma_k^T \cdot U_s^T U_s \Sigma_k V_s^T)$$

Therefore, we can derive

$$\mathbf{tr}(V_s \Sigma_k^T U_s^T U_s \Sigma_k V_s^T) \leq \sum_{i,j} (U'_s \Sigma_k V_s^T V_s \Sigma_k^T)_{(i,j)} \frac{U_{s(i,j)}^2}{U'_{s(i,j)}} \tag{15}$$

Second, since $\forall z > 0$, $z \geq 1 + \ln z$, we have

$$\begin{aligned} \mathbf{tr}(A_s^T U_s \Sigma_k V_s^T) &= \sum_{i,j} (A_s V_s \Sigma_k^T)_{(i,j)} U_{s(i,j)} \\ &\geq \sum_{i,j} (A_s V_s \Sigma_k^T)_{(i,j)} U'_{s(i,j)} (1 + \ln \frac{U_{s(i,j)}}{U'_{s(i,j)}}) \end{aligned} \tag{16}$$

If we multiply both sides of Eq. 16 with -2, we get

$$\mathbf{tr}(-2A_s^T U_s \Sigma_k V_s^T) \leq -2\sum_{i,j} (A_s V_s \Sigma_k^T)_{(i,j)} U'_{s(i,j)} (1 + \ln \frac{U_{s(i,j)}}{U'_{s(i,j)}}) \tag{17}$$

When adding Eqs. 15 and 17 on both sides, we have

$$\begin{aligned} \mathcal{J}(U_s) &= \mathbf{tr}(-2A_s^T U_s \Sigma_k V_s^T + V_s \Sigma_k^T U_s^T U_s \Sigma_k V_s^T) \\ &\leq -2\sum_{i,j} (A_s V_s \Sigma_k^T)_{(i,j)} U'_{s(i,j)} (1 + \ln \frac{U_{s(i,j)}}{U'_{s(i,j)}}) \\ &\quad + \sum_{i,j} (U'_s \Sigma_k V_s^T V_s \Sigma_k^T)_{(i,j)} \frac{U_{s(i,j)}^2}{U'_{s(i,j)}} \\ &= Aux(U_s, U'_s) \end{aligned}$$

In addition, it is easy to verify that $Aux(U_s, U_s) = \mathcal{J}(U_s)$. Therefore, we prove $Aux(U_s, U'_s)$ defined in Eq. 13 is the auxiliary function for $\mathcal{J}(U_s)$.

Next, if we fix U'_s and minimize $Aux(U_s, U'_s)$ w.r.t. each $U_{s(i,j)}$, we get

$$\begin{aligned} \frac{\partial Aux(U_s, U'_s)}{\partial U_{s(i,j)}} &= -2(A_s V_s \Sigma_k^T)_{(i,j)} \frac{U'_{s(i,j)}}{U_{s(i,j)}} \\ &+ 2(U'_s \Sigma_k V_s^T V_s \Sigma_k^T)_{(i,j)} \frac{U_{s(i,j)}}{U'_{s(i,j)}} \end{aligned} \tag{18}$$

Moreover, the second partial derivative (*Hessian* matrix) is

$$\begin{aligned} \frac{\partial^2 Aux(U_s, U'_s)}{\partial U_{s(i,j)} \partial U_{s(k,l)}} &= \delta_{ik}\delta_{jl} (2(A_s V_s \Sigma_k^T)_{(i,j)} \frac{U'_{s(i,j)}}{U_{s(i,j)}^2} \\ &+ \frac{2(U'_s \Sigma_k V_s^T V_s \Sigma_k^T)_{(i,j)}}{U'_{s(i,j)}}) \end{aligned} \tag{19}$$

We should notice that *Hessian* of $Aux(U_s, U'_s)$ w.r.t. U_s is a diagonal matrix with all positive diagonal elements. Thus, $Aux(U_s, U'_s)$ is convex over U_s and we can achieve its global minimum through setting Eq. 18 to be 0, which leads to the result in Eq. 14. □

Let $U'_{s(i,j)} = U^t_{s(i,j)}$, then according to the update $h^{t+1} = \arg\min_h Aux(h, h^t)$ in Lemma 1, $U_{s(i,j)}$ defined in Eq. 14 which minimizes $Aux(U_s, U^t_{s(i,j)})$ is exactly $U^{t+1}_{s(i,j)}$. This is essentially our update rule for U_s in Eq. 6. This leads to the following lemma.

LEMMA 3. *Using the update rule in Eq. 6 to update U_s, $\mathcal{J}(U_s)$ in Eq. 12 is monotonically decreasing.*

PROOF. By Lemma 1 and Theorem 1, we have

$$\mathcal{J}(U_s^0) = Aux(U_s^0, U_s^0) \geq Aux(U_s^1, U_s^0) \geq \mathcal{J}(U_s^1) \geq \cdots$$

where U_s^t is the value of matrix U_s in the t^{th} iteration. Therefore, $\mathcal{J}(U_s)$ is monotonically decreasing. □

THEOREM 2. *Using Algorithm 1 to update U_s, V_s, U_t, V_t and Σ_k, the value of the objective function $\mathcal{J}$ will monotonically decrease.*

The proof of Theorem 2 can be similarly achieved through Lemma 3. Since the objective function value $\mathcal{J}$ in Eq. 2 is lower bounded by 0, Algorithm 1 can guarantee convergence by Theorem 2.

Theoretically, the computational complexity of factorizing Eq. 2 is at most $\mathcal{O}(k \cdot \max\{|E_s|, |E_t|\}^2)$ where k is the length of the latent topological feature vector, and $E_t = E_t^l \cup E_t^u$. In practice, due to the sparsity of adjacency matrices A_s and A_t, the exact computational cost can be much lower than the theoretical result.

So far we have constructed both explicit and latent topological features for edge sign prediction. For an edge instance $e = (i, j) \in E_t^l$ with label $S(e)$, we have latent feature vectors $U_{t(i\cdot)}$ and $V_{t(j\cdot)}$ to represent node i's outgoing linkage pattern and node j's incoming linkage pattern. We also have 9 explicit features, including node degrees $deg_{out}(i)$ and $deg_{in}(j)$, betweenness centrality $f_{bc}(i)$ and $f_{bc}(j)$, triad counts $f_{FF}(e)$, $f_{FB}(e)$, $f_{BF}(e)$, $f_{BB}(e)$, and edge embeddedness $f_{eb}(e)$. We can similarly define features for edge instances in E_s. The features are used together, denoted as $F(e)$, for learning an edge sign prediction model.

5. EDGE SIGN PREDICTION BY TRANSFER LEARNING

With the explicit and latent topological features, it is natural to learn a model from the training instances in the target graph, i.e., $T_t = \{(e_t, S(e_t))\}$, $\forall e_t \in E_t^l$. However, using only the small amount of labeled edge instances for training does not give a classifier with good prediction performance on the target graph.

Fortunately, we still have the full knowledge of the edge signs in the source graph G_s. Thus our task is to learn an edge sign prediction model by leveraging the labeled instances in both the source and target graphs. With our extracted features, the source and target graph edge instances can be represented in the same feature space. A straightforward approach is to simply combine the source and target edge instances and treat them equally to learn a model.

However, since discrepancy always exists between the distribution of source and target graph edges, a classifier learned from this simple combination may not necessarily achieve better performance than the model learned from the target graph edges only. Sometimes the noise in the source graph instances may cause the model to predict wrongly on the test edges from the target graph, thus degrade the performance substantially. Thus we need an effective mechanism to distinguish the more useful edge instances from the less useful ones in the source graph.

5.1 Transfer Learning with Instance Weighting

To address the distributional difference issue, we borrow the AdaBoost idea from Freund and Schapire [5] and Dai et al. [3] for *instance weighting* in our transfer learning framework. That is, we treat the edge instances in $T_s = \{(e_s, S(e_s))\}$, $\forall e_s \in E_s$ and $T_t = \{(e_t, S(e_t))\}$, $\forall e_t \in E_t^l$ differently, by assigning and adjusting the weight of each training instance during model learning. For those edge instances in T_s that are more similar to the target edge instances in T_t, we should give them larger weights to attach more importance to them; conversely, for those edge instances in T_s that are less similar to the target edge instances in T_t, we should give smaller weights to weaken their impacts.

Algorithm 2 shows an iterative algorithm which updates instance weights according to the basic classifier P_t's performance in each round. It is similar to the traditional AdaBoost method where the accuracy of a learner is boosted by carefully adjusting instance weights. We use $w_1, \ldots, w_n$ to denote the weights of edges in T_s, and $w_{n+1}, \ldots, w_{n+m}$ to denote the weights of edges in T_t. It is worth noting the following special instance weighting policy in our transfer learning framework. For an edge e, $P_t(e) \in [-1, 1]$ is the predicted edge sign for e and $S(e)$ is the true edge sign. For any target graph edge $e_t \in E_t^l$, its weight will always get increased by a factor of $\beta_t^{-\frac{|P_t(e_t)-S(e_t)|}{2}} \in [1, +\infty)$, and the weight increment of a wrongly predicted edge is larger than that of a correctly predicted one. In contrast, for any source graph edge $e_s \in E_s$, its weight will always get decreased by a factor of $\beta^{\frac{|P_t(e_s)-S(e_s)|}{2}} \in (0, 1]$, and the weight decrement of a wrongly predicted edge is larger than that of a correctly predicted one, because the wrongly predicted source graph edge may be very dissimilar to the target graph. Therefore, the weights of source graph edges would never increase and are always less than those of target graph edges, which means that the source graph edges will never have a larger influence than the target graph edges in model learning. After K iterations, those source graph edges which are more similar to the target graph edges will have larger weights than the less similar ones to contribute to model learning.

5.2 Training Loss Analysis

We analyze the training loss from both source and target graphs, based on the analysis in Freund and Schapire [5] and Dai et al. [3]. Consider the t^{th} iteration training loss on source graph instances where each instance's normalized loss is defined as $l_t(e_i) = |P_t(e_i) - S(e_i)|/2$, and its overall training loss through K iterations is $\mathcal{L}_i = \sum_{t=1}^{K} l_t(e_i)$, $1 \leq i \leq n$. Thus all source instances' training loss suffered by Algorithm 2 is

$$\Upsilon = \sum_{t=1}^{K} \sum_{i=1}^{n} d_i^t l_t(e_i)$$

where $d_i^t = w_i^t / (\sum_{j=1}^{n} w_j^t)$. We first present the following conclusion.

THEOREM 3. *In Algorithm 2, we have*

$$\frac{\Upsilon}{K} \leq \min_{1 \leq i \leq n} \frac{\mathcal{L}_i}{K} + \sqrt{\frac{2 \ln n}{K}} + \frac{\ln n}{K} \tag{20}$$

Theorem 3 and its proof can be found in [5]. It can rigidly bound the average training loss of source graph instances through K iterations, which cannot exceed the minimum average training loss of a single instance by more than $\sqrt{\frac{2 \ln n}{K}} + \frac{\ln n}{K}$.

Similar to [3], we have the following theorem.

Algorithm 2: Transfer Learning & Instance Weighting

Data: source edge instances T_s, labeled target edge instances T_t, and the iteration number K
Result: edge sign classifier P
begin
Let $n \leftarrow |T_s|, m \leftarrow |T_t|$
Initialize the weight vector
$\mathbf{w}^1 = (w_1^1, \ldots, w_n^1, w_{n+1}^1, \ldots, w_{n+m}^1)$
for $t = 1, \ldots, K$ **do**
1. $\mathbf{q}^t \leftarrow \mathbf{w}^t / (\sum_{i=1}^{n+m} w_i^t)$
2. Call a basic learner on $T_s \cup T_t$ with $\mathbf{q}^t$ to learn a model $P_t : F(e) \rightarrow P_t(e) \in [-1, 1]$
3. Calculate the error of P_t on T_t

$$\epsilon_t = \frac{\sum_{i=n+1}^{n+m} q_i^t \cdot \frac{|P_t(e_i) - S(e_i)|}{2}}{\sum_{i=n+1}^{n+m} q_i^t}$$

4. Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$, $\beta = \frac{1}{1+\sqrt{2 \ln n / K}}$
5. Update weight vector $\mathbf{w}^t$

$$w_i^{t+1} = \begin{cases} w_i^t \beta^{\frac{|P_t(e_i) - S(e_i)|}{2}}, & 1 \leq i \leq n \\ w_i^t \beta_t^{-\frac{|P_t(e_i) - S(e_i)|}{2}}, & n+1 \leq i \leq n+m \end{cases}$$

$$P(e) = \begin{cases} 1, & \text{if } \sum_{t=1}^{K} \log \frac{1}{\beta_t} \cdot P_t(e) \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

end

THEOREM 4. *In Algorithm 2, q_i^t denotes the weight of the training instance e_i, which is defined as $\mathbf{q}^t = \mathbf{w}^t / (\sum_{i=1}^{n+m} w_i^t)$. Then,*

$$\lim_{K \rightarrow \infty} \frac{\sum_{t=1}^{K} \sum_{i=1}^{n} q_i^t l_t(e_i)}{K} = 0 \tag{21}$$

Theorem 4 shows that the weighted average training loss in the source graph edge instances gradually converges to zero.

Next, according to Step 4 in Algorithm 2, we have the constraint $\epsilon_t \leq 1/2 - \gamma$, for some $\gamma > 0$. Then we have the following bound on the prediction error of the final classifier on the labeled target edge instances.

THEOREM 5. *Let $\mathcal{I} = \{i : P(e_i) \neq S(e_i), e_i \in T_t\}$. Define the error of the final classifier P by Algorithm 2 as $\epsilon = \mathbf{Pr}_{e \in T_t}[P(e) \neq S(e)] = |\mathcal{I}|/|T_t|$ and it is bounded as*

$$\epsilon \leq \exp\{-2 \cdot K\gamma^2\}. \tag{22}$$

Theorem 5 and its proof can be found in [5]. From Theorem 5, we can observe that the final classifier P will reduce the error on target graph labeled instances, when the maximum number of iterations K increases. Therefore Algorithm 2 minimizes both the error on the target graph training instances and the weighted average loss on the source graph training instances simultaneously.

6. EXPERIMENTAL EVALUATION

In this section, we first describe how we prepare data for training and testing. Then we present experimental results to show the effectiveness of our proposed features and the transfer learning algorithm. Studies on the convergence properties of both Algorithms 1 and 2 are also provided.

6.1 Data Preparation

We use three large online social networks *Epinions*, *Slashdot* and *Wikipedia* where each link is explicitly labeled as positive or negative. All three networks are downloaded from Stanford Large Network Dataset Collection[4]. Since the original graphs are too large and sparse, we select 20,000 nodes from Epinions, 16,000 nodes from Slashdot and 7,000 nodes from Wikipedia with the highest degrees, as well as the edges between the selected nodes. There are 13 nodes in Epinions, 1 node in Slashdot and 2 nodes in Wikipedia that are disconnected from the remaining selected nodes. These isolated nodes are removed from the respective network and the remaining ones form a connected component. Table 1 shows the statistics of the three extracted networks.

Table 1: Statistics of Extracted Graphs

	Epinions	Slashdot	Wikipedia
Number of Nodes	19,987	15,999	6,998
Number of Edges	634,215	371,122	121,151
Average Degree	31.731	23.197	17.312
Positive Edges	87.6%	76.5%	71.3%
Average Distance	3.163	3.569	4.014

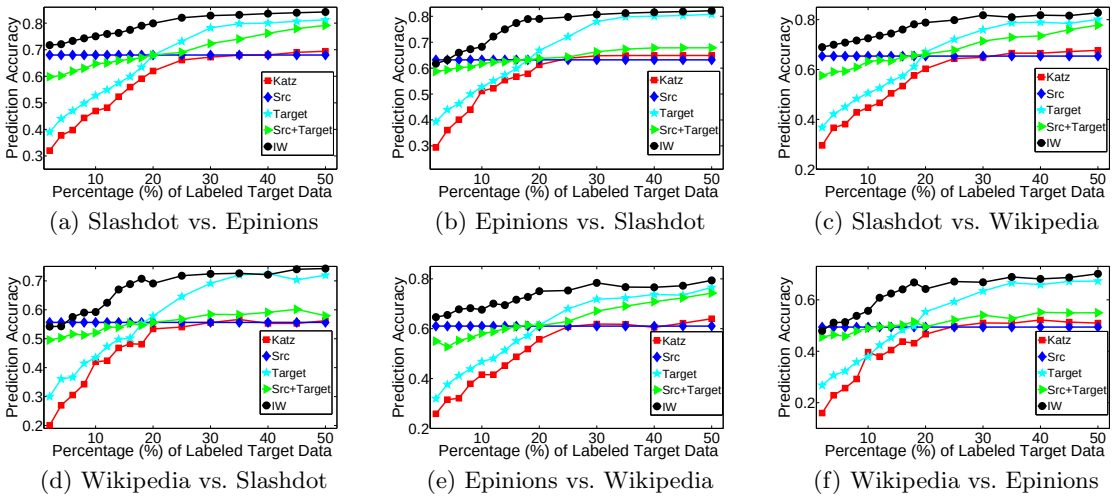
We can observe that Epinions has the largest number of nodes, edges, average degree and the percentage of positive edges among the three networks, while Wikipedia has the smallest number of nodes, edges, average degree and the percentage of positive edges. The statistics demonstrate that there indeed exists discrepancy in data distribution among the three networks.

As the edge signs in all these networks are overwhelmingly positive, we follow the methodology adopted by both Guha et al. [6] and Leskovec et al. [14], to create a balanced dataset from each signed social network. For every negative edge, we sample a random positive edge to ensure that the number of positive and negative edges is equal. We consider each pair of networks out of the three and use one network as the source and the other as the target for transfer learning, and then reverse the source and the target networks. There are totally $\binom{3}{2} \times 2 = 6$ $(source, target)$ pairs to test. Moreover, we use 4-fold cross validation for performance evaluation – in each target graph we partition the edge instances into four parts evenly, each having a balanced class distribution. We use one part as the test edge set E_t^u, and randomly sample a small percentage of edge instances in the remaining three parts to form the labeled edge set E_t^l. This small E_t^l and all edges E_s in the source graph form the training set. We run our experiment on the four test edge sets in turn and report the average classification accuracy.

6.2 Evaluation of Transfer Learning with Instance Weighting

Comparison with Other Approaches: We compare the prediction accuracy of the following methods.

[4]http://snap.stanford.edu/data/index.html



(a) Slashdot vs. Epinions (b) Epinions vs. Slashdot (c) Slashdot vs. Wikipedia

(d) Wikipedia vs. Slashdot (e) Epinions vs. Wikipedia (f) Wikipedia vs. Epinions

Figure 2: Prediction Accuracy by Varying the Percentage of Labeled Target Edge Instances

- *Katz*: *Katz* kernel [10] is a matrix kernel approach used in [11, 9] for edge sign prediction. It uses the adjacency matrix of labeled edges in the target graph to predict the sign of the test edges.
- *Src*: using all edge instances in the source graph only for training.
- *Target*: using labeled edge instances in the target graph only for training.
- *Src+Target*: using both source graph edges and labeled target graph edges equally for training without instance weighting.
- *IW*: using both source graph edges and labeled target graph edges for training with instance weighting (our Algorithm 2).

For all the above schemes we use SVM as the classification model and use the same test edge set for prediction. All schemes except *Katz* use our proposed explicit and latent topological features. We set $k = 30$ for matrix Σ_k representing the latent feature space.

In the first group of experiment, we use Slashdot as the source and Epinions as the target. We vary the percentage of labeled edge instances in the target graph from 2% to 50% for training and report the classification accuracy. The results are shown in Figure 2(a). We can observe that when the percentage of labeled target edge instances increases, the accuracy of all methods except *Src* increases and gradually saturates. The accuracy of *Target* improves greatly when the percentage increases from 2% to 20% and it outperforms *Src* when the percentage exceeds 20%. The performance of *Src+Target* without instance weighting lies between that of *Src* and *Target*. When the amount of labeled target edge instances is very small, *Src+Target* can improve the accuracy over *Target* by leveraging source graph edge instances; but when the labeled target edge instances are abundant, the source edge instances become less useful, and the noise in the source edge instances may become more obvious, causing *Target* to be better than *Src+Target*. *Katz* performs the worst among all methods. Our proposed method *IW* (Algorithm 2) achieves the highest accuracy in all cases, demonstrating the effectiveness of instance weighting in the transfer learning framework. *IW* consistently outperforms *Target* even if we have 50% labeled target edge instances for training. This result proves that the knowledge transferred from the source graph is beneficial to improve the model's performance, under proper instance weighting.

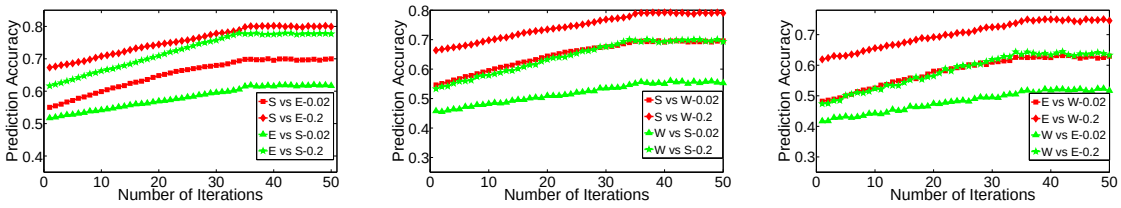
We can observe similar trends in Figure 2(b) in the second group of experiment when Epinions is the source and Slashdot is the target. But due to the larger number of edges in Epinions, the source edge instances have a larger influence in model learning. This effect causes *Src+Target* to have a very close performance to *Src*.

Similar conclusions can be drawn in Figures 2(c)–2(f) for the other four groups of experiments. Another important observation is that, when the distributional differences between the source and target networks become larger, the transfer learning performance becomes worse. For example, the differences between Wikipedia and Epinions as shown in Table 1 are larger than the differences between Slashdot and Epinions. Consider the case when Epinions is the target network. The prediction accuracy when using Wikipedia as the source (Figure 2(f)) is lower than the accuracy when using Slashdot as the source (Figure 2(a)), due to the larger differences between Wikipedia and Epinions.

Learning Convergence Analysis: It is important to assess the convergence of Algorithm 2 as an iterative algorithm. Besides the theoretical analysis of convergence in Section 5, we also test the learning convergence in all our experiments. Figure 3 shows the prediction accuracy in each iteration on all the $\binom{3}{2} = 3$ pairs of networks. Legend "S vs E-0.02" means using Slashdot as the source and Epinions as the target with 2% labeled target edges for training. We can see that the accuracy gradually increases with more iterations and converges after 30–35 iterations. This result confirms the theoretical analysis on learning convergence.

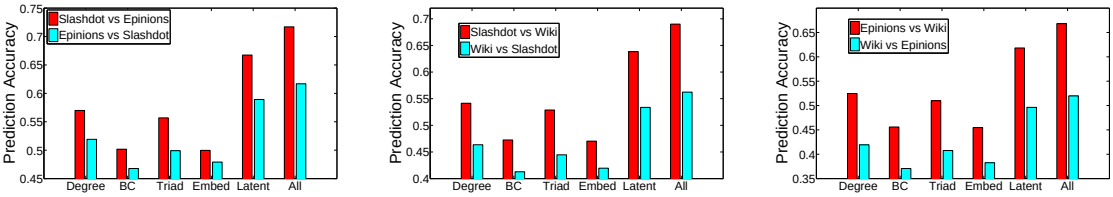
6.3 Effectiveness of Topological Features

In this experiment, we evaluate the effectiveness of each type of features we propose, including degree, betweenness centrality (BC), triad counts, edge embeddedness (Embed) and latent features constructed by NMTF. We set $k = 30$ in matrix Σ_k for the latent feature space. We use source graph edges and a certain percentage of labeled target graph edges for training; then use our instance weighting algorithm for



(a) Slashdot vs. Epinions (b) Slashdot vs. Wikipedia (c) Epinions vs. Wikipedia

Figure 3: Prediction Accuracy by Iteration of Algorithm 2



(a) Slashdot vs. Epinions (b) Slashdot vs. Wikipedia (c) Epinions vs. Wikipedia

Figure 4: Feature Effectiveness Comparison with 2% Target Training Instances

model learning. Only one type of feature is used in each learned model. We also use all these features (All) to learn a model to show their overall performance.

We first use 2% labeled edge instances in the target graph plus all source edge instances for training. Figures 4(a)–4(c) show the prediction accuracy of each feature type on the three network pairs respectively. Among the five types of features, we can see that our latent features always achieve the highest accuracy, followed by Degree, Triad, BC and Embed. This is because the latent features can capture the common structural patterns between the source and target graphs, despite the different distributions between them. Thus the latent features can generalize well from the source graph to the target. In contrast, the other four types of features are not generalizable from the source graph to the target. As the statistics in Table 1 show that the three social networks have different distributions in degree and other dimensions, it is not difficult to understand that a model learned from the source edge instances based on these explicit features cannot generalize well to predict the sign of test edges in the target graph. Finally, the model using all these features achieves the best performance.

When we increase the percentage of labeled target edge instances to 30% for training, the prediction accuracy is shown in Figures 5(a)–5(c). Among the five types of features, Latent still achieves the highest accuracy in most cases, followed by Degree, Triad, BC and Embed. But the performance difference between Latent and Degree/Triad is not as significant. This is because when the amount of labeled target edge instances is much larger, the labeled target instances provide reliable feature values on Degree and Triad. The source graph edges which have a different distribution will have a smaller influence in model learning through instance weighting. Finally, the model using all these features still achieves the best performance.

Tri-Factorization Sensitivity Test: We perform sensitivity test on the dimension of the latent feature space, i.e., k in the $k \times k$ matrix Σ_k computed by matrix tri-factorization. We vary the k value of the latent feature space and report the prediction accuracy in Figures 6(a)–6(c) on the three network pairs respectively. 30% labeled target edge instances plus all source graph edges are used for training. Here legend *Src+Target* means using both source graph edges and labeled target graph edges without instance weighting, and *IW* means our instance weighting method. We can observe that the prediction accuracy increases first when k increases and then becomes stable or even slightly decreases when $k > 30$ for all three groups of experiments. When $k > 30$, we find many entries in the latent feature vector become almost 0, thus contribute little to prediction, or even degrade the performance.

Tri-Factorization Convergence Analysis: We prove the convergence of Algorithm 1 in Section 4.2.2. We report the objective values (in log scale, i.e., $\ln \mathcal{J}$) over 20 iterations under three latent feature dimensions $k = 10, 30, 60$ in Figure 7. Due to the lack of space, we only show the results when Slashdot is the source and Epinions is the target. We can observe that the objective values which measure the difference between the original matrix and the decomposed ones converge very quickly (after 4 iterations) regardless of the feature dimension k. This result confirms our theoretical analysis on the convergence of Algorithm 1.

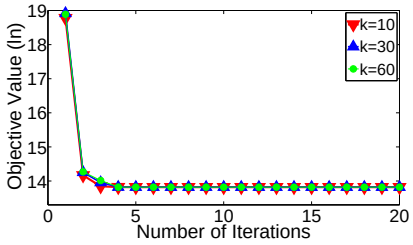
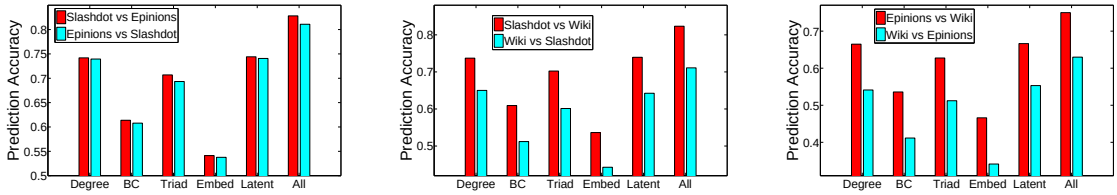


Figure 7: NMTF Convergence

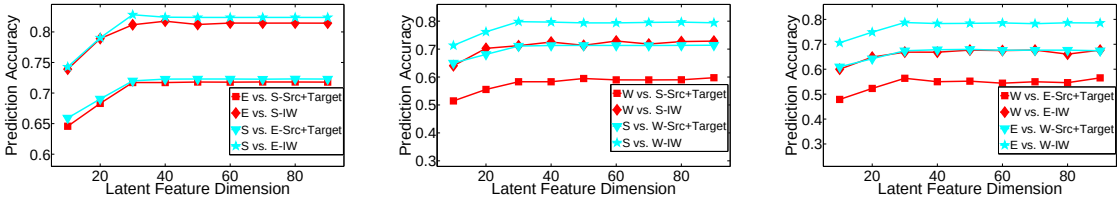
7. CONCLUSIONS

We studied the edge sign prediction problem in signed social networks, which have both positive and negative links. We assume the edge sign information is very scarce in the target network which is very common for newly formed networks. This problem is important because knowing the edge



(a) Slashdot vs. Epinions (b) Slashdot vs. Wikipedia (c) Epinions vs. Wikipedia

Figure 5: Feature Effectiveness Comparison with 30% Target Training Instances



(a) Slashdot vs. Epinions (b) Slashdot vs. Wikipedia (c) Epinions vs. Wikipedia

Figure 6: Sensitivity Test of Latent Feature Dimension

signs will provide us with better understanding of user opinions in a social network. It is challenging due to the inadequate edge signs in the target network and the prohibitive cost of manual labeling. We adopt the transfer learning approach by leveraging a source network with abundant edge sign information but possibly under a different yet related distribution. We propose to construct generalizable latent features through NMTF by considering both the source and target networks, and then adopt an AdaBoost-like algorithm with instance weighting to train a good classifier. Extensive experiments on three real signed networks Epinions, Slashdot and Wikipedia prove the effectiveness of our extracted features and transfer learning algorithm.

8. ACKNOWLEDGMENTS

The work described in this paper was partially supported by two China National 973 projects (grant No. 2012CB315904 and 2013CB336700), several grants from the University Grants Committee of the Hong Kong Special Administrative Region, China (Area of Excellence Project No. AoE/E-02/08 and General Research Fund Project 411211, 411310, 411010 and 411011), and two gift grants from Microsoft and Cisco.

9. REFERENCES

[1] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *NIPS*, pages 41–48, 2006.

[2] M. Burke and R. Kraut. Mopping up: Modeling wikipedia promotion decisions. In *CSCW*, pages 27–36, 2008.

[3] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu. Boosting for transfer learning. In *ICML*, pages 193–200, 2007.

[4] C. Ding, T. Li, W. Peng, and H. Park. Orthogonal nonnegative matrix tri-factorizations for clustering. In *KDD*, pages 126–135, 2006.

[5] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, pages 23–37, 1995.

[6] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *WWW*, pages 403–412, 2004.

[7] J. He and R. Lawrence. A graph-based framework for multi-task multi-view learning. In *ICML*, pages 25–32, 2011.

[8] J. He, Y. Liu, and R. Lawrence. Graph-based transfer learning. In *CIKM*, pages 937–946, 2009.

[9] T. Ito, M. Shimbo, T. Kudo, and Y. Matsumoto. Application of kernels to link analysis. In *KDD*, pages 586–592, 2005.

[10] E. Katz and P. F. Lazarsfeld. *Personal influence: The part played by people in the flow of mass communications*. Free Press, 1955.

[11] J. Kunegis, A. Lommatzsch, and C. Bauckhage. The slashdot zoo: Mining a social network with negative edges. In *WWW*, pages 741–750, 2009.

[12] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. Luca, and S. Albayrak. Spectral analysis of signed graphs for clustering, prediction and visualization. In *SDM*, pages 559–570, 2010.

[13] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000.

[14] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, pages 641–650, 2010.

[15] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[16] J. J. Lim, R. Salakhutdinov, and A. Torralba. Transfer learning by borrowing examples for multiclass object detection. In *NIPS*, pages 118–126, 2011.

[17] M. Long, J. Wang, G. Ding, W. Cheng, X. Zhang, and W. Wang. Dual transfer learning. In *SDM*, pages 540–551, 2012.

[18] P. Massa and P. Avesani. Controversial users demand local trust metrics: An experimental study on epinions.com community. In *AAAI*, pages 121–126, 2005.

[19] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[20] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135, 2008.

[21] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *ICML*, pages 759–766, 2007.

[22] T. Yang, R. Jin, A. K. Jain, Y. Zhou, and W. Tong. Unsupervised transfer classification: Application to text categorization. In *KDD*, pages 1159–1168, 2010.

[23] F. Zhuang, P. Luo, H. Xiong, Q. He, Y. Xiong, and Z. Shi. Exploiting associations between word clusters and document classes for cross-domain text categorization. *Statistical Analysis and Data Mining*, 4(1):100–114, 2011.