Supervised Network Alignment

4

4.1 Overview

Online social networks, such as Facebook,[1] Twitter,[2] Foursquare,[3] and LinkedIn,[4] have become more and more popular in recent years. Each social network can be represented as a heterogeneous network containing abundant information about: who, where, when, and what, i.e., who the users are, where they have been to, what they have done, and when they did these activities. Different online social networks can provide unique social network services for the users. For instance, Facebook is a general public social sharing site, Twitter is a micro blogging social site mainly about short posts, Foursquare is a location based social network, and LinkedIn is a business oriented professional social network site.

Nowadays, to enjoy the social network services from multiple sites at the same time, people are usually getting involved in more and more different kinds of social networks simultaneously. For example, people usually share reviews or tips about different locations or places with their friends using Foursquare. At the same time, they may also share the latest news using Twitter, and share photos using Facebook. Thus, each user often has multiple separate accounts in different social networks. However, these accounts of the same user are mostly isolated without any connections or correspondence relationships to each other.

Discovering the correspondence between accounts of the same user is a crucial prerequisite for many interesting inter-network applications, such as *friend recommendation* [25, 26, 31, 35, 36], *social community detection* [27, 31, 32], and *social information diffusion* [10, 31, 39, 40, 42] using information from multiple networks simultaneously. For example, in the Foursquare network, the social connections and activities of new users can be very sparse. The friend and location recommendations for users are very hard merely using the Foursquare network. However, if the user's Twitter account is also known, his/her social connections and location check-in data in Twitter network will be used to improve the recommendation services in the Foursquare network.

In this book, we focus on studying the common users shared by different social networks, and the correspondence relationship between the shared users across social networks are called the *anchor links* [12, 31, 37]. Formally, inferring the common users shared by different social network sites is

[1]https://www.facebook.com.

[2]https://twitter.com.

[3]https://foursquare.com.

[4]https://www.linkedin.com.

J. Zhang, P. S. Yu, *Broad Learning Through Fusions*,
https://doi.org/10.1007/978-3-030-12528-8_4

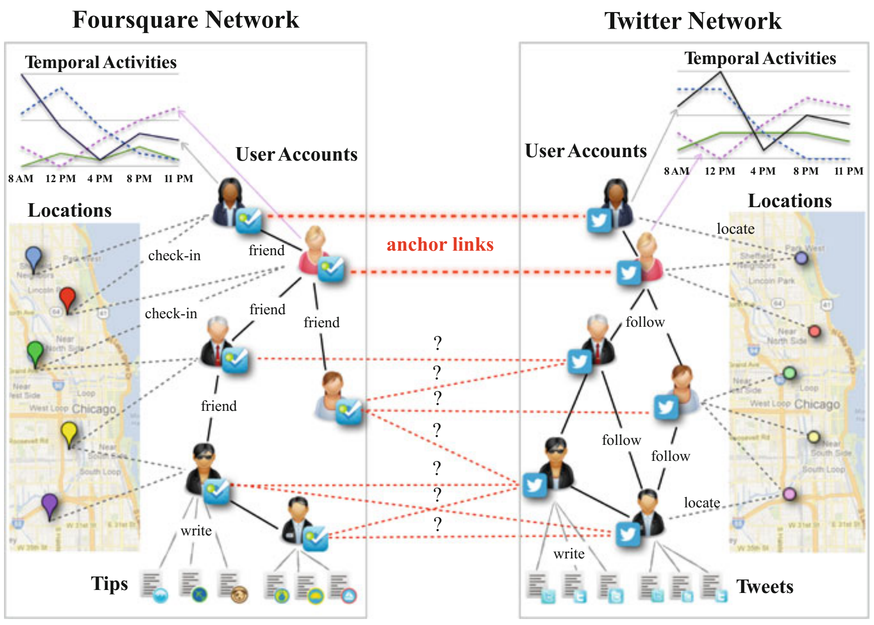


Fig. 4.1 An example of supervised network alignment problem

called the *network alignment* problem [12, 31, 33, 34, 38, 41], which can also be called the *anchor link prediction* problem [12, 31]. In this book, these two terms are used interchangeably when referring to the problem. In the real-world, the anchor links connecting the common information entities across different social networks are extremely hard to identify. Manual labeling of the anchor links between networks can be a tedious and complicated task. Depending on whether the pre-labeled training data is available or not, in this chapter as well as the following Chaps. 5–6, the network alignment problem will be introduced based on different learning settings, including *supervised learning setting*, *unsupervised learning setting*, and *semi-supervised learning setting*.

In this chapter, as illustrated in Fig. 4.1, the network alignment problem is studied based on the *supervised learning setting* specifically, assuming that we can obtain a set of labeled anchor links as the training set in advance. We will start this section with the *supervised network alignment* problem definition. Subject to the *one-to-one* constraint [12,33,34], two different network alignment models for *full network alignment* [12,31,33,34] and *partial network alignment* [31,38] models will be introduced afterwards, respectively, which both use the classification method as the base model. To incorporate the *one-to-one* cardinality constraint into the problem formulation, the *anchor link prediction with cardinality constraint* framework will be introduced, which is a generalized link prediction model and can be used in inferring other types of links subject to any cardinality constraints as well.

4.2 Supervised Network Alignment Problem Definition

Example 4.1 In Fig. 4.3, we show an example of two heterogeneous social networks (Twitter and Foursquare) with six users. Each user has two accounts in these two networks separately. In each network, users are connected with each other through the social links. Moreover, each user is also connected with a set of locations, timestamps, and text contents created by their online social activities. Note that the top two users in Fig. 4.3 also have another type of link, which connects the same user's accounts between two networks. We call these links the *anchor links* [12, 37] as introduced in Sect. 3.4.3. Each anchor link indicates a pair of accounts that belong to the same user. The task of network alignment problem is to discover which account pairs, as shown with question marks in Fig. 4.3, should be connected by the anchor links in the real world.

Both the Foursquare and Twitter networks shown in the example can be represented as heterogeneous social networks defined in the previous chapter. For instance, we can represent the Foursquare network as $G^{(1)} = (\mathcal{V}^{(1)}, \mathcal{E}^{(1)})$, where $\mathcal{V}^{(1)}$ and $\mathcal{E}^{(1)}$ denote the node and link sets, respectively. As introduced before, the Foursquare involves different types of information entities, and the node set $\mathcal{V}^{(1)} = \mathcal{U}^{(1)} \cup \mathcal{P}^{(1)} \cup \mathcal{L}^{(1)} \cup \mathcal{W}^{(1)} \cup \mathcal{T}^{(1)}$ contains the user, post, location check-in, word, and timestamp nodes, respectively. The node subset $\mathcal{U}^{(1)} = \{u_1^{(1)}, u_2^{(1)}, \ldots, u_{n^{(1)}}^{(1)}\}$ denotes the set of users in Foursquare. The posts written by the users are represented as the set $\mathcal{P}^{(1)}$, which may involve a set of location check-ins $\mathcal{L}^{(1)}$, text words $\mathcal{W}^{(1)}$, and timestamps $\mathcal{T}^{(1)}$. Meanwhile, the link set $\mathcal{E}^{(1)} = \mathcal{E}_{u,u}^{(1)} \cup \mathcal{E}_{u,p}^{(1)} \cup \mathcal{E}_{p,l}^{(1)} \cup \mathcal{E}_{p,w}^{(1)} \cup \mathcal{E}_{p,t}^{(1)}$ involves the links among users, between users and posts, as well as those between posts and locations check-ins, words, and timestamps. Similarly, the Twitter network can be represented as a heterogeneous social network $G^{(2)}$ with a similar structure, the user node set involved in which can be denoted as $\mathcal{U}^{(2)} = \{u_1^{(1)}, u_2^{(1)}, \ldots, u_{n^{(2)}}^{(1)}\}$.

In the network alignment problem settings, the anchor links are assumed to be subject to the *one-to-one* constraint, where a user from one network can be connected by at most one anchor link with users from another network. In the supervised network alignment setting, a set of existing and non-existing anchor links can be pre-identified and labeled as the (positive and negative) training set $\mathcal{A}_{train} \subset \mathcal{U}^{(1)} \times \mathcal{U}^{(2)}$. And the supervised network alignment problem aims at inferring the existence of the remaining potential anchor links, i.e., $\mathcal{A}_{test} = \mathcal{U}^{(1)} \times \mathcal{U}^{(2)} \setminus \mathcal{A}_{train}$, among users between networks $G^{(1)}$ and $G^{(2)}$.

In the supervised network alignment problem, a set of features will be extracted for the anchor links across networks with the heterogeneous information available in the networks. Meanwhile, the existing and non-existing anchor links will be labeled as positive and negative instances, respectively. Based on the training set $\mathcal{A}_{train}$, the feature vectors and labels of links in the set can be represented as tuples $\{(\mathbf{x}_l, y_l)\}_{l \in \mathcal{A}_{train}}$, where $\mathbf{x}_l$ represents the feature vector extracted for anchor link l and $y_l \in \{-1, +1\}$ denotes its label. Based on the training set, the *supervised network alignment* problem aims at building a mapping $f : \mathcal{A}_{test} \rightarrow \{-1, +1\}$ to determine the labels of the anchor links in the testing set.

4.3 Supervised Full Network Alignment

Depending on how many users in the networks are the shared anchor users, the supervised network alignment can be categorized into the *supervised full network alignment* [12] and *supervised (mutually) partial network alignment* [38] problems. In the *full network alignment* problem, the networks to be studied are fully aligned and each user will be connected by an anchor link in the results; while in the *partial network alignment* problem, the networks are partially aligned and many users should stay isolated without any matching partners across networks. In this section, we will introduce the model proposed to solve the *supervised full network alignment*, where the users in networks $G^{(1)}$ and $G^{(2)}$ are all anchor users and will get connected by the anchor links.

As introduced before, in this section, we will introduce a two-phase approach, namely MNA [12], to address the *full network alignment* problem. The first phase of MNA tackles the feature extraction problem, while the second phase takes care of the one-to-one constrained anchor link prediction.

The phase of feature extraction mainly explores two kinds of ideas on multiple heterogeneous social networks. First, we exploit social links in each network and the labeled anchor links across the two networks to extract social features for anchor link prediction. Second, we exploit the heterogeneous information in both networks to extract three sets of heterogeneous features for anchor link prediction, which correspond to aggregated patterns of users on spatial distribution, temporal activity distribution, and textual usage behavior separately. All these extracted features and the pairs of accounts with known labels will be used to learn a binary SVM for anchor link prediction. Since the label predictions of SVM don't satisfy the one-to-one constraint, the real-value scores of the SVM are used as the input for the second phase, and derive the anchor link predictions collectively according to the one-to-one constraint.

4.3.1 Feature Extraction for Anchor Links

There exist different types of information in the networks, including the social connections, location check-ins, text words, and timestamps, which can all indicate the correspondence relationships among users across networks.

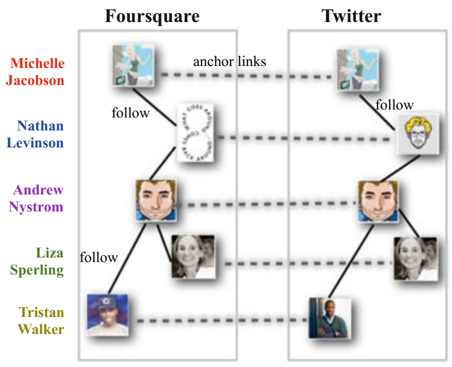
Example 4.2 We show a case study to demonstrate these heterogeneous information from two networks are useful for identifying the anchor links. In Fig. 4.2, we show a case of five real-world users who have both Twitter and Foursquare accounts. These five users are socially connected in both networks, as shown in Fig. 4.2a. By considering this social information, we can significantly shrink the search space for anchor links if one or some of these users' accounts in both networks have already been labeled by anchor links. In Fig. 4.2b, we show the spatial distributions of different users in both networks. We can see that the spatial distribution of the same user is pretty similar to each other. Michelle is mainly located in the central states of the USA, when sending tweets and foursquare tips. The spatial distributions of her foursquare account and twitter account are pretty similar. In Fig. 4.2c, we show the temporal distributions of the users. We can see that Tristan's temporal activities across both Twitter account and Foursquare account are very consistent, and his distribution is very different from Lisa's temporal activity pattern. In Fig. 4.2d, we show some frequently used words by the users, where the choices of words of the same user can be pretty consistent. For example, Andrew seems to prefer to use "awsm" instead of "awesome" when writing tweets and tips.

Most existing features for link prediction, such as numbers of common neighbors and other social closeness measures introduced in the previous chapter, mainly focus on one single network setting, and the target links are assumed to be subject to the many-to-many cardinality constraint. These features cannot be directly used for the anchor link prediction task across multiple networks. Based on the above example, in the following part, we will introduce the features that can be extracted for the anchor links between networks with such heterogeneous social information.

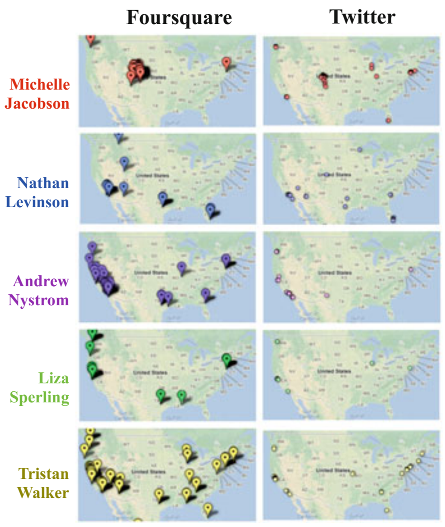
4.3.1.1 Social Connection Based Features

Users often have similar social links in different social networks, such as Twitter and Facebook, because such social links usually indicate the user's social ties in real life. In other words, the social similarity between two user accounts from different social networks can be exploited to help locate the same user.

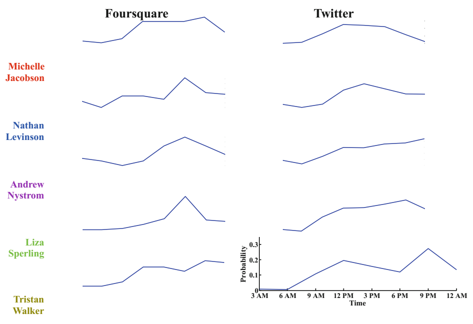
Our goal is to extract discriminative social features for a pair of user accounts in two disjoint social networks. Intuitively, the social neighbors of each user account can only involve user accounts from the same social network. For example, the neighbors for a Facebook account can only involve



(a)



(b)



(c)

user	word counts in both networks (Twitter, Foursquare)
Michelle Jacobson	art (65,2), style (16,3) audit (3,2), grill (19,2)
Nathan Levinson	happy (27,5), enjoy (9,4) week (18,4), shows (6,6)
Andrew Nystrom	awsm (2,3), kids (20,3) red (61,3), open (11,4)
Liza Sperling	ask (6,5), coffee (8,3) mochi (1,3), hangout (5,2)
Tristan Walker	win (19,4), amazing (55,5) awesome (51,4), please (9,4)

(d)

Fig. 4.2 Case study: five real-world users with their social, spatial, temporal, and text distributions. (**a**) Social. (**b**) Spatial. (**c**) Temporal. (**d**) Text

Facebook accounts instead of Twitter accounts. However, in anchor link prediction problem, we need to extract a set of features about a pair of user accounts in two different networks separately. The social neighbors for two user accounts are two disjoint sets of user accounts in two separate networks. There cannot exist any shared nodes among the neighbors of the pair of user accounts. In the following part, we will introduce the extension of several social features to multi-network settings.

Let $(u_i^{(1)}, u_j^{(2)})$ be a potential anchor link between these two networks, and $\mathcal{A}_{train}^{+}$ be the set of existing anchor links in the training set. Here we extend the definitions of some commonly used social closeness features in link prediction, i.e., "common neighbors," "Jaccard's coefficient," and "Adamic/Adar measure," to the *inter-network* scenarios for anchor links specifically.

- **Extended Common Neighbor**: The *extended common neighbor* measure $ECN(u_i^{(1)}, u_j^{(2)})$ represents the number of "common" neighbors between $u_i^{(1)}$ in network $G^{(1)}$ and $u_j^{(2)}$ in network $G^{(2)}$. We can denote the neighbors of $u_i^{(1)}$ in network $G^{(1)}$ as $\Gamma(u_i^{(1)})$, and the neighbors of $u_j^{(2)}$ in network $G^{(2)}$ as $\Gamma(u_j^{(2)})$. It is easy to identify that the sets $\Gamma(u_i^{(1)})$ and $\Gamma(u_j^{(2)})$ contain the users from two different networks, respectively, which are isolated without any common entries, i.e., $\Gamma(u_i^{(1)}) \cap \Gamma(u_j^{(2)}) = \emptyset$.

Meanwhile, based on the existing anchor links $\mathcal{A}^+_{train}$, some of the users in $\Gamma(u_i^{(1)})$ and $\Gamma(u_j^{(2)})$ can correspond to the accounts of the same users in these two networks, who are actually connected by the anchor links in $\mathcal{A}^+_{train}$. Therefore, based on the anchor links in set $\mathcal{A}^+_{train}$, the *extended common neighbor* measure between these two users can be defined as the number of shared anchor users in their neighbor sets, respectively.

Definition 4.1 (Extended Common Neighbor) The measure of *extended common neighbor* is defined as the number of shared users between $\Gamma(u_i^{(1)})$ and $\Gamma(u_j^{(2)})$.

$$
\begin{aligned}
&ECN(u_i^{(1)}, u_j^{(2)}) \\
&= \left|\{(u_p^{(1)}, u_q^{(2)}) | (u_p^{(1)}, u_q^{(2)}) \in \mathcal{A}^+_{train}, u_p^{(1)} \in \Gamma(u_i^{(1)}), u_q^{(2)} \in \Gamma(u_j^{(2)})\}\right| \\
&= \left| \Gamma(u_i^{(1)}) \bigcap_{\mathcal{A}^+_{train}} \Gamma(u_j^{(2)}) \right|. \qquad (4.1)
\end{aligned}
$$

Example 4.3 For instance, given an input network in Fig. 4.3 involving six users, there exist two known anchor links and the existing anchor link set $\mathcal{A}^+_{train} = \{(Alice_F, Alice_T), (Cindy_F, Cindy_T)\}$, where the subscript denotes the network identifier (F: Foursquare, T: Twitter). Based

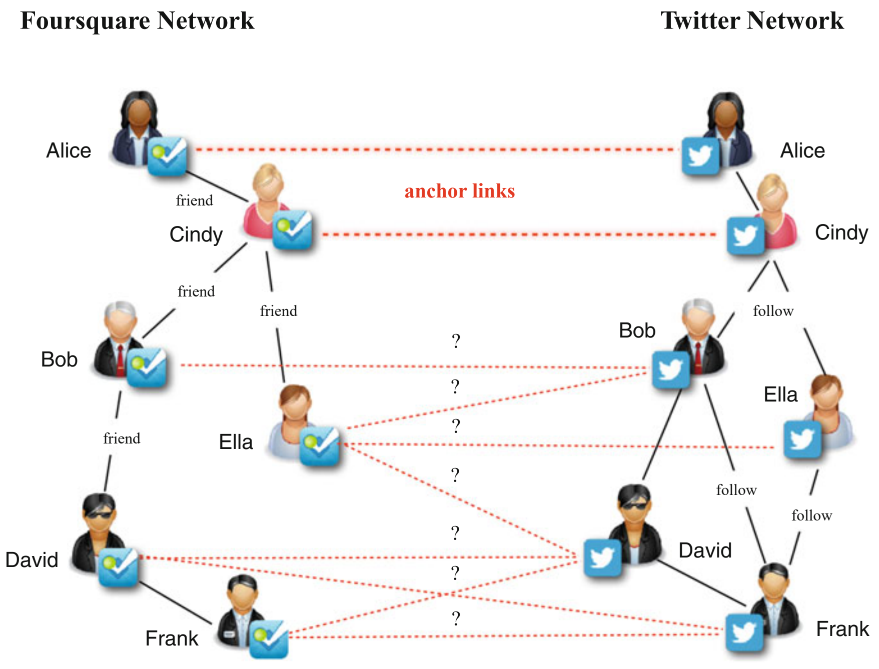


Fig. 4.3 An example of aligned social networks for feature extraction

on the input networks, the *extended common neighbor* feature between two different user pairs (Bob_F, Bob_T) and $(David_F, David_T)$ can be computed as follows:

(1) (Bob_F, Bob_T): We have the neighborhood sets of users Bob_F and Bob_T in the Foursquare and Twitter networks to be sets $\Gamma(Bob_F) = \{Cindy_F, David_F\}$ and $\Gamma(Bob_T) = \{Cindy_T, David_T, Frank_T\}$. We have

$$\begin{aligned} ECN(Bob_F, Bob_T) &= \left| \Gamma(Bob_F) \bigcap_{\mathcal{A}^+_{train}} \Gamma(Bob_T) \right| \\ &= |\{(Cindy_F, Cindy_T)\}| \\ &= 1. \end{aligned} \tag{4.2}$$

(2) $(David_F, David_T)$: We can represent the neighborhood sets of $David_F$ and $David_T$ in Foursquare and Twitter to be $\Gamma(David_F) = \{Bob_F, Frank_F\}$ and $\Gamma(David_T) = \{Bob_T, Frank_T\}$. According to the existing anchor link set, no shared users exist in the neighbor sets of $David_F$ and $David_T$. In other words, we have

$$ECN(David_F, David_T) = 0. \tag{4.3}$$

- **Extended Jaccard's Coefficient**: The measure of Jaccard's coefficient can also be extended to the multi-network setting for anchor links using a similar method of extending common neighbor. The *extended Jaccard's Coefficient* measure $EJC(u_i^{(1)}, u_j^{(2)})$ is a normalized version of the *extended common neighbor*, i.e., $ECN(u_i^{(1)}, u_j^{(2)})$ divided by the total number of distinct users in $\Gamma(u_i^{(1)})$ and $\Gamma(u_j^{(2)})$.

Definition 4.2 (Extended Jaccard's Coefficient) Given the neighborhood sets of users $u_i^{(1)}$ and $u_j^{(2)}$ in networks $G^{(1)}$ and $G^{(2)}$, respectively, the *Extended Jaccard's Coefficient* of user pair $u_i^{(1)}$ and $u_j^{(2)}$ can be represented as

$$EJC(u_i^{(1)}, u_j^{(2)}) = \frac{\left| \Gamma(u_i^{(1)}) \bigcap_{\mathcal{A}^+_{train}} \Gamma(u_j^{(2)}) \right|}{\left| \Gamma(u_i^{(1)}) \bigcup_{\mathcal{A}^+_{train}} \Gamma(u_j^{(2)}) \right|}, \tag{4.4}$$

where

$$\left| \Gamma(u_i^{(1)}) \bigcup_{\mathcal{A}^+_{train}} \Gamma(u_j^{(2)}) \right| = |\Gamma(u_i^{(1)})| + |\Gamma(u_j^{(2)})| - \left| \Gamma(u_i^{(1)}) \bigcap_{\mathcal{A}^+_{train}} \Gamma(u_j^{(2)}) \right|. \tag{4.5}$$

Example 4.4 For example, based on the aligned social networks in Fig. 4.3, we can compute the *extended Jaccard's Coefficient* of user pair (Bob_F, Bob_T) to be

$$\begin{aligned} EJC(Bob_F, Bob_T) &= \frac{\left| \Gamma(Bob_F) \bigcap_{\mathcal{A}^+_{train}} \Gamma(Bob_T) \right|}{\left| \Gamma(Bob_F) \bigcup_{\mathcal{A}^+_{train}} \Gamma(Bob_T) \right|} \\ &= \frac{1}{4}. \end{aligned} \tag{4.6}$$

- **Extended Adamic/Adar Index**: In addition, the Adamic/Adar measure can also be extended to the multi-network setting, where the common neighbors are weighted by their average degrees in both social networks.

Definition 4.3 (Extended Adamic/Adar Index) The *Extended Adamic/Adar Index* of the user pairs $u_i^{(1)}$ and $u_j^{(2)}$ across networks can be represented as

$$EAA(u_i^{(1)}, u_j^{(2)}) = \sum_{\forall (u_p^{(1)}, u_q^{(2)}) \in \Gamma(u_i^{(1)}) \bigcap_{\mathcal{A}^+_{train}} \Gamma(u_j^{(2)})} \log^{-1}\left(\frac{|\Gamma(u_p^{(1)})| + |\Gamma(u_q^{(2)})|}{2}\right). \tag{4.7}$$

In the EAA definition, for the common neighbors shared by $u_i^{(1)}$ and $u_j^{(2)}$, their degrees are defined as the average of their degrees in networks $G^{(1)}$ and $G^{(2)}$, respectively. Considering that different networks are of different scales, like Twitter is far larger than Foursquare, the node degree measure can be dominated by the degree of the larger networks. Some other weighted forms of the degree measure, like $\alpha \cdot |\Gamma(u_p^{(1)})| + (1-\alpha) \cdot |\Gamma(u_q^{(2)})$ ($\alpha \in [0, 1]$), can be applied to replace $\frac{|\Gamma(u_p^{(1)})| + |\Gamma(u_q^{(2)})|}{2}$ in the definition.

In addition to the social features mentioned above, heterogeneous social networks also involve abundant information about: where, when, and what. In the following part, we will introduce how to exploit the spatial, temporal, and text content information of different user accounts to facilitate anchor link prediction.

4.3.1.2 Spatial Check-In Distribution Features

Besides the social connection information, users' activities in the offline world may also provide important signals for inferring the anchor links across the networks as indicated in Fig. 4.2b. We notice that users in different social networks usually check-in at similar locations in real life, such as their home, working places, traveling spots, etc. The similarity between the spatial distributions of two user accounts from different social networks can also be used to help locate the same user.

Each location can be represented as a pair of (latitude, longitude) = $\ell \in \mathcal{L}^{(1)}$ (or $\ell \in \mathcal{L}^{(2)}$). Three different measures have been introduced in [12] to evaluate the similarity between the spatial distributions of two users accounts. Given a user $u_i^{(1)}$ in network $G^{(1)}$, we can represent the locations she/he has ever visited as set $L(u_i^{(1)}) \subset \mathcal{L}^{(1)}$. In a similar way, we can also represent the set of location visited by user $u_j^{(2)}$ as set $L(u_j^{(2)}) \subset \mathcal{L}^{(2)}$.

- **Number of Shared Locations**: Formally, we introduce a notation $l_1 = l_2$ to denote that these two locations are the same location, i.e., sharing common latitude and longitude. The number of shared locations which have been visited by users $u_i^{(1)}$ and $u_j^{(2)}$ can be denoted as

$$\left|\{(l^{(1)}, l^{(2)}) | l^{(1)} \in L(u_i^{(1)}), l^{(2)} \in L(u_j^{(2)}), l^{(1)} = l^{(2)}\}\right|. \tag{4.8}$$

- **Cosine Similarity Between Location Vectors**: Let $\mathcal{L} = \mathcal{L}^{(1)} \cup \mathcal{L}^{(2)}$ be the set of all locations in both network $G^{(1)}$ and network $G^{(2)}$, for user $u_i^{(1)}$ (or $u_j^{(2)}$) in these two networks, the locations visited by them can be organized as a binary vector of length $|\mathcal{L}|$, i.e., $\mathbf{l}(u_i^{(1)}) \in \mathbb{R}^{|\mathcal{L}|}$ (or $\mathbf{l}(u_j^{(2)}) \in \mathbb{R}^{|\mathcal{L}|}$), the entries in which denote the visiting times for users at these locations. The similarity of the location visiting records between the users $u_i^{(1)}$ and $u_j^{(2)}$ can be denoted as the cosine similarity of the location-visiting record vectors $\mathbf{l}(u_i^{(1)})$ and $\mathbf{l}(u_j^{(2)})$, which can be used as another feature based on the location check-in records

$$\frac{\mathbf{l}(u_i^{(1)})^\top \mathbf{l}(u_j^{(2)})}{\sqrt{\left(\mathbf{l}(u_i^{(1)})^\top \mathbf{l}(u_i^{(1)})\right)\left(\mathbf{l}(u_j^{(2)})^\top \mathbf{l}(u_j^{(2)})\right)}}. \tag{4.9}$$

- **Average Distance of Visited Locations**: The physical distance between the regions that users pairs are active in can indicate their potential similarity from the geographical perspective. The average distance between the locations visited by the users $u_i^{(1)}$ and $u_j^{(2)}$ can be calculated as another feature based on the check-in records

$$D(L(u_i^{(1)}), L(u_j^{(2)})) = \frac{\sum_{l^{(1)} \in L(u_i^{(1)})} \sum_{l^{(2)} \in L(u_j^{(2)})} D(l^{(1)}, l^{(2)})}{|L(u_i^{(1)})||L(u_j^{(2)})|}, \tag{4.10}$$

where term $D(l^{(1)}, l^{(2)})$ represents the distance between locations $l^{(1)}$ and $l^{(2)}$ (Manhattan distance can be used here).

4.3.1.3 Temporal Distribution Features

We also notice that users in different social networks usually publish posts at similar time slots in real life, e.g., hours after work and weekends, etc. Such temporal distribution can effectively indicate the user's online activity patterns. For example, some users may like to send tweets at night, while other users may like to write tweets at commuting time on the bus or train. The temporal distribution of different user accounts can also help us find the anchor links between two networks.

Users' online social activities can be organized into a temporal vector of length 24, where each entry denotes the ratio of activities in each of the time bin. For instance, the temporal activity vector of user $u_i^{(1)}$ can be represented as a vector $\mathbf{t}(u_i^{(1)}) \in \mathbb{R}^{24}$. Similarly, we can also obtain the temporal activity similarity of users from different networks by calculating either the inner product or the cosine similarity of these two temporal activity vectors as follows:

$$\text{(1) inner product: } \mathbf{t}(u_i^{(1)})^\top \mathbf{t}(u_j^{(2)}), \tag{4.11}$$

$$\text{(2) cosine similarity: } \frac{\mathbf{t}(u_i^{(1)})^\top \mathbf{t}(u_j^{(2)})}{\sqrt{\left(\mathbf{t}(u_i^{(1)})^\top \mathbf{t}(u_i^{(1)})\right)\left(\mathbf{t}(u_j^{(2)})^\top \mathbf{t}(u_j^{(2)})\right)}}. \tag{4.12}$$

4.3.1.4 Textual Content Features

According to the textual content analysis provided in the table in Fig. 4.2d, the textual content of posts written by users in different social networks can also be a great hint for the anchor links, because different users may have different choices of words in their posts. The words used by the users can reveal either their personal habits or their word usage patterns.

The post contents written by each user can be converted into a bag-of-words vector weighted by TF-IDF. Let $\mathcal{W}$ denote the set of words used by all the users in networks $G^{(1)}$ and $G^{(2)}$. The word usage record vectors of users $u_i^{(1)}$ and $u_j^{(2)}$ in these two networks can be represented as vectors $\mathbf{w}(u_i^{(1)}) \in \mathbb{R}^{|\mathcal{W}|}$ and $\mathbf{w}(u_j^{(2)}) \in \mathbb{R}^{|\mathcal{W}|}$, respectively. Similar to the temporal activities information, the inner product and cosine similarity can be applied to these two vectors to denote how similar these users are in the textual word usage patterns:

$$\text{(1) inner product: } \mathbf{w}(u_i^{(1)})^\top \mathbf{w}(u_j^{(2)}), \tag{4.13}$$

$$\text{(2) cosine similarity: } \frac{\mathbf{w}(u_i^{(1)})^\top \mathbf{w}(u_j^{(2)})}{\sqrt{\left(\mathbf{w}(u_i^{(1)})^\top \mathbf{w}(u_i^{(1)})\right)\left(\mathbf{w}(u_j^{(2)})^\top \mathbf{w}(u_j^{(2)})\right)}}. \tag{4.14}$$

4.3.2 Supervised Anchor Link Prediction Model

With the features introduced in the previous subsection, in this part, we will introduce the supervised anchor link prediction model. We will provide the general description of the model architecture, and then use an example to show how to build the model.

Given the multiple aligned social networks, via manual labeling, we can identify a set of existing anchor links, denoted as set $\mathcal{A}^+_{train}$, and a set of non-existing anchor links, denoted as set $\mathcal{A}^-_{train}$. The anchor links in sets $\mathcal{A}^+_{train}$ and $\mathcal{A}^-_{train}$ are assigned with the positive and negative labels, respectively, i.e., $\{-1, +1\}$, depending on whether they exist or not. For instance, given a link $l \in \mathcal{A}^+_{train}$, it will be associated with a positive label, i.e., $y_l = +1$, while if link $l \in \mathcal{A}^-_{train}$, it will be associated with a negative label, i.e., $y_l = -1$. With the information across these two aligned heterogeneous social networks, a set of features introduced in the previous subsection can be extracted for the links in sets $\mathcal{A}^+_{train}$ and $\mathcal{A}^-_{train}$. For instance, for a link l in the training set $\mathcal{A}^+_{train}$ (or $\mathcal{A}^-_{train}$), we can represent its feature vector as $\mathbf{x}_l$, which will be called an *anchor link instance* and each feature is an *attribute* of the anchor link (more information about *instance*, *attribute*, and *label* concepts is available in Chap. 2.2). With these anchor link instances and their labels, a supervised learning model can be trained, like the classification models SVM (support vector machine), Decision Tree, or neural networks.

In the test procedure, for each link l in the test set $\mathcal{A}_{test}$, we can represent it with a similar set of features (or attributes) and denote its feature vector as $\mathbf{x}_l$. However, we don't know its label, and we may want to determine whether it exists or not (its label is positive or negative). By applying the trained model to the feature vector of the anchor link, we will obtain a prediction label, which will be returned as the final result.

Example 4.5 In Fig. 4.4, we show an example of the supervised anchor link prediction model architecture. Given the input aligned networks, we can identify a set of existing and non-existing anchor links between them. These links can be used as the training set with the existing anchor links as the positive instances and non-existing ones as the negative instances. Formally, we can denote the training set as $\mathcal{A}_{train}$, where the sets $\mathcal{A}^+_{train} = \{(A_F, A_T), (C_F, C_T)\}$ and $\mathcal{A}^-_{train} = \{(A_F, C_T), (B_F, C_T), \ldots\}$ (the subscripts denotes the networks they belong to, i.e., F: Foursquare; T: Twitter).

These anchor links are labeled as the positive and negative instances, respectively, where the existing anchor links are assigned with the positive label and the non-existing anchor links are assigned with the negative label. Based on the heterogeneous information inside these two networks, a group of features (i.e., the features that we have introduced in the previous part) can be extracted for these links in the training sets. With the training data, a supervised learning model can be built, and further be applied to the feature vectors of some unknown anchor links, e.g., (B_F, B_T), which will output either a label or a score indicating its existence confidence scores.

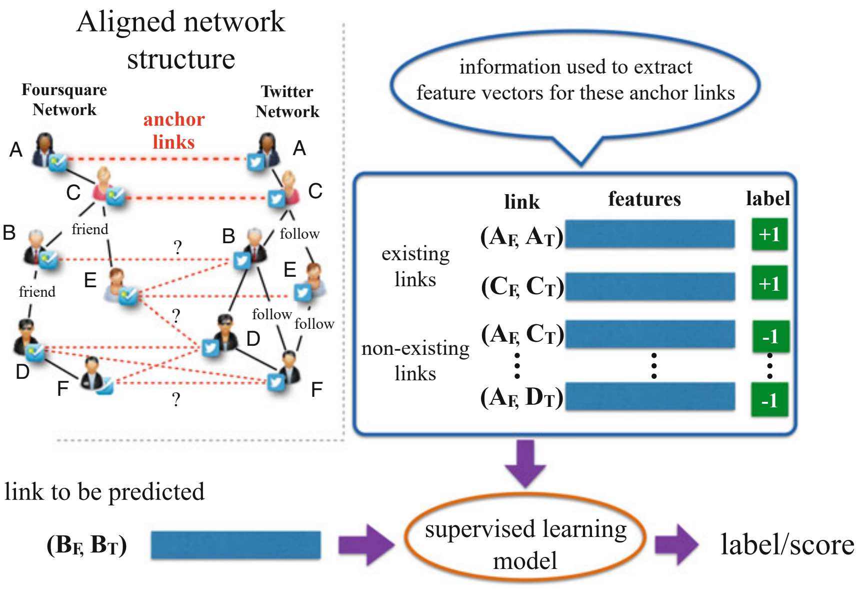


Fig. 4.4 An example of supervised anchor link prediction framework

Meanwhile, due to the *one-to-one* cardinality constraint on the anchor links, among all the potential anchor links incident to each user, only one of them will be positive and the remaining ones will be all negative. In other words, the negative training set is usually much larger than the positive set in the anchor link prediction model. In this part, we will not handle such a challenging problem, which will be taken care of in Sect. 4.4 instead.

4.3.3 Stable Matching

After extracting all the four types of heterogeneous features, we can train a binary classifier, such as SVM or logistic regression, for anchor link prediction. However, in the inference process, the predictions of the binary classifier cannot be directly used as anchor links due to the following issues:

- The inference of conventional classifiers are designed for constraint-free settings, and the one-to-one constraint may not necessarily hold in the label prediction of the classifier (e.g., SVM).
- Most classifiers also produce output scores, which can be used to rank the data points in the test set. However, these ranking scores are uncalibrated in scale to anchor link prediction task. Existing classifier calibration methods [29] can only be applied to classification problems without any constraint.

In order to tackle the above issues, we will introduce an inference process, called MNA (Multi-Network Anchoring) as proposed in [12], to infer anchor links based upon the ranking scores of the classifier. This model is motivated by the *stable marriage problem* [5] in mathematics.

Example 4.6 We first use a toy example in Fig. 4.5 to illustrate the main idea of MNA. Suppose in Fig. 4.5a, we are given the ranking scores from the classifiers, between the four user pairs across

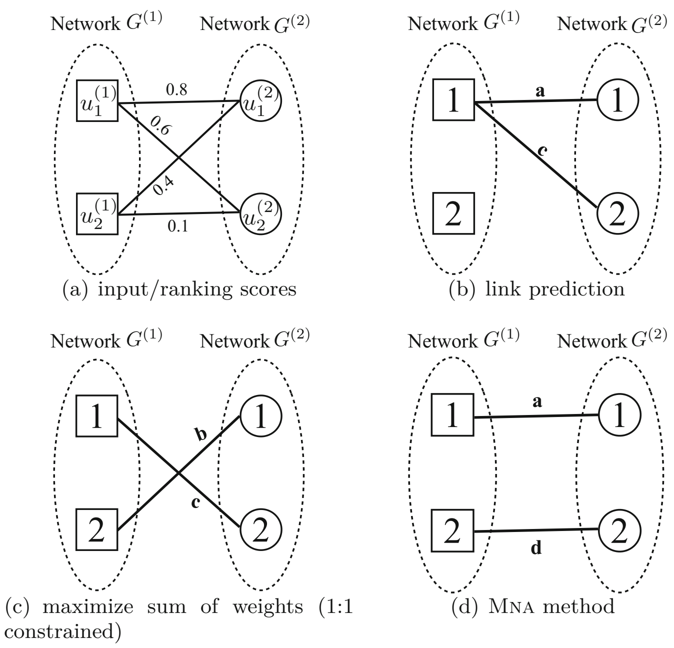


Fig. 4.5 An example of anchor link inference by different methods. (**a**) is the input, ranking scores. (**b**)–(**d**) are the results of different methods for anchor link inference

two networks (i.e., network $G^{(1)}$ and network $G^{(2)}$). We can see in Fig. 4.5b that link prediction methods with a fixed threshold may not be able to predict well, because the predicted links do not satisfy the constraint of one-to-one relationship. Thus one user account in network $G^{(1)}$ can be linked with multiple accounts in network $G^{(2)}$. In Fig. 4.5c, *weighted maximum matching* methods can find a set of links with the maximum sum of weights. However, it is worth noting that the input scores are uncalibrated, so the maximum weight matching may not be a good solution to the anchor link prediction problems. The input scores only indicate the ranking of different user pairs, i.e., the preference relationship among different user pairs.

Here we say "node x prefers node y over node z," if the score of pair (x, y) is larger than the score of pair (x, z). For example, in Fig. 4.5c, the weight of pair a, i.e., Score(a) = 0.8, is larger than Score(c) = 0.6. It shows that user $u_1^{(1)}$ (the first user in network $G^{(1)}$) *prefers* $u_1^{(2)}$ over $u_2^{(2)}$. The problem with the prediction result in Fig. 4.5c is that the pair $(u_1^{(1)}, u_1^{(2)})$ should be more likely to be an anchor link due to the following reasons: (1) $u_1^{(1)}$ prefers $u_1^{(2)}$ over $u_2^{(2)}$; (2) $u_1^{(2)}$ also prefers $u_1^{(1)}$ over $u_2^{(1)}$.

By following such an intuition, we can obtain the final stable matching result in Fig. 4.5d, where anchor links $(u_1^{(1)}, u_1^{(2)})$ and $(u_2^{(1)}, u_2^{(2)})$ are selected in the matching process.

Definition 4.4 (Matching) Mapping $\mu : \mathcal{U}^{(1)} \cup \mathcal{U}^{(2)} \to \mathcal{U}^{(1)} \cup \mathcal{U}^{(2)}$ is defined to be a *matching* iff (1) $|\mu(u_i^{(1)})| = 1, \forall u_i^{(1)} \in \mathcal{U}^{(1)}$ and $\mu(u_i^{(1)}) \in \mathcal{U}^{(2)}$; (2) $|\mu(u_j^{(2)})| = 1, \forall u_j^{(2)} \in \mathcal{U}^{(2)}$ and $\mu(u_j^{(2)}) \in \mathcal{U}^{(1)}$; and (3) $\mu(u_i^{(1)}) = u_j^{(2)}$ iff $\mu(u_j^{(2)}) = u_i^{(1)}$.

Definition 4.5 (Blocking Pair) A pair $(u_i^{(1)}, u_j^{(2)})$ is a blocking pair iff $u_i^{(1)}$ and $u_j^{(2)}$ both prefer each other over their current assignments, respectively in the predicted set of anchor links $\mathcal{A}'$.

Definition 4.6 (Stable Matching) An inferred anchor link set $\mathcal{A}'$ is stable if there is no blocking pair.

Based on the result from the previous step, MNA formulates the anchor link pruning problem as a stable matching problem between user accounts in network $G^{(1)}$ and accounts in network $G^{(2)}$. Assume that we have two sets of unlabeled user accounts, i.e., $\mathcal{U}^{(1)}$ in network $G^{(1)}$ and $\mathcal{U}^{(2)}$ in network $G^{(2)}$. Each user $u_i^{(1)}$ has a ranking list or preference list $P(u_i^{(1)})$ over all the user accounts in network $G^{(2)}$ ($u_j^{(2)} \in \mathcal{U}^{(2)}$) based upon the input scores of different pairs.

Example 4.7 For example, in Fig. 4.5a, the preference list of node $u_1^{(1)}$ is $P(u_1^{(1)}) = (u_1^{(2)} > u_2^{(2)})$, indicating that node $u_1^{(2)}$ is preferred by $u_1^{(1)}$ over $u_2^{(2)}$. The preference list of node $u_2^{(1)}$ is also $P(u_2^{(1)}) = (u_1^{(2)} > u_2^{(2)})$. Similarly, a preference list for each user account in network $G^{(2)}$ can also be built. In Fig. 4.5a, $P(u_1^{(2)}) = P(u_2^{(2)}) = (u_1^{(1)} > u_2^{(1)})$.

The proposed MNA method for anchor link prediction is shown in Algorithm 1. In each iteration, MNA first randomly selects a free user account $u_i^{(1)}$ from network $G^{(1)}$. Then MNA gets the most preferred user node $u_j^{(2)}$ by $u_i^{(1)}$ in its preference list $P(u_i^{(1)})$, and removes $u_j^{(2)}$ from the preference list, i.e., $P(u_i^{(1)}) = P(u_i^{(1)}) \setminus u_j^{(2)}$. If $u_j^{(2)}$ is also a free account, MNA adds the pair of accounts $(u_i^{(1)}, u_j^{(2)})$ into the current solution set $\mathcal{A}'$. Otherwise, $u_j^{(2)}$ is already occupied with $u_p^{(1)}$ in $\mathcal{A}'$. MNA then examines the preference of $u_j^{(2)}$. If $u_j^{(2)}$ also prefers $u_i^{(1)}$ over $u_p^{(1)}$, it means that the pair $(u_i^{(1)}, u_j^{(2)})$ is a blocking pair. MNA removes the blocking pair by replacing the pair $(u_p^{(1)}, u_j^{(2)})$ in the solution set $\mathcal{A}'$ with the pair $(u_i^{(1)}, u_j^{(2)})$. Otherwise, if $u_j^{(2)}$ prefers $u_p^{(1)}$ over $u_i^{(1)}$, MNA starts the next iteration to reach out the next free node in network $G^{(1)}$. The algorithm stops when all the users

Algorithm 1 Multi-network stable matching

```
Require: two heterogeneous social networks, $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$.
         a set of known anchor links $\mathcal{A}$
Ensure: a set of inferred anchor links $\mathcal{A}'$
 1: Construct a training set of user account pairs with known labels using $\mathcal{A}$.
 2: For each pair $(u_i^{(1)}, u_j^{(2)})$, extract four types of features.
 3: Training classification model $C$ on the training set.
 4: Perform classification using model $C$ on the test set.
 5: For each unlabeled user account, sort the ranking scores into a preference list of the matching accounts.
 6: Initialize all unlabeled $u_i^{(1)}$ in $\mathcal{G}^{(1)}$ and $u_j^{(2)}$ in $\mathcal{G}^{(2)}$ as free
 7: $\mathcal{A}' = \emptyset$
 8: while $\exists$ free $u_i^{(1)}$ in $\mathcal{G}^{(1)}$ and $u_i^{(1)}$'s preference list is non-empty do
 9:     Remove the top-ranked account $u_j^{(2)}$ from $u_i^{(1)}$'s preference list
10:     if $u_j^{(2)}$ is free then
11:         $\mathcal{A}' = \mathcal{A}' \cup \{(u_i^{(1)}, u_j^{(2)})\}$
12:         Set $u_i^{(1)}$ and $u_j^{(2)}$ as occupied
13:     else
14:         $\exists u_p^{(1)}$ that $u_j^{(2)}$ is occupied with.
15:         if $u_j^{(2)}$ prefers $u_i^{(1)}$ to $u_p^{(1)}$ then
16:             $\mathcal{A}' = (\mathcal{A}' \setminus \{(u_p^{(1)}, u_j^{(2)})\}) \cup \{(u_i^{(1)}, u_j^{(2)})\}$
17:             Set $u_p^{(1)}$ as free and $u_i^{(1)}$ as occupied
18:         end if
19:     end if
20: end while
```

in network $G^{(1)}$ are occupied, or all the preference lists of free accounts in network $G^{(1)}$ are empty. Finally, the selected anchor links in set $\mathcal{A}'$ will be returned as the final network alignment result.

4.4 Supervised Partial Network Alignment

The method MNA introduced in the previous section assumes that the online social networks are fully aligned, and all the users in the networks are *anchor users*, who will all be connected by the anchor links in the final results. However, in the real world, such a strong assumption can hardly hold. These online social networks generally contain different groups of users, which can be highly likely to be partially aligned actually. For instance, As pointed out in [6], by the end of 2013, about 42% of online adults are using multiple social sites at the same time. Meanwhile, 93% of Instagram users are involved in Facebook concurrently and 53% Twitter users are using Instagram as well [16]. In other words, there still exist a large number of users who merely use Instagram or Twitter but are not involved in Facebook, which will make these online social networks partially aligned [37,38] instead.

4.4.1 Partial Network Alignment Description

Different from the "*supervised full network alignment*" problem, we will study a more general *partial network alignment* problem in this section. There exist several significant differences between these two problems, which are provided as follows to help the readers distinguish these two works. Firstly, the networks studied in this section are partially aligned [37], which contain a large number of anchor and non-anchor users [37] at the same time. Secondly, the networks studied here are not confined to the Foursquare and Twitter social networks, and a more general feature extraction method will be needed. We hope a minor revision of the "*partial network alignment*" problem can be mapped to many other existing tough problems, e.g., large biology network alignment [2], entity resolution in database integration [3], ontology matching [8], and various types of entity matching in online social networks [19]. Thirdly, due to the cardinality constraints on the anchor links as mentioned at the end of Sect. 4.3.2, the number of *positive* and *negative* anchor link instances across networks can be highly imbalanced, i.e., the negative training set is far larger than the positive training set. Such a class imbalance problem will make most of the existing supervised classification model fail to work. Finally, many of the users will stay isolated in the alignment results, since they can be non-anchor users actually. The constraint on *anchor links* is updated to "$one\text{-}to\text{-}one_{\leq}$" (called "*one-to-at-most-one*"), i.e., each user in one network can be mapped to at most one user in another network. Across partially aligned networks, only anchor users can be connected by anchor links. Therefore, identifying the non-anchor users from networks and pruning all the predicted potential anchor links connected to them is a novel yet challenging problem. The "$one\text{-}to\text{-}one_{\leq}$" constraint on anchor links can distinguish the "*partial network alignment*" problem from most existing link prediction problems. For example, in traditional link prediction and link transfer problems [20,21,37], the constraint on links is "*many-to-many*," while in the "*anchor link inference*" problem [12] across fully aligned networks, the constraint on *anchor links* is strictly "*one-to-one*."

The *supervised partial network alignment* problem studied in this section follows the identical definition as that introduced in Sect. 4.2, except that the anchor links will follow the "$one\text{-}to\text{-}one_{\leq}$" constraint instead. To address such a problem, in this section, we will introduce the PNA method proposed in [38], which contains three phases: (1) *general feature extraction based on meta paths*, (2) *class-imbalance anchor link classification*, and (3) *generic stable matching* to preserve the "$one\text{-}to\text{-}one_{\leq}$" constraint on anchor links.

4.4.2 Inter-Network Meta Path Based Feature Extraction

Different from the diverse features extracted for the Twitter and Foursquare networks specifically as introduced in Sect. 4.3.1, in this part, we will introduce two different general feature extraction methods across online social networks, including both the *meta path* [37] based explicit feature extraction and *tensor* [11] based latent feature extraction.

4.4.2.1 Inter-Network Meta Paths

The *inter-network meta path* has been introduced in the previous chapter already (detailed information is available in Sect. 3.5). Via the instances of *inter-network meta paths*, users across *aligned social networks* can be extensively connected to each other. In these two partially aligned online social networks (e.g., $\mathcal{G} = ((G^{(1)}, G^{(2)}), (\mathcal{A}^{(1,2)}))$) studied here, we can represent their network schemas in Fig. 4.6, where the *Word*, *Location*, and *Timestamp* node types are attached to the *Post* node type. For the network schema of network $G^{(1)}$, users may also create several lists, which can contain a bunch of location check-ins. Various *inter-network meta paths* between $G^{(1)}$ (i.e., Foursquare) and $G^{(2)}$ (i.e., Twitter) can be defined as follows:

- *Common Out Neighbor Inter-Network Meta Path* (Ψ_1): $User^{(1)} \xrightarrow{follow} User^{(1)} \xleftrightarrow{Anchor} User^{(2)} \xleftarrow{follow} User^{(2)}$ or "$\mathcal{U}^{(1)} \rightarrow \mathcal{U}^{(1)} \leftrightarrow \mathcal{U}^{(2)} \leftarrow \mathcal{U}^{(2)}$" for short.
- *Common In Neighbor Inter-Network Meta Path* (Ψ_2): $User^{(1)} \xleftarrow{follow} User^{(1)} \xleftrightarrow{Anchor} User^{(2)} \xrightarrow{follow} User^{(2)}$ or "$\mathcal{U}^{(1)} \leftarrow \mathcal{U}^{(1)} \leftrightarrow \mathcal{U}^{(2)} \rightarrow \mathcal{U}^{(2)}$."
- *Common Out In Neighbor Inter-Network Meta Path* (Ψ_3): $User^{(1)} \xrightarrow{follow} User^{(1)} \xleftrightarrow{Anchor} User^{(2)} \xrightarrow{follow} User^{(2)}$ or "$\mathcal{U}^{(1)} \rightarrow \mathcal{U}^{(1)} \leftrightarrow \mathcal{U}^{(2)} \rightarrow \mathcal{U}^{(2)}$."
- *Common In Out Neighbor Inter-Network Meta Path* (Ψ_4): $User^{(1)} \xleftarrow{follow} User^{(1)} \xleftrightarrow{Anchor} User^{(2)} \xleftarrow{follow} User^{(2)}$ or "$\mathcal{U}^{(1)} \leftarrow \mathcal{U}^{(1)} \leftrightarrow \mathcal{U}^{(2)} \leftarrow \mathcal{U}^{(2)}$."

Besides the users who can be shared across the online social networks, many other nodes can be shared across networks as well by capturing the identical physical information, like words, location latitude-longitude pairs, and timestamps, which are defined as the *bridge nodes* [38] across the aligned social networks.

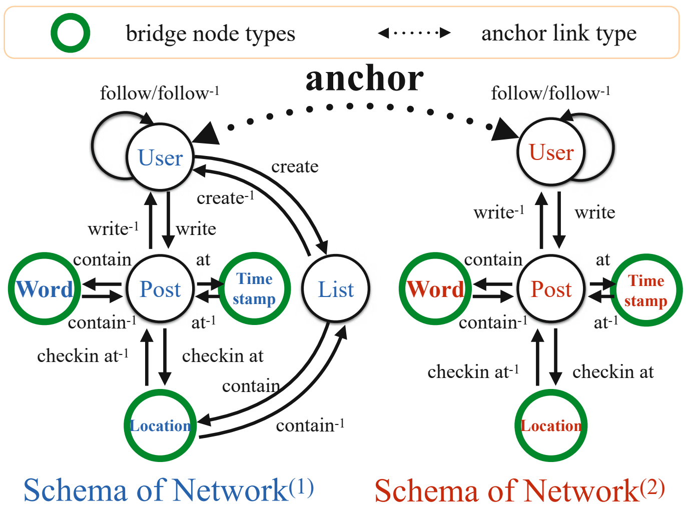


Fig. 4.6 Schema of aligned heterogeneous network

Definition 4.7 (Bridge Nodes) The *bridge nodes* shared between $G^{(1)}$ and $G^{(2)}$ can be represented as $\mathcal{B}^{(1,2)} = \{v | (v \in \mathcal{V}^{(1)} \setminus \mathcal{U}^{(1)}) \wedge (v \in \mathcal{V}^{(2)} \setminus \mathcal{U}^{(2)})\}$.

The *bridge node* concept is different from the *anchor node* concept defined before. Different from the concrete user information entities denoted by *anchor node*, the *bridge node* can get shared across networks merely because of their identical physical meanings (e.g., the words) or physical representations (e.g., the POI latitude-longitude pairs or timestamps).

For instance, in the network schema as shown in Fig. 4.6, the word node type, location node type, and timestamp node type are the bridge node types shared by the networks $G^{(1)}$ and $G^{(2)}$. These *inter-network meta paths* defined above all involve one single node type, i.e., the "User" node type, across the *partially aligned social networks*. In addition to these meta paths, there can exist many other *inter-network meta paths* consisting of user node type and other *bridge node* types from Foursquare to Twitter, e.g., Location, Word, and Timestamp.

- *Common Location Check-in Inter-Network Meta Path 1* (Ψ_5): $User^{(1)} \xrightarrow{write} Post^{(1)} \xrightarrow{check\text{-}in\ at} Location \xleftarrow{check\text{-}in\ at} Post^{(2)} \xleftarrow{write} User^{(2)}$ or "$\mathcal{U}^{(1)} \rightarrow \mathcal{P}^{(1)} \rightarrow \mathcal{L} \leftarrow \mathcal{P}^{(2)} \leftarrow \mathcal{U}^{(2)}$."
- *Common Location Check-in Inter-Network Meta Path 2* (Ψ_6): $User^{(1)} \xrightarrow{create} List^{(1)} \xrightarrow{contain} Location \xleftarrow{check\text{-}in\ at} Post^{(2)} \xleftarrow{write} User^{(2)}$ or "$\mathcal{U}^{(1)} \rightarrow \mathcal{I}^{(1)} \rightarrow \mathcal{L} \leftarrow \mathcal{P}^{(2)} \leftarrow \mathcal{U}^{(2)}$."
- *Common Timestamps Inter-Network Meta Path* (Ψ_7): $User^{(1)} \xrightarrow{write} Post^{(1)} \xrightarrow{at} Time \xleftarrow{at} Post^{(2)} \xleftarrow{write} User^{(2)}$ or "$\mathcal{U}^{(1)} \rightarrow \mathcal{P}^{(1)} \rightarrow \mathcal{T} \leftarrow \mathcal{P}^{(2)} \leftarrow \mathcal{U}^{(2)}$."
- *Common Word Usage Inter-Network Meta Path* (Ψ_8): $User^{(1)} \xrightarrow{write} Post^{(1)} \xrightarrow{contain} Word \xleftarrow{contain} Post^{(2)} \xleftarrow{write} User^{(2)}$ or "$\mathcal{U}^{(1)} \rightarrow \mathcal{P}^{(1)} \rightarrow \mathcal{W} \leftarrow \mathcal{P}^{(2)} \leftarrow \mathcal{U}^{(2)}$."

4.4.2.2 Explicit Inter-Network Adjacency Features Extraction

Based on the above defined *inter-network meta paths*, different kinds of inter-network meta path based adjacency relationship can be extracted from the network. Formally, a new concept of *inter-network adjacency score* has been defined to describe such relationships among users across *partially aligned social networks*.

Definition 4.8 (Inter-Network Meta Path Instance) Based on *inter-network meta path* $\Psi_i = T_1 \xrightarrow{R_1} T_2 \xrightarrow{R_2} \cdots \xrightarrow{R_{k-1}} T_k$, a concrete path $\psi = n_1 - n_2 - \ldots - n_{k-1} - n_k$ is an instance of Ψ_i iff n_j is an instance of node type T_j, $\forall j \in \{1, 2, \ldots, k\}$ and (n_j, n_{j+1}) is an instance of link type $T_j \xrightarrow{R_j} T_{j+1}$, $\forall j \in \{1, 2, \ldots, k-1\}$.

Definition 4.9 (Inter-Network Adjacency Score) The *inter-network adjacency* score is quantified as the number of concrete path instances of various *inter-network meta paths* connecting users across networks. The *inter-network adjacency score* between $u^{(1)} \in \mathcal{U}^{(1)}$ and $v^{(2)} \in \mathcal{U}^{(2)}$ based on meta path Ψ_i is defined as:

$$\text{score}_{\Psi_i}(u^{(1)}, v^{(2)}) = \left| \{\psi | (\psi \in \Psi_i) \wedge (u^{(1)} \in T_1) \wedge (v^{(2)} \in T_k)\} \right|, \tag{4.15}$$

where path ψ starts and ends with node types T_1 and T_k, respectively and $\psi \in \Psi_i$ denotes that ψ is a path instance of meta path Ψ_i.

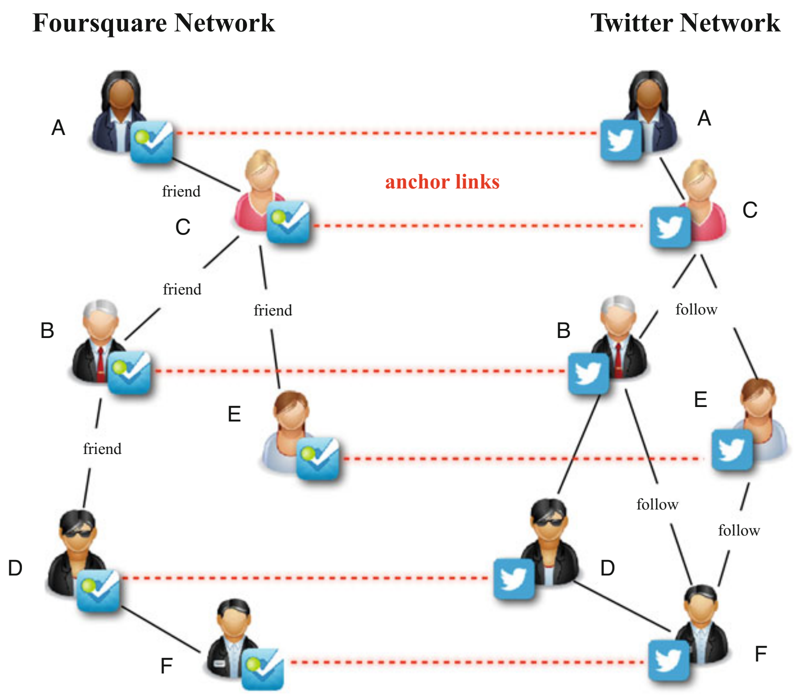


Fig. 4.7 An example of input aligned social networks

Example 4.8 For instance, given a pair of input aligned social networks as shown in Fig. 4.7, where the Foursquare network is $G^{(1)}$ and the Twitter network is $G^{(2)}$, two meta paths can be defined as follows:

1. $\mathcal{U}^{(1)} \rightarrow \mathcal{U}^{(1)} \leftrightarrow \mathcal{U}^{(2)} \leftarrow \mathcal{U}^{(2)}$
2. $\mathcal{U}^{(1)} \rightarrow \mathcal{U}^{(1)} \rightarrow \mathcal{U}^{(1)} \leftrightarrow \mathcal{U}^{(2)} \leftarrow \mathcal{U}^{(2)}$

For a random user pair, like (B_F, E_T), based on meta path 1, we identify one single path $B_F \rightarrow C_F \leftrightarrow C_T \leftarrow E_T$ connecting these two users. Meanwhile, based on meta path 2, there will exist another path connecting them, which includes $B_F \rightarrow D_F \rightarrow F_F \leftrightarrow F_T \leftarrow E_T$. In other words, the *inter-network adjacency score* between B_F and E_T based on meta paths 1 and 2 are both 1 actually.

The *anchor adjacency scores* among all users across *partially aligned networks* can be stored in the *anchor adjacency matrix* as follows:

Definition 4.10 (Inter-Network Adjacency Matrix) Given a certain *anchor meta path*, e.g., Ψ, the *anchor adjacency matrix* between networks $G^{(1)}$ and $G^{(2)}$ can be defined as $\mathbf{A}_\Psi \in \mathbb{N}^{|\mathcal{U}^{(1)}| \times |\mathcal{U}^{(2)}|}$ and $A_\Psi(l, m) = score_\Psi(u_l^{(1)}, u_m^{(2)})$ for users $u_l^{(1)} \in \mathcal{U}^{(1)}, u_m^{(2)} \in \mathcal{U}^{(2)}$.

Multiple *anchor adjacency matrix* can be grouped together to form a *high-order tensor*. A *tensor* [11] is a multidimensional array and an N-order *tensor* is an element of the tensor product of N vector spaces, each of which can have its own coordinate system. As a result, a 1-order *tensor* is a vector, a 2-order *tensor* is a matrix and *tensors* of three or higher order are called the *higher-order tensor* [11, 18].

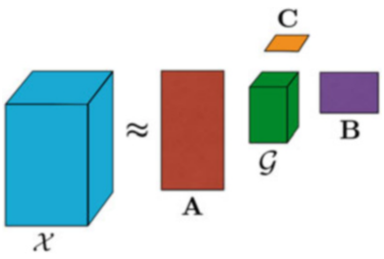


Fig. 4.8 Tensor decomposition

Definition 4.11 (Inter-Network Adjacency Tensor) Based on the eight defined *inter-network meta paths* $\{\Psi_1, \Psi_2, \ldots, \Psi_8\}$, a set of *anchor adjacency matrices* between users in two *partially aligned networks* can be obtained to be $\{\mathbf{A}_{\Psi_1}, \mathbf{A}_{\Psi_2}, \ldots, \mathbf{A}_{\Psi_8}\}$. With $\{\mathbf{A}_{\Psi_1}, \mathbf{A}_{\Psi_2}, \ldots, \mathbf{A}_{\Psi_8}\}$, a 3-order *anchor adjacency tensor* $\mathcal{X} \in \mathbb{R}^{|\mathcal{U}^{(1)}|\times|\mathcal{U}^{(2)}|\times 8}$ can be constructed, where the ith layer of $\mathcal{X}$ is the *anchor adjacency matrix* based on *anchor meta path* Ψ_i, i.e., $\mathcal{X}(:, :, i) = \mathbf{A}_{\Psi_i}, i \in \{1, 2, \ldots, 8\}$.

Based on the *anchor adjacency tensor*, a set of *explicit anchor adjacency features* can be extracted for *anchor links* across *partially aligned social networks*. For a certain *anchor link* $(u_l^{(1)}, u_m^{(2)})$, the *explicit anchor adjacency feature vectors* extracted based on the *anchor adjacency tensor* $\mathcal{X}$ can be represented as $\mathbf{x} = [x_1, x_2, \ldots, x_8]$ (i.e., the *anchor adjacency scores* between $u_l^{(1)}$ and $u_m^{(2)}$ based on these 8 different *anchor meta paths*), where $x_k = \mathcal{X}(l, m, k), k \in \{1, 2, \ldots, 8\}$.

4.4.2.3 Latent Topological Feature Vectors Extraction

Explicit anchor adjacency features can express manifest properties of the connections across *partially aligned networks* and are the *explicit topological features*. Besides these explicit topological connections, there can also exist some hidden common connection patterns [28] across *partially aligned networks*. In [38], a group of *latent topological feature vectors* are extracted from the *anchor adjacency tensor*.

As proposed in [11, 18], a *higher-order tensor* can be decomposed into a *core tensor*, e.g., $\mathcal{G}$, multiplied by a matrix along each mode, e.g., $\mathbf{A}, \mathbf{B}, \ldots, \mathbf{Z}$, with various *tensor decomposition methods*, e.g., Tucker decomposition [11]. For example, in Fig. 4.8, the *3-order anchor adjacency tensor* $\mathcal{X}$ can be decomposed into three matrices $\mathbf{A} \in \mathbb{R}^{|\mathcal{U}^{(1)}|\times P}$, $\mathbf{B} \in \mathbb{R}^{|\mathcal{U}^{(2)}|\times Q}$, and $\mathbf{C} \in \mathbb{R}^{8\times R}$ and a core tensor $\mathcal{G} \in \mathbb{R}^{P\times Q\times R}$, where P, Q, R are the number of columns of matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ [11]:

$$\mathcal{X} = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr}\mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r = [\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}], \tag{4.16}$$

where $\mathbf{a}_p \circ \mathbf{b}_q$ denotes the vector outer product of $\mathbf{a}_p$ and $\mathbf{b}_q$. Each row of $\mathbf{A}$ and $\mathbf{B}$ represents a *latent topological feature vector* of users in $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$, respectively [18].

4.4.3 Class-Imbalance Classification Model

Based on the *one-to-one*$_\leq$ cardinality constraint, the number of non-existing anchor links will be far more than the existing anchor links. In other words, the negative instances will be far more than that of the positive instances in both the training set and testing set of the supervised link prediction model. To overcome such a disadvantage, in this part, we will introduce several existing techniques that can be applied to sample or prune the training/testing sets.

4.4.3.1 Training Set Sampling

Based on the anchor adjacency scores calculated according to various anchor meta paths in previous section, various supervised link prediction models [12,36,37] can be built to infer the potential anchor links across networks. As proposed in [15, 17], conventional supervised link prediction methods [22] can suffer from the *class imbalance* problem a lot. To address the problem, two effective methods (*down sampling* [14] and *over sampling* [4]) can be applied.

Down sampling methods aim at deleting the *unreliable negative instances* from the training set. In Fig. 4.9, we show the distribution of training instances in the feature space, where negative instances can be generally divided into 4 different categories [14]:

- *noisy instances*: instances mixed in the positive instances;
- *borderline instances*: instances close to the decision boundary;
- *redundant instances*: instances which are too far away from the decision boundary in the negative region;
- *safe instances*: instances which are helpful for determining the classification boundary.

Different heuristics have been proposed to remove the *noisy instances* and *borderline instances*, like the *Tomek links* method proposed in [14, 23]. For any two given instances $\mathbf{x}_1$ and $\mathbf{x}_2$ of different labels, pair $(\mathbf{x}_1, \mathbf{x}_2)$ is called a *tomek link* if there exists no other instances, e.g., $\mathbf{z}$, such that $d(\mathbf{x}_1, \mathbf{z}) < d(\mathbf{x}_1, \mathbf{x}_2)$ and $d(\mathbf{x}_2, \mathbf{z}) < d(\mathbf{x}_1, \mathbf{x}_2)$. Examples that participate in *Tomek links* are either borderline or noisy instances [14, 23]. As to the *redundant instances*, they will not harm correct classifications as their existence will not change the classification boundary but they can lead to extra classification costs. To remove the *redundant instances*, a *consistent subset* $\mathcal{C}$ of the training set will be created, e.g., $\mathcal{S}$ [14]. Subset $\mathcal{C}$ is *consistent* with $\mathcal{S}$ if classifiers built with $\mathcal{C}$ can correctly classify instances in $\mathcal{S}$. Initially, $\mathcal{C}$ consists of all positive instances and one randomly selected negative instances. A classifier, e.g., kNN, built with $\mathcal{C}$ is applied to $\mathcal{S}$, where instances that are misclassified will be added into $\mathcal{C}$. The final set $\mathcal{C}$ only contains the *safe links*.

Another method to overcome the *class imbalance* problem in the training set is to *over sample* the *minority class*. Many *over sampling* methods have been proposed, e.g., *over sampling with replacement*, *over sampling with "synthetic" instances* [4]. Among them, the *over sampling with "synthetic" instances* is frequently used due to its effectiveness and wide usage in many scenarios [4]. The minority class is over sampled by introducing new "synthetic" examples along the line segment joining m of the k nearest minority class neighbors for each minority class instances. The value of parameter m can be determined according to the ratio to *over sample* the minority class. For example, if the minority class need to be *over sampled* by 200%, then $m = 2$. The instance to be created between a certain example $\mathbf{x}$ and one of its nearest neighbor $\mathbf{y}$ can be denoted as $\mathbf{x} + \boldsymbol{\theta}^\top(\mathbf{x} - \mathbf{y})$,

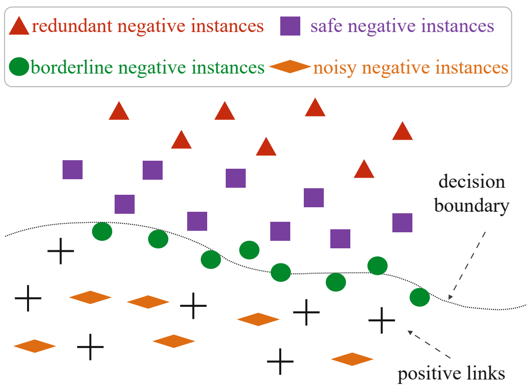


Fig. 4.9 Instance distribution in feature space

where $\mathbf{x}$ and $\mathbf{y}$ are the feature vectors of two instances and $\boldsymbol{\theta}^\top$ is the transpose of a coefficient vector containing random numbers in range [0, 1].

4.4.3.2 Test Set Pre-pruning

Across two *partially aligned social networks*, users in a certain network can have a large number of potential *anchor link candidates* in the other network, which can lead to great time and space costs in predicting the anchor links. The problem can be even worse when the networks are of large scales, e.g., containing millions or even billions of users, which can make the *partial network alignment* problem unsolvable. To shrink size of the candidate set, a *candidate pre-pruning* step of links in the test set will be conducted before applying the *class imbalance link prediction model*, i.e., $\mathcal{M}$, to links in the test set.

Each user in one network can have millions of potential *anchor link candidates* in another network. To address such a problem, the *candidate pre-pruning* step can be conducted on the test set $\mathcal{L}$ before applying the built model $\mathcal{M}$ to predict these links in $\mathcal{L}$. The *pre-pruning* method adopted here includes (1) *profile pre-pruning* and (2) *inter-network adjacency score pre-pruning*.

- *profile pre-pruning*: the profile information of users shared across *partially aligned social networks*, e.g., Foursquare and Twitter, can include username and hometown [30]. Given an anchor link $(u_l^{(1)}, u_m^{(2)}) \in \mathcal{A}_{test}$, if the username and hometown of $u_l^{(1)}$ and $u_m^{(2)}$ are totally different, e.g., cosine similarity scores are 0, then link $(u_l^{(1)}, u_m^{(2)})$ will be pruned from the testing set $\mathcal{A}_{test}$.
- *inter-network adjacency score pruning*: based on the *explicit inter-network adjacency tensor* $\mathcal{X}$ introduced in the previous sections, for a given link $(u_l^{(1)}, u_m^{(2)}) \in \mathcal{A}_{test}$, if its extracted *explicit inter-network adjacency features* are all 0, i.e., $\mathcal{X}(l, m, x) = 0, x \in \{1, 2, \ldots, 8\}$, then link $(u_l^{(1)}, u_m^{(2)})$ will be pruned from the testing set $\mathcal{A}_{test}$.

Example 4.9 For example, in Fig. 4.10, we give 6 users in two different networks together with all the 9 potential *anchor links* between them in the test set, where "*William*" and "*Wm*" are the same

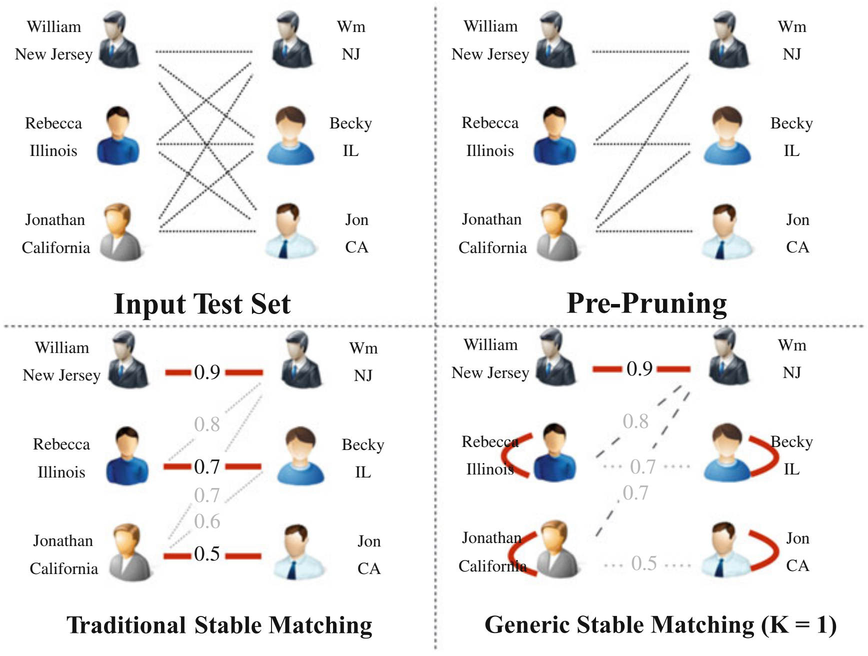


Fig. 4.10 Partial network alignment with pruning

user in these two networks and the remaining users are all different users. Users' profile information (i.e., names and hometowns) is given in the figure. By applying the *profile pre-pruning* method, we can remove 3 *anchor links* from the test set. The advantages of the *pre-pruning* will be more significant when being applied to very large-scale real-world *partially aligned social networks*.

4.4.4 Generic Stable Matching

Given the user sets $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$ of two *partially aligned social networks* $G^{(1)}$ and $G^{(2)}$, each user in $\mathcal{U}^{(1)}$(or $\mathcal{U}^{(2)}$) has his preference over users in $\mathcal{U}^{(2)}$(or $\mathcal{U}^{(1)}$). Term $v_j P_{u_i}^{(1)} v_k$ can be used to denote that $u_i \in \mathcal{U}^{(1)}$ prefers v_j to v_k for simplicity, where $v_j, v_k \in \mathcal{U}^{(2)}$ and $P_{u_i}^{(1)}$ is the preference operator of $u_i \in \mathcal{U}^{(1)}$. Similarly, we can use term $u_i P_{v_j}^{(2)} u_k$ to denote that $v_j \in \mathcal{U}^{(2)}$ prefers u_i to u_k in $\mathcal{U}^{(1)}$ as well.

Stable matching based method [12] introduced for MNA in Sect. 4.3.3 can only work well in *fully aligned social networks*. However, in the real world, few social networks are fully aligned and lots of users in social networks are involved in one network only, i.e., *non-anchor users*, and they should not be connected by any anchor links. However, traditional *stable matching* method cannot identify these *non-anchor users* and remove the predicted *potential anchor links* connected with them. To overcome such a problem, the *generic stable matching* has been introduced in [38] to identify the *non-anchor users* and prune the anchor link results to meet the $one\text{-}to\text{-}one_{\leq}$ constraint.

In partial network matching method PNA [38], a novel concept, *self-matching*, has been introduced, which allows users to be mapped to themselves if they are discovered to be *non-anchor users*. In other words, the *non-anchor users* will be identified as those who are mapped to themselves in the final matching results.

Definition 4.12 (Self-matching) For the given two partially aligned networks $G^{(1)}$ and $G^{(2)}$, user $u_i \in \mathcal{U}^{(1)}$ can have his preference $P_{u_i}^{(1)}$ over users in $\mathcal{U}^{(2)} \cup \{u_i\}$ and u_i preferring u_i himself denotes that u_i is a *non-anchor user* and prefers to stay unconnected, which is formally defined as *self-matching*.

Users in one social network will be matched with either partners in other social networks or themselves according to their preference lists (i.e., from high preference scores to low preference scores). Only partners that users prefer over themselves will be *accepted* finally, otherwise users will be matched with themselves instead.

Definition 4.13 (Acceptable Partner) For a given *matching* $\mu : \mathcal{U}^{(1)} \cup \mathcal{U}^{(2)} \to \mathcal{U}^{(1)} \cup \mathcal{U}^{(2)}$, the mapped partner of users $u_i \in \mathcal{U}^{(1)}$, i.e., $\mu(u_i)$, is *acceptable* to u_i iff $\mu(u_i) P_{u_i}^{(1)} u_i$.

To cut off the partners with very low *preference scores*, the *partial matching strategy* is proposed to obtain the promising partners, who will participate in the matching finally.

Definition 4.14 (Partial Matching Strategy) The *partial matching strategy* of user $u_i \in \mathcal{U}^{(1)}$, i.e., $Q_{u_i}^{(1)}$, consists of the first K the *acceptable partners* in u_i's preference list $P_{u_i}^{(1)}$, which are in the same order as those in $P_{u_i}^{(1)}$, and u_i in the $(K+1)$th entry of $Q_{u_i}^{(1)}$. Parameter K is called the *partial matching rate* as introduced in [38].

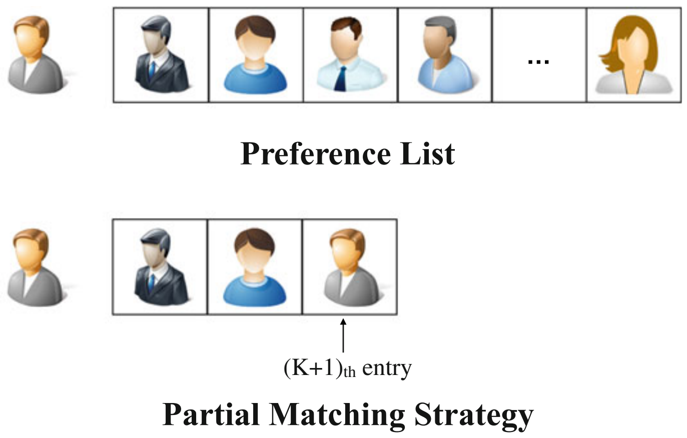


Fig. 4.11 An example of partial matching strategy ($K = 2$)

Example 4.10 An example is given in Fig. 4.11, where to get the top two promising partners for the user, the user himself is placed at the third cell in the preference list. All the remaining potential partners will be cut off and only the top three users will participate in the final matching.

Based on the concepts of *self-matching* and *partial matching strategy*, the concepts of *partial stable matching* and *generic stable matching* can be defined as follows:

Definition 4.15 (Partial Stable Matching) For a given *matching* μ, μ is (1) *rational* if $\mu(u_i) Q^{(1)}_{u_i} u_i, \forall u_i \in \mathcal{U}^{(1)}$ and $\mu(v_j) Q^{(2)}_{v_j} v_j, \forall v_j \in \mathcal{U}^{(2)}$, (2) *pairwise stable* if there exist no *blocking pairs* in the matching results, and (3) *stable* if it is both *rational* and *pairwise stable*.

Definition 4.16 (Generic Stable Matching) For a given *matching* μ, μ is a *generic stable matching* iff μ is a *self-matching* or μ is a *partial stable matching*.

Example 4.11 An example of *generic stable matching* is shown in the bottom two plots of Fig. 4.10. *Traditional stable matching* can prune most non-existing anchor links and make sure the results can meet the *one-to-one* constraint. However, it preserves the anchor links (Rebecca, Becky) and (Jonathan, Jon), which are connecting *non-anchor users* actually. In the *generic stable matching* with parameter $K = 1$, users will be either connected with their most preferred partner or stay *unconnected*. Users “William” and “Wm” are matched as link (William, Wm) with the highest score. “Rebecca” and “Jonathan” will prefer to stay *unconnected* as their most preferred partner “Wm” is connected with “William” already. Furthermore, “Becky” and “Jon” will stay *unconnected* as their most preferred partners “Rebecca” and “Jonathan” prefer to stay *unconnected*. In this way, *generic stable matching* can further prune the non-existing anchor links (Rebecca, Becky) and (Jonathan, Jon).

The *generic stable matching* results can be achieved with the *Generic Gale-Shapley* algorithm, whose pseudo-code is available in Algorithm 2.

Algorithm 2 Generic Gale-Shapley algorithm

```
Require: user sets of aligned networks: $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$.
         classification results of potential anchor links in $\mathcal{L}$
         known anchor links in $\mathcal{A}^{(1,2)}$
         truncation rate $K$
Ensure: a set of inferred anchor links $\mathcal{L}'$
 1: Initialize the preference lists of users in $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$ with predicted existence probabilities of links in $\mathcal{L}$ and known
    anchor links in $\mathcal{A}^{(1,2)}$, whose existence probabilities are 1.0
 2: construct the truncated strategies from the preference lists
 3: Initialize all users in $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$ as free
 4: $\mathcal{L}' = \emptyset$
 5: while $\exists$ free $u_i^{(1)}$ in $\mathcal{U}^{(1)}$ and $u_i^{(1)}$'s truncated strategy is non-empty do
 6:   Remove the top-ranked account $u_j^{(2)}$ from $u_i^{(1)}$'s truncated strategy
 7:   if $u_j^{(2)}$==$u_i^{(1)}$ then
 8:     $\mathcal{L}' = \mathcal{L}' \cup \{(u_i^{(1)}, u_i^{(1)})\}$
 9:     Set $u_i^{(1)}$ as stay unconnected
10:   else
11:     if $u_j^{(2)}$ is free then
12:       $\mathcal{L}' = \mathcal{L}' \cup \{(u_i^{(1)}, u_j^{(2)})\}$
13:       Set $u_i^{(1)}$ and $u_j^{(2)}$ as occupied
14:     else
15:       $\exists u_p^{(1)}$ that $u_j^{(2)}$ is occupied with.
16:       if $u_j^{(2)}$ prefers $u_i^{(1)}$ to $u_p^{(1)}$ then
17:         $\mathcal{L}' = (\mathcal{L}' - \{(u_p^{(1)}, u_j^{(2)})\}) \cup \{(u_i^{(1)}, u_j^{(2)})\}$
18:         Set $u_p^{(1)}$ as free and $u_i^{(1)}$ as occupied
19:       end if
20:     end if
21:   end if
22: end while
```

4.5 Anchor Link Inference with Cardinality Constraint

In the previous two sections, the anchor links are assumed to be subject to the *one-to-one* and "*one-to-one*$_{\leq}$" constraints, respectively. Besides these two cases, users can also have multiple accounts in one social network. For instance, some people will create multiple accounts in Facebook, and tend to use different accounts to socialize with different groups of online friends. In such a scenario, the *cardinality constraint* on the anchor links will become *many-to-many* or *one-to-many*. In this section, we will introduce a model ITERCLIPS [43], which can infer the anchor links with a general *cardinality constraint*, covering *one-to-one*, *one-to-many*, and *many-to-many* simultaneously. The problem definition is identical to that introduced in Sect. 4.2, except that the positive and negative instances labels become $+1$ and 0, respectively. Actually, the ITERCLIPS model to be introduced here can not only infer the anchor links, but also be applied to solve many other types of link prediction tasks.

4.5.1 Loss Function for Anchor Link Prediction

Let set $\mathcal{L} = \mathcal{U}^{(1)} \times \mathcal{U}^{(2)}$ denote all the potential anchor links between networks $G^{(1)}$ and $G^{(2)}$, where $\mathcal{L} = \mathcal{A}_{train} \cup \mathcal{A}_{test}$. Based on the whole link set $\mathcal{L}$, as introduced in the previous sections, a set of features can be extracted for these links with the information available across the social networks, which can be represented as set $\mathcal{X} = \{\mathbf{x}_l\}_{l \in \mathcal{L}}$ ($\mathbf{x}_l \in \mathbb{R}^m, \forall l \in \mathcal{L}$). Given the link existence label set $\mathcal{Y} = \{0, 1\}$, the objective of the anchor link prediction problem is to achieve a general link inference function $f : \mathcal{X} \to \mathcal{Y}$ to map the link feature vectors to their corresponding labels. Here, 0 denotes the label of the negative class, and $+1$ denotes the label of the positive class. Depending on the specific application settings and information available in the networks, the feature vectors extracted for links in $\mathcal{L}$ can be very diverse. Various explicit and latent features introduced in the previous sections can be applied, and we will not repeat them here.

Formally, the loss introduced in the mapping $f(\cdot)$ can be represented as function $L : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ over the link feature vector/label pairs. Meanwhile, for one certain input feature vector $\mathbf{x}_l$ of link $l \in \mathcal{L}$, its inferred label introducing the minimum loss can be denoted as:

$$\hat{y}_l = \arg \min_{y_l \in \mathcal{Y}, \mathbf{w}} L(\mathbf{x}_l, y_l; \mathbf{w}), \tag{4.17}$$

where vector $\mathbf{w}$ represents the parameters involved in the mapping function $f(\cdot)$.

Therefore, given the pre-defined loss function $L(\cdot)$, the general form of the objective mapping $f : \mathcal{X} \to \mathcal{Y}$ parameterized by vector $\mathbf{w}$ can be represented as:

$$f(\mathbf{x}; \mathbf{w}) = \arg \min_{y_l \in \mathcal{Y}} L(\mathbf{x}, y; \mathbf{w}). \tag{4.18}$$

In many cases (e.g., when the links are not linearly separable), the feature vector $\mathbf{x}_l$ of link l needs to be transformed as $g(\mathbf{x}_l) \in \mathbb{R}^k$ (k is the transformed feature number) and the transformation function $g(\cdot)$ can be different *kernel projections* depending on the separability of instances. Here, we can assume loss function $L(\cdot)$ to be linear based on some combined representation of the transformed link feature vector $g(\mathbf{x}_l)^\top$ and label y_l, i.e.,

$$L(\mathbf{x}_l, y_l; \mathbf{w}) = (\langle \mathbf{w}, g(\mathbf{x}_l) \rangle - y_l)^2 = (\mathbf{w}^\top g(\mathbf{x}_l) - y_l)^2. \tag{4.19}$$

Furthermore, based on all the links in $\mathcal{L}$, the extracted feature vectors for these links can be represented as matrix $\mathbf{X} = [g(\mathbf{x}_{l_1}), g(\mathbf{x}_{l_2}), \ldots, g(\mathbf{x}_{l_{|\mathcal{L}|}})]^\top \in \mathbb{R}^{|\mathcal{L}| \times k}$ (for simplicity, linear kernel projection can used here, and $g(\mathbf{x}_l) = \mathbf{x}_l$). Meanwhile, their existence labels can be represented as vector $\mathbf{y} = [y_{l_1}, y_{l_2}, \ldots, y_{l_{|\mathcal{L}|}}]^\top$, where $y_l \in \{0, 1\}, \forall l \in \mathcal{L}$. Specifically, for the existing links in $\mathcal{A}^+_{train}$, their labels are known to be positive in advance, i.e., $y_l = 1, \forall l \in \mathcal{A}^+_{train}$. According to the above loss function definition, based on $\mathbf{X}$ and $\mathbf{y}$, the loss introduced by all links in $\mathcal{L}$ can be represented to be

$$L(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 . \tag{4.20}$$

To learn the parameter vector $\mathbf{w}$ and infer the potential label vector $\mathbf{y}$, the loss term introduced by all the links in $\mathcal{L}$ will be minimized. Meanwhile, to avoid overfitting the training set, besides minimizing the loss function $L(\mathbf{X}, \mathbf{y}; \mathbf{w})$, a regularization term $\|\mathbf{w}\|_2^2$ about the parameter vector $\mathbf{w}$ is added to the objective function:

$$\begin{aligned} &\min_{\mathbf{w}, \mathbf{y}} \frac{1}{2} \|\mathbf{w}\|_2^2 + \frac{c}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 , \\ &s.t. \ \ \mathbf{y} \in \{0, 1\}^{|\mathcal{L}| \times 1}, \text{ and } y_l = 1, \forall l \in \mathcal{A}^+_{train}, \end{aligned} \tag{4.21}$$

where constant c denotes the weight of the loss term in the function.

4.5.2 Cardinality Constraint Description

The *cardinality constraints* define both the limit on link cardinality and the limit on node degrees that those links are incident to. To be general, the links studied can be either uni-directional or bi-directional, where uni-directional links are treated as bi-directional. For each node $u \in \mathcal{V}$ in the network, we can represent the potential links going-out from u as set $\Gamma^{out}(u) = \{l | l \in \mathcal{L}, \exists v \in \mathcal{V}, l = (u, v)\}$, and those going-into u as set $\Gamma^{in}(u) = \{l | l \in \mathcal{L}, \exists v \in \mathcal{V}, l = (v, u)\}$. Furthermore, with the link label variables $\{y_l\}_{l \in \mathcal{L}}$, we can represent the out-degree and in-degree of node $u \in \mathcal{V}$ as $degree^{out}(u) = \sum_{l \in \Gamma^{out}(u)} y_l$ and $degree^{in}(u) = \sum_{l \in \Gamma^{in}(u)} y_l$, respectively. Considering that the node degrees cannot be negative, besides the upper bounds introduced by the *cardinality constraints*, a lower bound "≥ 0" is also added to guarantee validity of node degrees by default.

One-to-One Cardinality Constraint
For the bi-directional anchor links with 1:1 *cardinality constraint*, the nodes in the information networks can be attached with at most one such kinds of link. In other words, for all the nodes (e.g., $u \in \mathcal{V}$) in the network, their in-degree and out-degree cannot exceed 1, i.e.,

$$0 \leq \sum_{l \in \Gamma^{out}(u)} y_l \leq 1, \forall u \in \mathcal{V}, \text{ and } 0 \leq \sum_{l \in \Gamma^{in}(u)} y_l \leq 1, \forall u \in \mathcal{V}. \tag{4.22}$$

One-to-Many Cardinality Constraint
Meanwhile, when aligning two network structured data sources, where the cardinality constraint on *anchor links* is *one-to-many*. For instance, in some social networks, they may require the SSN or ID number from users in registration and each user can only contain one single account. The alignment of such a social network with the general social networks without such a limitation will be subject to the *one-to-many* cardinality constraint instead. In such a case, for all the nodes (e.g., $u \in \mathcal{V}$) in the network, their *out-degree* cannot exceed N and the *in-degree* should be exactly 1, i.e.,

$$0 \leq \sum_{l \in \Gamma^{out}(u)} y_l \leq N, \forall u \in \mathcal{V}, \text{ and } 1 \leq \sum_{l \in \Gamma^{in}(u)} y_l \leq 1, \forall u \in \mathcal{V}. \tag{4.23}$$

Many-to-Many Cardinality Constraint
In many cases, there usually exist no specific *cardinality constraints* on the anchor links, and nodes can be connected with each other freely. Simply, we assume the node *in-degrees* and *out-degrees* to be limited by the maximum degree parameter $N = |\mathcal{V}| - 1$, i.e.,

$$0 \leq \sum_{l \in \Gamma^{out}(u)} y_l \leq N, \forall u \in \mathcal{V}, \text{ and } 0 \leq \sum_{l \in \Gamma^{in}(u)} y_l \leq N, \forall u \in \mathcal{V}. \tag{4.24}$$

General Cardinality Constraint Representation
The *cardinality constraint* on links can be generally represented with the linear algebra equations. The relationship between nodes $\mathcal{V}$ and links $\mathcal{L}$ can actually be represented as matrices $\mathbf{T}^{out} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{L}|}$ and $\mathbf{T}^{in} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{L}|}$, where entry $\mathbf{T}^{out}(u, l) = 1$ iff $l \in \Gamma^{out}(u)$ and $\mathbf{T}^{in}(u, l) = 1$ iff $l \in \Gamma^{in}(u)$. Based on the link label vector $\mathbf{y}$, the node out-degrees and in-degrees can be represented as vectors $\mathbf{T}^{out} \cdot \mathbf{y}$ and $\mathbf{T}^{in} \cdot \mathbf{y}$, respectively. The general representation of the *cardinality constraints* introduced above can be rewritten as follows:

$$\underline{\mathbf{b}}^{out} \preccurlyeq \mathbf{T}^{out} \cdot \mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{out}, \text{ and } \underline{\mathbf{b}}^{in} \preccurlyeq \mathbf{T}^{in} \cdot \mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{in}, \tag{4.25}$$

where vectors $\underline{\mathbf{b}}^{out}$, $\overline{\mathbf{b}}^{out}$, $\underline{\mathbf{b}}^{in}$, and $\overline{\mathbf{b}}^{in}$ can take different values depending on the cardinality constraint on the links (e.g., for the 1:1 constraint, the bounds will have values $\underline{\mathbf{b}}^{out} = \underline{\mathbf{b}}^{in} = \mathbf{0}$ and $\overline{\mathbf{b}}^{out} = \overline{\mathbf{b}}^{in} = \mathbf{1}$).

4.5.3 Joint Optimization Function

Based on the above remarks, the constrained optimization objective function of the problem can be represented as

$$\begin{aligned} &\min_{\mathbf{w},\mathbf{y}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{c}{2}\|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2, \\ &s.t.\ \mathbf{y} \in \{0,1\}^{|\mathcal{L}|\times 1}, y_l = 1, \forall l \in \mathcal{E}, \\ &\quad \underline{\mathbf{b}}^{out} \preccurlyeq \mathbf{T}^{out}\cdot\mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{out}, \underline{\mathbf{b}}^{in} \preccurlyeq \mathbf{T}^{in}\cdot\mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{in}. \end{aligned} \tag{4.26}$$

The above objective function involves variables $\mathbf{w}$ and $\mathbf{y}$ at the same time, which is actually not jointly convex and can be very challenging to solve. In [43], the proposed ITERCLIPS model addresses the function with an alternative updating framework by fixing one variable and updating the other one iteratively. The framework involves two steps:

Step 1: Fix $\mathbf{y}$ and Update $\mathbf{w}$

By fixing $\mathbf{y}$ (i.e., treating $\mathbf{y}$ as a constant vector), the objective function about $\mathbf{w}$ can be simplified as

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{c}{2}\|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2. \tag{4.27}$$

Let $h(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{c}{2}\|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2$. By taking the derivative of the function $h(\mathbf{w})$ regarding $\mathbf{w}$ we can have

$$\frac{\mathrm{d}h(\mathbf{w})}{\mathrm{d}\mathbf{w}} = \mathbf{w} + c\mathbf{X}\mathbf{w}\mathbf{X}^\top - c\mathbf{y}\mathbf{X}^\top. \tag{4.28}$$

By making the derivation to be zero, the optimal vector $\mathbf{w}$ can be represented to be

$$\mathbf{w} = c(\mathbf{I} + c\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}, \tag{4.29}$$

and the minimum value of the function will be

$$\frac{c}{2}\mathbf{y}^\top\mathbf{y} - \frac{c^2}{2}\mathbf{y}^\top\mathbf{X}(\mathbf{I} + c\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}. \tag{4.30}$$

Step 2: Fix $\mathbf{w}$ and Update $\mathbf{y}$

When fixing $\mathbf{w}$ and treating it as a constant vector, the objective function about $\mathbf{y}$ can be represented as

$$\begin{aligned} &\min_{\mathbf{y}} \frac{c}{2}\|\hat{\mathbf{y}}-\mathbf{y}\|_2^2, \\ &s.t.\ \mathbf{y} \in \{0,1\}^{|\mathcal{L}|\times 1}, y_l = 1, \forall l \in \mathcal{E}, \\ &\quad \underline{\mathbf{b}}^{out} \preccurlyeq \mathbf{T}^{out}\cdot\mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{out}, \underline{\mathbf{b}}^{in} \preccurlyeq \mathbf{T}^{in}\cdot\mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{in}, \end{aligned} \tag{4.31}$$

Algorithm 3 Greedy link selection

```
Require: link estimate result $\hat{\mathbf{y}}$, parameter $k$
Ensure: link label vector $\mathbf{y}$
 1: initialize link label vector $\mathbf{y} = \mathbf{0}$
 2: for $l \in \mathcal{E}$ do
 3:     $y_l = 1$
 4: end for
 5: for $l \in \mathcal{L} \setminus \mathcal{E}$ and $\hat{y}_l < 0.5$ do
 6:     $y_l = 0$
 7: end for
 8: Let $\tilde{\mathcal{L}} = \{l | l \in \mathcal{L} \setminus \mathcal{E}, \hat{y}_l \geq 0.5\}$
 9: while $\tilde{\mathcal{L}} \neq \emptyset$ do
10:     select $l \in \tilde{\mathcal{L}}$ with the highest estimation score
11:     if add $l$ as positive instance violates the cardinality constraint or more than $k$ links have been selected then
12:         $y_l = 0$
13:     else
14:         $y_l = 1$
15:     end if
16: end while
17: return $\mathbf{y}$
```

Algorithm 4 Cardinality constrained anchor link prediction framework

```
Require: link feature vector $\mathbf{X}$
         weight parameter $c$
Ensure: parameter vector $\mathbf{w}$, link label vector $\mathbf{y}$
 1: Initialize label vector $\mathbf{y} = \frac{1}{2} \cdot \mathbf{1}$
 2: For links in $\mathcal{E}$, assign their label as 1
 3: Initialize parameter vector $\mathbf{w} = \mathbf{0}$
 4: Initialize convergence-tag = False
 5: while convergence-tag == False do
 6:     Update vector $\mathbf{w}$ with equation $\mathbf{w} = c(\mathbf{I} + c\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$
 7:     Calculate link estimation result $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
 8:     Update vector $\mathbf{y}$ with Algorithm Greedy($\hat{\mathbf{y}}$)
 9:     if $\mathbf{w}$ and $\mathbf{y}$ both converge then
10:         convergence-tag = True
11:     end if
12: end while
```

where $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ denotes the inference results of the links in $\mathcal{L}$ with the updated parameter vector $\mathbf{w}$ from Step 1. The objective function is a constrained non-linear integer programming problem about variable $\mathbf{y}$. Formally, the above optimization sub-problem is named as the **C**LS (Cardinality Constrained Link Selection) problem [43]. The **C**LS problem is shown to be NP-hard (we will analyze it in the next subsection), and achieving the optimal solution to it is very time consuming. To preserve the *cardinality constraints* on the variables and minimize the loss term, one brute-force way to achieve the optimal solution $\mathbf{y}$ is to enumerate all the feasible combination of links candidates to be selected as the positive instances, which will lead to very high time complexity. In [43], a greedy link selection algorithm is proposed to resolve the problem, and the pseudo-code of the greedy link selection method is available in Algorithm 3. Meanwhile, the framework is illustrated with the pseudo-code available in Algorithm 4. The framework updates vectors $\mathbf{w}$ and $\mathbf{y}$ alternatively until both of them converge.

4.5.4 Problem and Algorithm Analysis

In this part, we will show the **C**LS problem with *M*:*N cardinality constraints* can be reduced to the *k-maximum weighted matching problem* [7], which is NP-hard and not solvable in polynomial time. In addition, we will also prove that the greedy method can actually achieve a $\frac{1}{2}$-approximation of the optimal result of the **C**LS problem.

In the **C**LS problem, for all the existing links in $\mathcal{A}^+_{train}$, we know their label should be 1 in advance. For all the links in $\mathcal{L} \setminus \mathcal{A}^+_{train}$ with estimation score (i.e., $\hat{y}$) lower than 0.5, assigning their label with value 0 will introduce less loss and has no impact on the cardinality constraints. Therefore, in Algorithm 3, these links are handled in advance to simplify the problem. For the remaining links, we need to select those with high scores to assign with label 1 (so as to minimize the loss term), and preserve the *cardinality constraints* at the same time. For the links selection of which violate the cardinality constraints, they will be assigned with label 0 instead.

Formally, we can represent the unlabeled links with confidence scores greater than 0.5 as set $\tilde{\mathcal{L}} = \{l | l \in \mathcal{L} \setminus \mathcal{A}^+_{train}, \hat{y}_l > 0.5\}$. For all the links in set $\tilde{\mathcal{L}}$, the introduced loss term can be represented as

$$\sum_{l \in \tilde{\mathcal{L}}} (\hat{y}_l - y_l)^2 = \sum_{l \in \tilde{\mathcal{L}}} \hat{y}_l^2 + \sum_{l \in \tilde{\mathcal{L}}} y_l^2 - \sum_{l \in \tilde{\mathcal{L}}} 2\hat{y}_l \cdot y_l, \tag{4.32}$$

where term $\sum_{l \in \tilde{\mathcal{L}}} \hat{y}_l^2$ is a constant, term $\sum_{l \in \tilde{\mathcal{L}}} y_l^2$ denotes the number of selected links, and $\sum_{l \in \tilde{\mathcal{L}}} 2\hat{y}_l \cdot y_l$ represents the confidence scores of the selected links. Let's assume k links are selected finally, i.e., $\sum_{l \in \tilde{\mathcal{L}}} y_l^2 = k$, the optimal k links which can minimize the loss term can be achieved by maximizing the confidence scores of the selected links:

$$\begin{aligned} &\max \sum_{l \in \tilde{\mathcal{L}}} \hat{y}_l y_l \\ &s.t.\ \ y_l \in \{0, 1\}, \forall l \in \tilde{\mathcal{L}}, \sum_{l \in \tilde{\mathcal{L}}} y_l = k, \\ &\quad \underline{\mathbf{b}}^{out} \preccurlyeq \mathbf{T}^{out} \cdot \mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{out}, \underline{\mathbf{b}}^{in} \preccurlyeq \mathbf{T}^{in} \cdot \mathbf{y} \preccurlyeq \overline{\mathbf{b}}^{in}. \end{aligned} \tag{4.33}$$

By enumerating different k values in range $[1, |\tilde{\mathcal{L}}|]$, the optimal link set can be identified for the **C**LS problem.

Theorem 4.1 *The k-maximum weighted matching problem can be reduced to the above optimization problem with a general M*:*N cardinality constraint.*

Proof The above optimization problem with 1:1 *cardinality constraint* is actually identical to the *k-maximum weighted matching problem* studied in the existing works [7], and the reduction is trivial. Meanwhile, for the above optimization problem with N:1 *cardinality constraints* on the links, we can have vectors $\overline{\mathbf{b}}^{in} = \underline{\mathbf{b}}^{in} = [1, 1, \dots, 1]^\top$, $\overline{\mathbf{b}}^{out} = [N, N, \dots, N]^\top$, and $\underline{\mathbf{b}}^{out} = \mathbf{0}$. Given the information network G with $1 : N$ *cardinality constraints*, N dummy nodes can be constructed for each the nodes with out-going links. The constructed dummy nodes are connected to the original nodes to indicate the belonging relationships. For each original link, e.g., (u, v), a dummy directed link connecting the dummy node created for u with node v will be added, whose weight is identical to the weight of the original link (u, v). Given the *k-maximum weighted matching* result on the constructed dummy network, the optimal solution to the above optimization problem on network G can be obtained by replacing all the created dummy nodes with the original nodes corresponding to them. Meanwhile, for any solution to the above optimization problem on network G, the solution to the *k-maximum weighted matching* problem can be obtained on the constructed dummy network. In other words, the *k-maximum weighted matching problem* can be reduced to the above optimization problem with N:1 *cardinality constraints* via the constructed dummy network. Meanwhile, for the networks with general M:N constraint, dummy nodes can be created for both nodes with out-going links and in-coming links at the same time, whose reduction to the *k-maximum weighted matching problem* is not provided due to the limited space. In addition, in the M:N case, to avoid the case that solutions pick links connecting the more than one link connecting the dummy nodes corresponding the common node pairs (e.g., both (u', v') and (u'', v'') are selected, where u', u'' and v', v'' are the dummy nodes of u and v, respectively), more constraints will be added to the objective function of the *k-maximum weighted matching problem*.

According to the existing works [9], the *k-maximum weighted matching problem* is actually NP-hard. To address the problem efficiently, a greedy link selection method is applied as introduced in the previous subsection. As shown in Algorithm 3, among all the remaining links in $\tilde{\mathcal{L}}$, the greedy link selection method picks the links with the highest confidence scores $\hat{y}_l$. If the selection of a link doesn't violate the *cardinality constraint*, the greedy method will add it to the final result. We will show that the method can actually achieve a $\frac{1}{2}$-approximation of the optimal result.

Theorem 4.2 *The greedy method can achieve a $\frac{1}{2}$-approximation of the optimal solution to the* **C**LS *problem.*

Proof Formally, let $\mathcal{C}$ be the set of links selected by the greedy method to assign with label $+1$, while the optimal solution to the **C**LS problem can be represented as OPT. Every time, when the method selects the links with the highest confidence score (e.g., $l = (u, v)$) to add to $\mathcal{C}$, the degrees of nodes u and v will get increased by 1 and some other links incident to u, v will no longer get added to $\mathcal{C}$ due to the degree limit (introduced by the *cardinality constraint*). At most two links incident to u and v can get removed due to the selection of (u, v), since (u, v) occupies the degree space of u and v by one, respectively. Formally, the set of links incident to l can be represented as set $\Gamma(l) = \Gamma^{out}(u) \cup \Gamma^{in}(v)$. Depending on whether link $l \in \mathcal{C}$ is in OPT or not and the number of links in the optimal solution but are removed in $\mathcal{C}$ due to the selection of l (i.e., links in $\Gamma(l) \cap (OPT \setminus \mathcal{C})$), there exist three cases:

1. $l \in OPT$: Link l also belongs to the optimal result, and adding l into $\mathcal{C}$ will not affect the selection of other links.
2. $l \notin OPT$ and $\Gamma(l) \cap (OPT \setminus \mathcal{C}) = \{l_1\}$: Link l is not in the optimal solution, and adding l to the result $\mathcal{C}$ will occupy the degree space and make the optimal link $l_1 \in OPT$ (incident to either u or v) fail to be selected. Meanwhile, since l is the link with the highest score at selection, if l_1 is not selected ahead of l, it is easy to show that $\hat{y}_l > \hat{y}_{l_1} > \frac{1}{2}\hat{y}_{l_1}$.
3. $l \notin OPT$ and $\Gamma(l) \cap (OPT \setminus \mathcal{C}) = \{l_1, l_2\}$: Link l is not in the optimal solution, and adding of $l = (u, v)$ will occupy the degrees of nodes u and v by 1 and make links $l_1, l_2 \in OPT$ incident to u and v, respectively to be removed. Since l has the highest score, if links l_1 and l_2 are not selected ahead of l, it is easy to show that $\hat{y}_l > \hat{y}_{l_1}$ and $\hat{y}_l > \hat{y}_{l_2}$. Therefore, we have $\hat{y}_l > \frac{1}{2}(\hat{y}_{l_1} + \hat{y}_{l_2})$.

Based on the above remarks, for all the selected links in $\mathcal{C}$, we have

$$\begin{aligned}
\hat{y}(\mathcal{C}) &= \hat{y}\left((\mathcal{C} \cap OPT) \cup (\mathcal{C} \setminus OPT)\right) \\
&= \hat{y}(\mathcal{C} \cap OPT) + \hat{y}(\mathcal{C} \setminus OPT) \\
&= \hat{y}(\mathcal{C} \cap OPT) + \sum_{l \in \mathcal{C} \setminus OPT} \hat{y}_l \\
&> \frac{1}{2}\hat{y}(OPT \cap \mathcal{C}) + \frac{1}{2} \sum_{l \in OPT \setminus \mathcal{C}} \hat{y}_l \\
&= \frac{1}{2}\hat{y}(OPT).
\end{aligned} \tag{4.34}$$

where $\hat{y}(\mathcal{C}) = \sum_{l \in \mathcal{C}} \hat{y}_l$ denotes the score sum of the links in $\mathcal{C}$.

Therefore, the greedy anchor link selection algorithm can achieve a $\frac{1}{2}$ approximation of the optimal solution for the **CLS** problem with $M{:}N$ link *cardinality constraint*, and the time complexity of the greedy method is $O(|\tilde{\mathcal{L}}|)$.

4.5.5 Distributed Algorithm

Meanwhile, for large-scale networks involving billions of nodes and links, the complete network data can hardly be stored in one single machine and the learning framework may suffer from the high computing cost problem a lot. In this section, we will introduce a scalable version of the ITERCLIPS model introduced before based on distributed computational platforms proposed in [43]. The framework involves two iterative steps actually. In the first step, it updates vector $\mathbf{w}$ to calculate the confidence vector $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} = c\mathbf{X}(\mathbf{I} + c\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$, where matrix $c\mathbf{X}(\mathbf{I} + c\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ can actually be pre-computed and divided into blocks to be stored in different slaves (i.e., worker nodes in a cluster). For instance, in Spark, the matrix can be divided into rows, where each row can be saved as an RDD (resilient distributed dataset) in one slave, and each entry in vector $\hat{\mathbf{y}}$ can be updated independently in different slaves simultaneously. The updated values in $\hat{\mathbf{y}}$ can be exchanged among the slaves with very low communication costs. Meanwhile, for the second step in framework, how to generalize the greedy method to the distributed version is not very straightforward, which will be the focus in the following part of this subsection.

According to Theorem 4.1, the *k-maximum weighted matching problem* can be reduced to the objective function of k-**CLS** with general $M{:}N$ cardinality constraint in polynomial time. Therefore, next we will talk about the distributed version of the greedy method for the **CLS** with 1:1 constraint specifically (which can be applied for the general $M{:}N$ cardinality constraint as well). Before diving into details of the distributed greedy algorithm, we first provide some intuitive ideas about how the distributed method works. In the distributed weighted link selection, each process representing one node in the graph knows its neighbor nodes as well as their inferred confidence scores $\hat{y}$. These processes can also communicate with each other by sending and receiving messages. Via the communication among processes, links with locally highest confidence scores can be identified concurrently. Intuitively, by running the algorithm on all the nodes simultaneously, the same matching result can be obtained as the greedy algorithm based on the stand-alone mode.

Algorithm 5 Distributed Greedy algorithm with 1:1 cardinality constraint

```
Require: node $u$, neighbor set $\Gamma(u)$
 1: Initialize neighborhood set $N = \Gamma(u)$
 2: Initialize matching candidate set $C = \emptyset$
 3: Select candidate $c = candidate(u, N)$
 4: if $c \neq null$ then
 5:     Send $\langle invite \rangle$ message to $c$
 6: end if
 7: while $N \neq \emptyset$ do
 8:     Receive a message $m$ from neighbor $v$
 9:     if $m == \langle invite \rangle$ then
10:         $C = C \cup \{v\}$
11:     end if
12:     if $m == \langle remove \rangle$ then
13:         $N = N \setminus \{v\}$
14:         $C = C \setminus \{v\}$
15:         if $v == c$ then
16:             Select new candidate $c = candidate(u, N)$
17:             if $c \neq null$ then
18:                 Send $\langle invite \rangle$ message to $c$
19:             end if
20:         end if
21:     end if
22:     if $c \neq null \wedge c \in C$ then
23:         for $w \in C \setminus \{c\}$ do
24:             Send $\langle remove \rangle$ message to $w$
25:         end for
26:         $C = \emptyset$
27:     end if
28: end while
```

Formally, the pseudo-code of the distributed greedy algorithm is available in Algorithm 5. According to the algorithm, for each node u, its neighbor set N can be initialized as $\Gamma(u)$ (N will change dynamically in the algorithm). Function $c = candidate(u, N)$ returns the candidate of u, whose link with u is of the highest confidence score, i.e.,

$$c = candidate(u, N) = \arg_{v \in N} \max \hat{y}_{(u,v)}. \tag{4.35}$$

Initially, node u will send the *invite* message to the candidate c, and receive messages from all the neighbors in set N. If the message received from neighbor v is also an *invite* (i.e., u is the most promising candidate of v), u will add v to its *matching candidate set* C. Meanwhile, if the message is *remove*, it denotes v has already found its partner and link between them has already been selected. Node v will be removed from u's *neighbor set* and *matching candidate set*. What's more, if v happens to be candidate c, node u will retrieve the next most promising candidate c and send a new *invite* message again. Finally, if candidate c invites u and u also invites c, the link between whom will be of the highest score and selected finally.

Lemma 4.1 *In the distributed greedy algorithm, each process (node) sends out at most one message over each incident edge.*

Proof In the algorithm, for each node u, it will send an *invite* message to the first candidate as well as other candidate if the previous candidates send a *remove* message to u. Therefore, for each potential candidate c obtained via function $candidate(u, N)$, u sends at most one *invite* message to c.

Meanwhile, the *remove* messages are merely sent to the other neighbors in N (excluding candidates c) only once. In other words, all the neighbors in $\Gamma(u)$ only receive exactly one message (either *invite* or *remove*) from u in the whole process.

According to the analysis in Lemma 4.1, we can prove the time complexity of the distributed greedy Algorithm to be $O(|\tilde{\mathcal{L}}|)$.

4.6 Summary

In this chapter, we focused on the supervised network alignment problem. Based on a set of labeled anchor links, the supervised network alignment problem aims at learning a mapping to infer the potential labels of the unlabeled anchor links. To address such a problem, three different supervised network alignment approaches have been introduced in this chapter, including the full network alignment method MNA, partial network alignment method PNA, and the general network alignment method ITERCLIPS with different cardinality constraints.

Based on a detailed data analysis, we illustrated that the heterogeneous information available in the online social networks can be utilized to extract some useful features for the anchor links across networks, which include the social connections, textual contents and spatial check-ins and temporal activity distribution. Based on these features, we introduced the MNA model, which covers two phases: (1) anchor link classification, and (2) social network matching for anchor link pruning. In MNA, the studied social networks are assumed to be fully aligned, i.e., all the involved users will be connected by an anchor link, and the anchor links are assumed to be subject to the one-to-one cardinality constraint.

Instead of studying the network alignment problem based on two specific online social network, e.g., Foursquare and Twitter, we introduced another general network alignment method PNA. Based on the heterogeneous information across the social networks, PNA introduces a general feature extraction approach based on meta path and tensor decomposition. The social networks to be aligned by PNA are partially aligned instead, where a bunch of the users are the non-anchor users. In other words, the anchor links are subject to the one-to-one $_\leq$ cardinality constraint instead. To pruning the non-existing anchor links across the social networks, PNA adopts a generic stable matching algorithm to extract the final mapping of users across networks.

Finally, in the last section of this chapter, we generalized the cardinality constraint on anchor links, and introduced the network alignment approach ITERCLIPS. Model ITERCLIPS can handle the network alignment problem very well, where the cardinality constraint on anchor links can be either *one-to-one*, *one-to-many*, or *many-to-many*, respectively. ITERCLIPS models the cardinality constraint on anchor links as the mathematical constraint on node degrees instead, and addresses the network alignment problem as an optimization problem. ITERCLIPS adopts a greedy search approach to pick the anchor links across networks, which can achieve a $\frac{1}{2}$-approximation of the optimal solution.

4.7 Bibliography Notes

In recent years, witnessing the rapid growth of online social networks, researchers start to shift their attention to align multiple online social networks. Homogeneous network alignment was studied in [24], enlightened by which the problem of aligning two bipartite networks is studied by Koutra [13], where a fast alignment algorithm which can be applied to large-scale networks is introduced. Users can have various types of attribute information in social networks generated by their social activities, based on which Zafarani et al. study the cross-network user matching problem in [30]. These proposed approaches are mostly proposed based on some simple heuristics and assumptions.

The supervised network alignment problem initially proposed in [12] has become one of the most important research problems in social network studies. By extending the traditional supervised link prediction approach [1] to the inter-network scenario, Kong et al. [12] introduces a two-phase approach to address the network alignment via anchor link prediction and network matching. A set of useful inter-network features extracted for anchor links have also been provided in Kong et al. [12] as well, which can capture very useful signals about the anchor links.

Supervised partial network alignment is introduced in [38], which allows users to stay isolated without connections to anchor links in the network alignment process. A detailed description about the tensor and related tensor decomposition approaches is available in [11]. Class imbalance is a serious problem in traditional machine learning, and the frequently used techniques proposed to handle such a problem include both *down sampling* [14] and *over sampling* [4]. The readers may refer to these papers for more information when reading Sect. 4.4.

Graph matching has been an important research problem in graph studies for a very long time, and Jack Edmonds introduced an efficient algorithm to address the maximum matching problem based on graphs in [7]. As proposed in the existing works [9], the *k-maximum weighted matching problem* is actually NP-hard. The network alignment approach ITERCLIPS with the general cardinality constraint was initially introduced in [43], which unifies the prediction tasks of links subject to different cardinality constraints into the one framework.

4.8 Exercises

1. (Easy) Given two partially aligned networks shown in Fig. 4.12, please compute the *extended common neighbor* between user pair $C^{(1)}$ and $C^{(2)}$ across the networks, respectively.
2. (Easy) Given two partially aligned networks shown in Fig. 4.12, please compute the *extended Jaccard's Coefficient* between user pair $C^{(1)}$ and $C^{(2)}$ across the networks, respectively.
3. (Easy) Given two partially aligned networks shown in Fig. 4.12, please compute the *extended Adamic/Adar Index* between user pair $C^{(1)}$ and $C^{(2)}$ across the networks, respectively.
4. (Easy) Please compute the number of meta path instances connecting B_F and D_T across the networks shown in Fig. 4.12 ($G^{(1)}$: Foursquare, $G^{(2)}$: Twitter) based on meta path $U^{(1)} \rightarrow U^{(1)} \rightarrow U^{(1)} \leftrightarrow U^{(2)} \leftarrow U^{(2)}$.
5. (Easy) Please compute the number of meta path instances connecting B_F and E_T across the networks shown in Fig. 4.12 ($G^{(1)}$: Foursquare, $G^{(2)}$: Twitter) based on meta path $U^{(1)} \rightarrow U^{(1)} \rightarrow U^{(1)} \leftrightarrow U^{(2)} \leftarrow U^{(2)} \leftarrow U^{(2)}$.
6. (Medium) Please identify the *stable matching* of the networks as shown in Fig. 4.13 with Algorithm 1.
7. (Medium) Please identify the *generic stable matching* of the networks as shown in Fig. 4.13 with Algorithm 2 (with $K = 1$).
8. (Medium) Please identify the *greedy matching* of the networks subject to *one-to-one* cardinality constraint as shown in Fig. 4.13 with the *greedy link selection* method introduced in Algorithm 3.
9. (Hard) Please try to implement the *stable matching* algorithm (i.e., Algorithm 1) and the *generic stable matching* algorithm (i.e., Algorithm 2) in a programming language you prefer. You can also input the aligned network structures shown in Fig. 4.13 as the input to test the correctness of your implementation.
10. (Hard) Please try to implement the *greedy link selection* algorithm (i.e., Algorithm 3) and use the networks in Fig. 4.13 to text the implementation correctness.

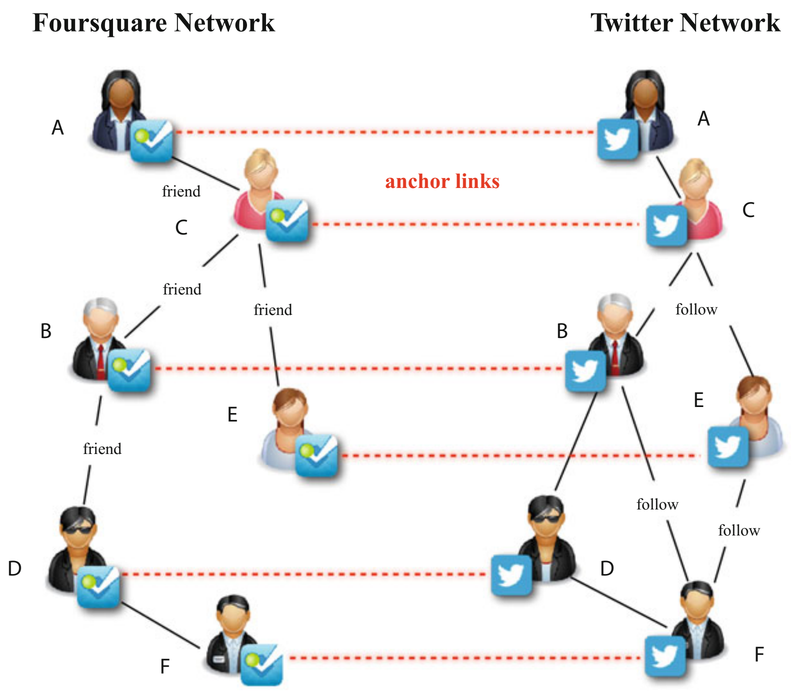


Fig. 4.12 Multiple aligned social networks input

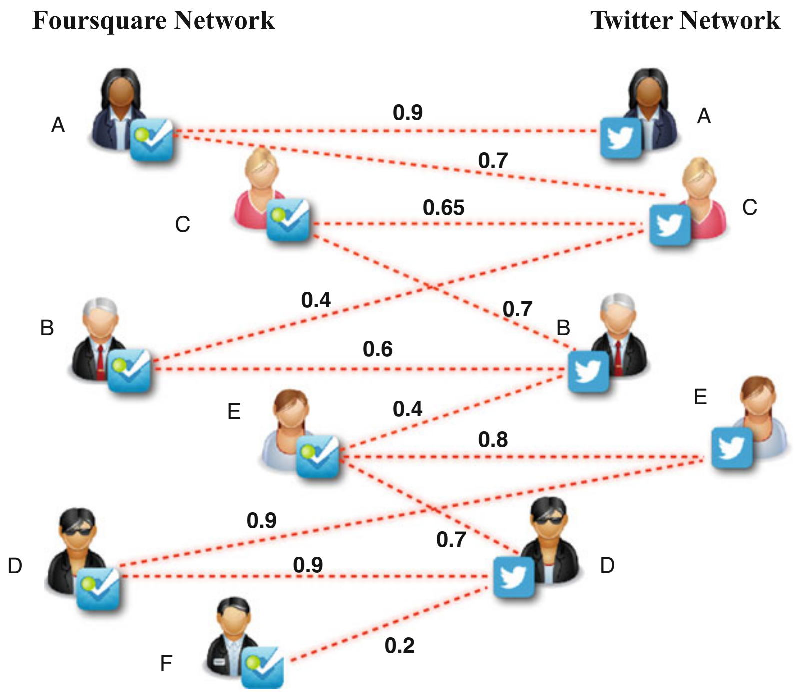


Fig. 4.13 Aligned social networks with inference confidence scores

References

1. M. Al Hasan, V. Chaoji, S. Salem, M. Zaki, Link prediction using supervised learning, in *Workshop on Link Analysis, Counterterrorism and Security (SDM 06)* (2006)
2. M. Bayati, M. Gerritsen, D. Gleich, A. Saberi, Y. Wang, Algorithms for large, sparse network alignment problems, in *2009 Ninth IEEE International Conference on Data Mining* (2009)
3. I. Bhattacharya, L. Getoor, Collective entity resolution in relational data. ACM Trans. Knowl. Discov. Data **1**(1), 5 (2007)
4. N. Chawla, K. Bowyer, L. Hall, P. Kegelmeyer, Smote: synthetic minority over-sampling technique. J. Artif. Int. Res. **16**, 321–357 (2002)
5. L. Dubins, D. Freedman, Machiavelli and the Gale-Shapley algorithm. Am. Math. Mon. **88**(7), 485–494 (1981)
6. M. Duggan, A. Smith, Social media update 2013 (2013). Report available at http://www.pewinternet.org/2013/12/30/social-media-update-2013/
7. J. Edmonds, Maximum matching and a polyhedron with 0, 1 vertices. J. Res. Natl. Bur. Stand. **69**(125–130), 55–56 (1965)
8. J. Euzenat, P. Shvaiko, *Ontology Matching* (Springer, Secaucus, 2007)
9. M. Garey, D. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness* (W. H. Freeman & Co., New York, 1990)
10. D. Kempe, J. Kleinberg, É. Tardos, Maximizing the spread of influence through a social network, in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)* (2003)
11. T. Kolda, B. Bader, Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
12. X. Kong, J. Zhang, P. Yu, Inferring anchor links across multiple heterogeneous social networks, in *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (CIKM '13)* (2013)
13. D. Koutra, H. Tong, D. Lubensky, Big-align: fast bipartite graph alignment, in *2013 IEEE 13th International Conference on Data Mining* (2013)
14. M. Kubat, S. Matwin, Addressing the curse of imbalanced training sets: one-sided selection, in *Proceedings of the Fourteenth International Conference on Machine Learning* (1997)
15. R. Lichtenwalter, J. Lussier, N. Chawla, New perspectives and methods in link prediction, in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '10)* (2010)
16. MarketingCharts, Majority of twitter users also use Instagram (2014). Report available at http://www.marketingcharts.com/wp/online/majority-of-twitter-users-also-use-instagram-38941/
17. A. Menon, C. Elkan, Link prediction via matrix factorization, in *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2011)* (2011)
18. S. Moghaddam, M. Jamali, M. Ester, ETF: extended tensor factorization model for personalizing prediction of review helpfulness, in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)* (2012)
19. O. Peled, M. Fire, L. Rokach, Y. Elovici, Entity matching in online social networks, in *2013 International Conference on Social Computing* (2013)
20. Y. Sun, R. Barber, M. Gupta, C. Aggarwal, J. Han, Co-author relationship prediction in heterogeneous bibliographic networks, in *2011 International Conference on Advances in Social Networks Analysis and Mining* (2011)
21. Y. Sun, J. Han, C. Aggarwal, N. Chawla, When will it happen?: relationship prediction in heterogeneous information networks, in *Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM 2012)* (2012)
22. J. Tang, H. Gao, X. Hu, H. Liu, Exploiting homophily effect for trust prediction, in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13)* (2013)
23. I. Tomek, Two modifications of CNN, IEEE Trans. Syst. Man Cybern. **6**, 769–772 (1976)
24. S. Umeyama, An eigendecomposition approach to weighted graph matching problems, IEEE Trans. Pattern Anal. Mach. Intell. **10**(5), 695–703 (1988)
25. Z. Wang, J. Liao, Q. Cao, H. Qi, Z. Wang, Friendbook: a semantic-based friend recommendation system for social networks. IEEE Trans. Mob. Comput. **14**(3), 538–551 (2015)
26. X. Xie, Potential friend recommendation in online social network, in *2010 IEEE/ACM International Conference on Green Computing and Communications International Conference on Cyber, Physical and Social Computing* (2010)
27. J. Yang, J. McAuley, J. Leskovec, Community detection in networks with node attributes, CoRR, abs/1401.7267 (2014)
28. J. Ye, H. Cheng, Z. Zhu, M. Chen, Predicting positive and negative links in signed social networks by transfer learning, in *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)* (2013)

29. B. Zadrozny, C. Elkan, Transforming classifier scores into accurate multiclass probability estimates, in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)* (2002)
30. R. Zafarani, H. Liu, Connecting users across social media sites: a behavioral-modeling approach, in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)* (2013)
31. J. Zhang, Social network fusion and mining: a survey (2018). arXiv preprint. arXiv:1804.09874
32. J. Zhang, P. Yu, Community detection for emerging networks, in *Proceedings of the 2015 SIAM International Conference on Data Mining* (2015)
33. J. Zhang, P. Yu, Multiple anonymized social networks alignment, in *2015 IEEE International Conference on Data Mining* (2015)
34. J. Zhang, P. Yu, PCT: partial co-alignment of social networks, in *Proceedings of the 25th International Conference on World Wide Web (WWW '16)* (2016)
35. J. Zhang, X. Kong, P. Yu, Predicting social links for new users across aligned heterogeneous social networks, in *2013 IEEE 13th International Conference on Data Mining* (2013)
36. J. Zhang, X. Kong, P. Yu, Transferring heterogeneous links across location-based social networks, in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM '14)* (2014)
37. J. Zhang, P. Yu, Z. Zhou, Meta-path based multi-network collective link prediction, in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)* (2014)
38. J. Zhang, W. Shao, S. Wang, X. Kong, P. Yu, Pna: partial network alignment with generic stable matching, in *2015 IEEE International Conference on Information Reuse and Integration* (2015)
39. J. Zhang, S. Wang, Q. Zhan, P. Yu, Intertwined viral marketing in social networks, in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (2016)
40. J. Zhang, P. Yu, Y. Lv, Q. Zhan, Information diffusion at workplace, in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)* (2016)
41. J. Zhang, Q. Zhan, P. Yu, Concurrent alignment of multiple anonymized social networks with generic stable matching, in *Information Reuse and Integration* (2016)
42. J. Zhang, C. Aggarwal, P. Yu, Rumor initiator detection in infected signed networks, in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017)
43. J. Zhang, J. Chen, J. Zhu, Y. Chang, P. Yu, Link prediction with cardinality constraints, in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)* (2017)