# Chapter 1

# Collective Classification of Network Data

## 1.1 Introduction

Network data has become ubiquitous. Communication networks, social networks and the World Wide Web are becoming increasingly important to our day-to-day life. Moreover, networks can be defined implicitly by certain structured data sources, such as images and text. We are often interested in inferring hidden attributes (i.e., labels) about network data, such as whether a Facebook user will adopt a product, or whether a pixel in an image is part of the foreground, background or some specific object. Intuitively, the network should help guide this process. For instance, observations and inference about someone's Facebook friends should play a role in determining their adoption probability. This type of joint reasoning about label correlations in network data is often referred to as *collective classification.*

Classic machine learning literature tends to study the supervised setting, in which a classifier is learned from a fully-labeled *training set*; classification performance is measured by some form of statistical accuracy, which is typically estimated from a held-out *test set.* It is commonly assumed that data points (i.e., feature-label pairs) are generated independently and identically from an underlying distribution over the domain, as illustrated in Figure 1.1(a). As a result, classification is performed independently on each object, without taking into account any underlying network between the objects. Classification of network data does not fit well into this setting. Domains such as webpages, citation networks and social networks have naturally occurring relationships between objects. Because of these connections (illustrated in Figure 1.1(b)), their features and labels are likely to be correlated. Neighboring points may be more likely to share the same label (a phenomenon sometimes referred to as *social influence* or *contagion*), or links may be more likely between instances of the same class (referred to as *homophily* or *assortativity*). Models that classify each object independently are ignoring a wealth of information, and may not perform well.

Classifying real network data is further complicated by heterogenous networks, in which nodes may not have uniform local features and degrees (as illustrated in Figure 1.1(c)). Because of this, we cannot assume that nodes are identically distributed. Also, it is likely that there is not a clean split between the training and test sets (as shown in Figure 1.1(d)), which is common in relational datasets. Without independence between training and testing, it may be difficult to isolate training accuracy from testing accuracy, so the statistical properties of the estimated model are not straightforward.

In this article, we provide an overview of existing approaches to collective classification. We begin by formally defining the problem. We then examine several approaches to collective classification: iterative wrappers for local predictors, graph-based regularization and probabilistic graphical models. To help ground these concepts in practice, we review some common feature engineering techniques for real-world problems. Finally, we conclude with some interesting applications of collective classification.

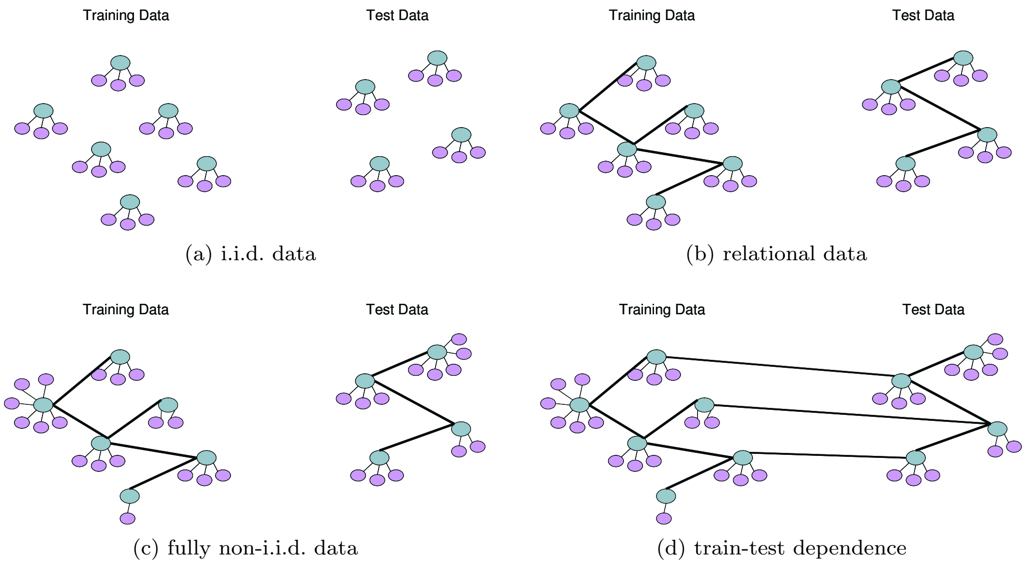


FIGURE 1.1: (a) An illustration of the common i.i.d. supervised learning setting. Here each instance is represented by a subgraph consisting of a label node (blue) and several local feature nodes (purple). (b) The same problem, cast in the relational setting, with links connecting instances in the training and testing sets, respectively. The instances are no longer independent. (c) A relational learning problem in which each node has a varying number of local features and relationships, implying that the nodes are neither independent nor identically distributed. (d) The same problem, with relationships (links) between the training and test set.

## 1.2 Collective Classification Problem Definition

Fix a graph $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$ on $n$ nodes $\mathcal{V} \triangleq \{1, \ldots, n\}$, with edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. For the purposes of this chapter, assume that the structure of the graph is given or implied by an observed network topology. For each node $i$, we associate two random variables: a set of local features $X_i$ and a label $Y_i$, whose (possibly heterogeneous) domains are $\mathcal{X}_i$ and $\mathcal{Y}_i$ respectively. Assume that the local features of the entire network, $\mathbf{X} \triangleq \{X_1, \ldots, X_n\}$, are observed. In some cases, a subset of the labels, $\mathbf{Y} \triangleq \{Y_1, \ldots, Y_n\}$, are observed as well; we denote the labeled and unlabeled subsets by $\mathbf{Y}^\ell \subseteq \mathbf{Y}$ and $\mathbf{Y}^u \subseteq \mathbf{Y}$ respectively, where $\mathbf{Y}^\ell \cap \mathbf{Y}^u = \emptyset$ and $\mathbf{Y}^\ell \cup \mathbf{Y}^u = \mathbf{Y}$. Given $\mathbf{X}$ and $\mathbf{Y}^\ell$, the collective classification task is to infer $\mathbf{Y}^u$.

In general, collective classification is a combinatorial optimization problem. The objective function varies, depending on the choice of model; generally, one minimizes an energy function that depends on parametric assumptions about the generating distribution. Here we describe several common approaches: iterative classification, label propagation and graphical models.

Throughout this document, we employ the following notation. Random variables will be denoted by uppercase letters, e.g., $X$, while realizations of variables will be indicated by lowercase, e.g., $x$. Sets or vectors of random variables (or realizations) will be denoted by bold, e.g., $\mathbf{X}$ (or $\mathbf{x}$). For a node $i$, let $\mathcal{N}_i$ denote the set of indices corresponding to its (open) neighborhood; that is, the set of nodes adjacent to $i$ (but not including it).

### 1.2.1 Inductive vs. Transductive Learning

Learning scenarios for collective classification broadly fall into two main categories: *inductive* and *transductive*. In inductive learning, data is assumed to be drawn from a distribution over the domain; that is, a sentence, image, social network or some other data structure is generated according to a distribution over instances of said structure. Given a number of labeled structures drawn from this distribution, the objective is to learn to predict on new draws. In the transductive setting, the problem domain is fixed, meaning the data simply exists. The distribution from which the data was drawn is therefore irrelevant, since there is no randomness over what values could occur. Instead, randomness comes from which nodes are labeled, which happens via some stochastic sampling process. Given a labeled subset of the data, the goal is to learn to correctly predict the remaining instances.

In the inductive setting, one commonly assumes that draws of test examples are independent of the training examples. However, this may not hold with relational data, since the data-generating process may inject some dependence between draws from the distribution. There may be dependencies between nodes in the train and test data, as illustrated in Figure 1.1. The same is true of the transductive setting, since the training data may just be a labeled portion of one large network. The dependencies between training and testing must be considered when computing certain metrics, such as train and test accuracy [21].

Since collective methods leverage the correlations between adjacent nodes, researchers typically assume that a small subset of labels are given during prediction. In the transductive setting, these node are simply the training set; in the inductive setting, this assumes that draws from the distribution over network structures are partially labeled. However, inductive collective classification is still possible even if no labels are given.

### 1.2.2 Active Collective Classification

An interesting subproblem in collective classification is how one acquires labels for training (or prediction). In supervised and semi-supervised learning, it is commonly assumed that annotations are given *a priori*—either adversarially, as in the online model, or agnostically, as in the *probably approximately correct* (PAC) model. In these settings, the learner has no control over which or data points are labeled.

This motivates the study of *active* learning. In active learning, the learner is given access to an *oracle*, which it can query for the labels of certain examples. In active collective classification, the learning algorithm is allowed to ask for the labels of certain nodes, so as to maximize its performance using the minimal number of labels. How it decides *which* labels to query for is an open problem that is generally NP-hard; nonetheless, researchers have proposed many heuristics that work well in practice [4, 21, 27]. These typically involve some trade-off between propagation of information from an acquired label and coverage of the network.

The label acquisition problem is also relevant during inference, since the predictor might have access to a label oracle at test time. This form of *active inference* has been very successful in collective classification [3, 35]. Queries can sometimes be very sophisticated: a query might return not only a node's label, but also the labels of its neighbors; if the data graph is uncertain, a query might also return information about a node's local relationships (e.g., friends, citations, influencers, etc.). This has been referred to as *active surveying* [30, 38].

Note that an optimal label acquisition strategy for active learning may not be optimal for active inference, so separate strategies may be beneficial. However, the relative performance improvements of active learning and inference are easily conflated, making it difficult to optimize the acquisition strategies. Kuwadekar and Neville [21] propose a relational active learning framework to combat this problem.

## 1.3 Iterative Methods

Collective classification can in some sense be seen as achieving agreement amongst a set of interdependent, local predictions. Viewed as such, some approaches have sought ways to iteratively combine and revise individual node predictions so as to reach an equilibrium. The collective inference algorithm is essentially just a wrapper (or *meta-algorithm*) for a local prediction subroutine. One of the benefits of this technique is that local predictions can be made efficiently, so the compexity of collective inference is effectively the number of iterations needed for convergence. Though neither convergence nor optimality is guaranteed, in practice, this approach typically converges quickly to a good solution, depending on the graph structure and problem complexity. The methods presented in this section are representative of this iterative approach.

### 1.3.1 Label Propagation

A natural assumption in network classification is that adjacent nodes are likely to have the same label, (i.e., contagion). If the graph is weighted, then the edge weights $\{w_{i,j}\}_{(i,j)\in\mathcal{E}}$ can be interpreted as the strength of the associativity. Weights can sometimes be derived from observed features $\mathbf{X}$ via a similarity function. For example, a *radial basis function* can be computed between adjacent nodes $(i, j)$ as

$$w_{i,j} \triangleq \exp\left(-\frac{\|X_i - X_j\|_2^2}{\sigma^2}\right), \tag{1.1}$$

where $\sigma$ is a parameter that determines the width of the Gaussian.

Suppose the labels are binary, with $\mathcal{Y}_i \in \{\pm 1\}$ for all $i = 1, \ldots, n$. If node $i$ is unlabeled, we could predict a score for $Y_i = 1$ (or $Y_i = -1$) as the weighted average of its neighbors' labels, i.e.,

$$Y_i \leftarrow \left(\frac{1}{\sum_{j \in \mathcal{N}_i} w_{i,j}}\right) \sum_{j \in \mathcal{N}_i} w_{i,j} Y_j.$$

One could then clamp $Y_i$ to $\{\pm 1\}$ using its sign, $\operatorname{sgn}(Y_i)$. While we probably will not know all of the labels of $\mathcal{N}_i$, if we already had predictions for them, we could use these, then iterate until the predictions converge. This is precisely the idea behind a method known as *label propagation*. Though the algorithm was originally proposed by Zhu and Ghahramani [48] for general transductive learning, it can easily be applied to network data by constraining the similarities according to a graph. An example of this is the *modified adsorption* algorithm [39].

Algorithm 1 provides pseudocode for a simple implementation of label propagation. The algorithm assumes that all labels are $k$-valued, meaning $|\mathcal{Y}_i| = k$ for all $i = 1, \ldots, n$. It begins by constructing a $n \times k$ label matrix $\mathbf{Y} \in \mathbb{R}^{n \times k}$, where entry $i, j$ corresponds to the probability that $Y_i = j$. The label matrix is initialized as

$$\mathbf{Y}_{i,j} = \begin{cases} 1 & \text{if } Y_i \in \mathbf{Y}^\ell \text{ and } Y_i = j, \\ 0 & \text{if } Y_i \in \mathbf{Y}^\ell \text{ and } Y_i \neq j, \\ 1/k & \text{if } Y_i \in \mathbf{Y}^u. \end{cases} \tag{1.2}$$

It also requires an $n \times n$ *transition matrix* $\mathbf{T} \in \mathbb{R}^{n \times n}$; semantically, this captures the probability that a label propagates from node $i$ to node $j$, but it is effectively just the normalized edge weight, defined as

$$\mathbf{T}_{i,j} = \begin{cases} \frac{w_{i,j}}{\sum_{j \in \mathcal{N}_i} w_{i,j}} & \text{if } j \in \mathcal{N}_i, \\ 0 & \text{if } j \notin \mathcal{N}_i. \end{cases} \tag{1.3}$$

The algorithm iteratively multiplies $\mathbf{Y}$ by $\mathbf{T}$, thereby propagating label probabilities via a weighted average. After the multiply step, the unknown rows of $\mathbf{Y}$, corresponding to the unknown labels, must be normalized, and the known rows must be clamped to their known values. This continues until the values of $\mathbf{Y}$ have stabilized (i.e., converged to within some sufficiently small $\epsilon$ of change), or until a maximum number of iterations has been reached.

One interesting property of this formulation of label propagation is that it is guaranteed to converge to a unique solution. In fact, there is a closed-form solution, which we will describe in Section 1.4.

### 1.3.2 Iterative Classification Algorithms

While label propagation is surprisingly effective, its predictor is essentially just a weighted average of neighboring labels, which may sometimes fail to capture complex relational dynamics. A more sophisticated approach would be to use a richer predictor. Suppose we have a classifier $h$ that has been trained to classify a node $i$, given its features $X_i$ and the features $\mathbf{X}_{\mathcal{N}_i}$ and labels $\mathbf{Y}_{\mathcal{N}_i}$ of its neighbors. *Iterative classification* does just that,

**Algorithm 1** Label propagation

```
1: Initialize $\mathbf{Y}$ per Eq. (1.2)
2: Initialize $\mathbf{T}$ per Eq. (1.3)
3: repeat
4:    $\mathbf{Y} \leftarrow \mathbf{TY}$
5:    Normalize unknown rows of $\mathbf{Y}$
6:    Clamp known rows of $\mathbf{Y}$ using known labels
7: until convergence or maximum iterations reached
8: Assign to $Y_i$ the $j^{\text{th}}$ label, where $j$ is the highest value in row $i$ of $\mathbf{Y}$.
```

applying local classification to each node, conditioned on the current predictions (or ground truth) on its neighbors, and iterating until the local predictions converge to a global solution. Iterative classification is an "algorithmic framework," in that it is it is agnostic to the choice of predictor; this makes it a very versatile tool for collective classification.

Chakrabarti et al. [6] introduced this approach and reported impressive gains in classification accuracy. Neville and Jensen [31] further developed the technique, naming it "iterative classification," and studied the conditions under which it improved classification performance [14]. Researchers [24, 25, 28] have since proposed various improvements and extensions to the basic algorithm we present.

**Algorithm 2** Iterative classification

```
1: for $Y_i \in \mathbf{Y}^u$ do {bootstrapping}
2:    $Y_i \leftarrow h(X_i, \mathbf{X}_{\mathcal{N}_i}, \mathbf{Y}^{\ell}_{\mathcal{N}_i})$
3: end for
4: repeat {update predicted labels}
5:    $\pi \leftarrow$ GenPerm($n$) {generate permutation $\pi$ over $1, \ldots, n$}
6:    for $i = 1, \ldots, n$ do
7:       if $Y_{\pi(i)} \in \mathbf{Y}^u$ then
8:          $Y_{\pi(i)} \leftarrow h(X_{\pi(i)}, \mathbf{X}_{\mathcal{N}_{\pi(i)}}, \mathbf{Y}_{\mathcal{N}_{\pi(i)}})$
9:       end if
10:   end for
11: until convergence or maximum iterations reached
```

Algorithm 2 depicts pseudo-code for a simple iterative classification algorithm. The algorithm begins by initializing all unknown labels $\mathbf{Y}^u$ using only the features $(X_i, \mathbf{X}_{\mathcal{N}_i})$ and *observed* neighbor labels $\mathbf{Y}^{\ell}_{\mathcal{N}_i} \subseteq \mathbf{Y}_{\mathcal{N}_i}$. (This may require a specialized initialization classifier.) This process is sometimes referred to as *bootstrapping.* It then iteratively updates these values using the current predictions as well as the observed features and labels. This process repeats until the predictions have stabilized, or until a maximum number of iterations has been reached.

Clearly, the order in which nodes are updated affects the predictive accuracy and convergence rate, though there is some evidence to suggest that iterative classification is fairly robust to a number of simple ordering strategies—such as random ordering, ascending order of neighborhood diversity and descending order of prediction confidences [11]. Another practical issue is when to incorporate the predicted labels from the previous round into the the current round of prediction. Some researchers [28, 31] have proposed a "cautious" approach, in which only predicted labels are introduced gradually. More specifically, at each iteration, only the top $k$ most confident predicted labels are used, thus ignoring less confident, potentially noisy predictions. At the start of the algorithm, $k$ is initialized to some small number; then, in subsequent iterations, the value of $k$ is increased, so that in the last iteration all predicted labels are used.

One benefit of iterative classification is that it can be used with any local classifier, making it extremely flexible. Nonetheless, there are some practical challenges to incorporating certain classifiers. For instance, many classifiers are defined on a predetermined number of features, making it difficult to accommodate arbitrarily-sized neighborhoods. A common workaround is to aggregate the neighboring features and labels, such as using the proportion of neighbors with a given label, or the most frequently occurring label. For classifiers that return a vector of scores (or conditional probabilities) instead of a label, one typically uses the label that corresponds to the maximum score. Some of the classifiers used included: naïve Bayes [6, 31], logistic regression [24], decision trees [14] and weighted-majority [25].

Iterative classification prescribes a method of inference, but it does not instruct how to train the local classifiers. Typically, this is performed using traditional, non-collective training.

## 1.4 Graph-based Regularization

When viewed as a transductive learning problem, the goal of collective classification is to complete the labeling of a partially-labeled graph. Since the problem domain is fixed (that is, the data to be classified is known), there is no need to learn an inductive model[1]; simply the predictions for the unknown labels will suffice. A broad category of learning algorithms, known as *graph-based regularization* techniques, are designed for this type of *model-free* prediction. In this section, we review these methods.

For the remainder of this section, we will employ the following notation. Let $\mathbf{y} \in \mathbb{R}^n$ denote a vector of labels corresponding to the nodes of the graph. For the methods considered in this section, we assume that the labels are binary; thus, if the $i^{\text{th}}$ label is known, then $y_i \in \{\pm 1\}$, and otherwise, $y_i = 0$. The learning objective is to produce a vector $\mathbf{h} \in \mathbb{R}^n$ of predictions that minimizes the L2 distance to $\mathbf{y}$ for the known labels. We can formulate this as a weighted inner product using a diagonal matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$, where the $(i, i)$ entry is set to 1 if the $i^{\text{th}}$ label is observed and 0 otherwise.[2] The error can thus be expressed as

$$(\mathbf{h} - \mathbf{y})^\top \mathbf{C} (\mathbf{h} - \mathbf{y}).$$

Unconstrained graph-based regularization methods can be generalized using the following abstraction (due to Cortes et al. [8]). Let $\mathbf{Q} \in \mathbb{R}^{n \times n}$ denote a symmetric matrix, whose entries are determined based on the structure of the graph $\mathcal{G}$, the local attributes $\mathbf{X}$ (if available) and the observed labels $\mathbf{Y}^\ell$. We will give several explicit definitions for $\mathbf{Q}$ shortly; for the time being, it will suffice to think of $\mathbf{Q}$ as a *regularizer* on $\mathbf{h}$. Formulated as an unconstrained optimization, the learning objective is

$$\arg\min_{\mathbf{h}} \mathbf{h}^\top \mathbf{Q} \mathbf{h} + (\mathbf{h} - \mathbf{y})^\top \mathbf{C} (\mathbf{h} - \mathbf{y}).$$

One can interpret the first term as penalizing certain label assignments, based on observed

[1]This is not to say that inductive models are not useful in the transductive setting. Indeed, many practitioners apply model-based approaches to transductive problems [37].

[2]One could also apply different weights to certain nodes; or, if $\mathbf{C}$ were not diagonal, one could weight errors on certain combinations of nodes differently.

information; the second term is simply the prediction error with respect to the training labels. Using vector calculus, we obtain a closed-form solution to this optimization as

$$\mathbf{h}^\star = (\mathbf{C}^{-1}\mathbf{Q} + \mathbf{I})^{-1}\mathbf{y} = (\mathbf{Q} + \mathbf{C})^{-1}\mathbf{C}\mathbf{y}, \tag{1.4}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix. This is fairly efficient to compute for moderate-sized networks; the time complexity is dominated by $\mathrm{O}(n^3)$ operations for the matrix inversion and multiplication. For prediction, the "soft" values of $\mathbf{h}$ can be clamped to $\{\pm 1\}$ using the sign operator.

The effectiveness of this generic approach comes down to how one defines the regularizer, $\mathbf{Q}$. One of the first instances is due to Zhu et al. [49]. In this formulation, $\mathbf{Q}$ is a *graph Laplacian*, constructed as follows: for each edge $(i, j) \in \mathcal{E}$, define a weight matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$, where each element $w_{i,j}$ is defined using the radial basis function in (1.1); define a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ as

$$d_{i,i} \triangleq \sum_{j=1}^{n} w_{i,j};$$

one then computes the regularizer as

$$\mathbf{Q} \triangleq \mathbf{D} - \mathbf{W}.$$

One could alternately define the regularizer as a *normalized* Laplacian,

$$\mathbf{Q} \triangleq \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}},$$

per Zhou et al. [47]. Ji et al. [15] extended this method for heterogeneous networks—that is, graphs with multiple types of nodes and edges. Another variant, due to Wu and Schölkopf [43], sets

$$\mathbf{Q} \triangleq (\mathbf{I} - \mathbf{A})^\top (\mathbf{I} - \mathbf{A}),$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a row-normalized matrix capturing the local pairwise similarities. All of these formulations impose a *smoothness* constraint on the predictions, that "similar" nodes—where similarity can be defined by the Gaussian in (1.1) or some other kernel—should be assigned the same label.

There is an interesting connection between graph-based regularization and label propagation. Under the various parameterizations of $\mathbf{Q}$, one can show that (1.4) provides a closed-form solution to the label propagation algorithm in Section 1.3.1 [48]. This means that one can compute certain formulations of label propagation without directly computing the iterative algorithm. Heavily optimized linear algebra solvers can be used to compute (1.4) quickly. Another appealing aspect of these methods is their strong theoretical guarantees [8].

## 1.5 Probabilistic Graphical Models

Graphical models are powerful tools for joint, probabilistic inference, making them ideal for collective classification. They are characterized by a graphical representation of a probability distribution $P$, in which random variables are nodes in a graph $\mathcal{G}$. Graphical models can be broadly categorized by whether the underlying graph is directed (e.g., Bayesian networks or collections of local classifiers) or undirected (e.g., Markov random fields). In this section, we discuss both kinds.

Collective classification in graphical models involves finding the assignment $\mathbf{y}^u$ that maximized the conditional likelihood of $\mathbf{Y}^u$, given evidence $(\mathbf{X} = \mathbf{x}, \mathbf{Y}^\ell = \mathbf{y}^\ell)$; i.e.,

$$\arg\max_{\mathbf{y}} P(\mathbf{Y}^u = \mathbf{y}^u \,|\, \mathbf{X} = \mathbf{x}, \mathbf{Y}^\ell = \mathbf{y}^\ell), \tag{1.5}$$

where $\mathbf{y} = (\mathbf{y}^\ell, \mathbf{y}^u)$. This type of inference—known alternately as *maximum a posteriori* (MAP) or *most likely explanation* (MPE)—is known to be NP-hard in general graphical models, though there certain exceptions in which it can be computed efficiently, and many approximation algorithms that perform well in most settings. We will review selected inference algorithms where applicable.

### 1.5.1 Directed Models

The fundamental directed graphical model is a *Bayesian network* (also called *Bayes net*, or BN).

**Definition [Bayesian Network].** A *Bayesian network* consists of a set of random variables $\mathbf{Z} \triangleq \{Z_1, \ldots, Z_n\}$ a directed, acyclic graph (DAG) $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$, and a set of *conditional probability distributions* (CPDs), $\{P(Z_i \,|\, \mathbf{Z}_{\mathcal{P}_i})\}_{i=1}^n$, where $\mathcal{P}_i$ denotes the indices corresponding to the causal parents of $Z_i$. When multiplied, the CPDs describe the joint distribution of $\mathbf{Z}$; i.e., $P(\mathbf{Z}) = \prod_i P(Z_i \,|\, \mathbf{Z}_{\mathcal{P}_i})$.

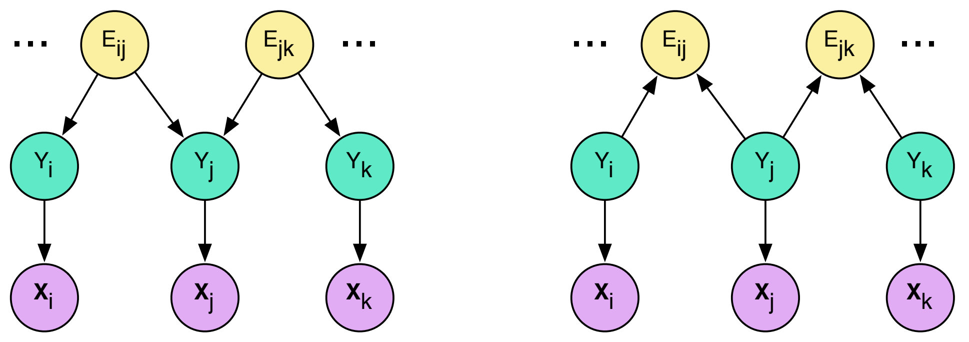
BNs model causal relationships, which are captured by the directionalities of the edges; an edge $(i, j) \in \mathcal{E}$ indicates that $Z_i$ influences $Z_j$. For a more thorough review of BNs, see [18] or Chapter X of this book.

Though BNs are very popular in machine learning and data mining, they can only be used for models with fixed structure, making them inadequate for problems with arbitrary relational structure. Since collective classification is often applied to arbitrary data graphs—such as those found in social and citation networks—some notion of *templating* is required. In short, templating defines subgraph patterns that are *instantiated* (or, *grounded*) by the data graph; model parameters (CPDs) are thus *tied* across different instantiations. This allows directed graphical models to be used on complex relational structures.

One example of a templated model is *probabilistic relational models* (PRMs) [9, 12]. A PRM is a directed graphical model defined by a relational database schema. Given an input database, the schema is instantiated by the database records, thus creating a BN. This has been shown to work for some collective classification problems [12, 13], and has the advantage that a full joint distribution is defined.

To satisfy the requirement of acyclicity when the underlying data graph is undirected, one constructs a (templated) BN or PRM as follows. For each potential edge $\{i, j\}$ in data graph, define a binary random variable $E_{i,j}$. Assume that a node's features are determined by its label. If we further assume that its label is determined by its neighbors' labels (i.e., contagion), then we draw a directed edge from each $E_{i,j}$ to the corresponding $Y_i$ and $Y_j$, as illustrated in Figure 1.2a. On the other hand, if we believe that a node's label determines who it connects to (i.e., homophily), then we draw an edge from each $Y_i$ to all $E_{i,\cdot}$, as shown in Figure 1.2b. The direction of causality is a modeling decision, which depends on one's prior belief about the problem. Note that, in both cases, the resulting graphical model is acyclic.

Another class of templated, directed graphical models is *relational dependency networks* (RDNs) [32]. RDNs have the advantage of supporting graph cycles, though this comes at the cost of *consistency*; that is, the product of an RDN's CPDs does not necessarily define a valid probability distribution. RDN inference is therefore only approximate, but can be



(a) links determine labels (b) labels determine links

FIGURE 1.2: Example BN for collective classification. Label nodes (green) determine features (purple), which are represented by a single vector-valued variable. An edge variable (yellow) is defined for all potential edges in the data graph. In (a), labels are determined by link structure, representing contagion. In (b), links are functions of labels, representing homophily. Both structures are acyclic.

very fast. Learning RDNs is also fast, because it reduces to independently learning a set of CPDs.

### 1.5.2 Undirected Models

While directed graphical models are useful for their representation of causality, sometimes we do not need (or want) to explicitly define causality; sometimes we only know the interactions between random variables. This is where undirected graphical models are useful. Moreover, undirected models are strictly more general than directed models; any directed model can be represented by an undirected model, but there are distributions induced by undirected models that cannot be reproduced in directed models. Specifically, graph structures involving cycles are representable in undirected models, but not in directed models.

Most undirected graphical models fall under the umbrella of Markov random fields (MRFs), sometimes called Markov networks [40].

**Definition [Markov random field].** A *Markov random field* (MRF) is defined by a set of random variables $\mathbf{Z} \triangleq \{Z_1, \ldots, Z_n\}$, a graph $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$ and a set of clique potentials $\{\phi_c : \text{dom}(c) \rightarrow \mathbb{R}\}_{c \in \mathcal{C}}$, where $\mathcal{C}$ is a set of predefined cliques and $\text{dom}(c)$ is the domain of clique $c$. (To simplify notation, assume that potential $\phi_c$ only operates on the set of variables contained in clique $c$.) The potentials are often defined as a log-linear combination of features $f_c$ and weights $w_c$, such that $\phi_c(\mathbf{z}) \triangleq \exp\left(w_c \cdot f_c(\mathbf{z})\right)$. An MRF defines a probability distribution $P$ that factorizes as

$$P(\mathbf{Z} = \mathbf{z}) = \frac{1}{\Phi} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{z}) = \frac{1}{\Phi} \exp\left(\sum_{c \in \mathcal{C}} w_c \cdot f_c(\mathbf{z})\right),$$

where $\Phi \triangleq \sum_{\mathbf{z}'} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{z}')$ is a normalizing constant. This model is said to be *Markovian* because any variable $Z_i$ is independent of any non-adjacent variables, conditioned its neighborhood $\mathbf{Z}_{\mathcal{N}_i}$ (sometimes referred to as its *Markov blanket*).

For collective classification, one can define a *conditional* MRF (sometimes called a CRF), whose conditional distribution is

$$P(\mathbf{Y}^u = \mathbf{y}^u \,|\, \mathbf{X} = \mathbf{x}, \mathbf{Y}^\ell = \mathbf{y}^\ell) = \frac{1}{\Phi} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}, \mathbf{y}) = \frac{1}{\Phi} \exp\left( \sum_{c \in \mathcal{C}} w_c \cdot f_c(\mathbf{x}, \mathbf{y}) \right).$$

For relational tasks, such as collective classification, one typically defines the cliques via templating. Similar to a PRM (see Section 1.5.1), a clique template is just a subgraph pattern—although in this case it is a fully-connected, undirected subgraph. The types of templates used directly affects model complexity, in that smaller templates correspond to a simpler model, which usually generalizes better to unseen data. Thus, MRFs are commonly defined using low-order templates, such as singletons, pairs and sometimes triangles. Examples of templated MRFs include *relational MRFs* [40], *Markov logic networks* [36] and *hinge-loss Markov random fields* [2].

To make this concrete, we consider the class of *pairwise* MRFs. A pairwise MRF has features and weights for all singleton and pairwise cliques in the graph; thus, its distribution factorizes as

$$\begin{aligned} P(\mathbf{Z} = \mathbf{z}) &= \frac{1}{\Phi} \left( \prod_{i \in \mathcal{V}} \phi_i(\mathbf{z}) \right) \left( \prod_{\{i,j\} \in \mathcal{E}} \phi_{i,j}(\mathbf{z}) \right) \\ &= \frac{1}{\Phi} \exp\left[ \left( \sum_{i \in \mathcal{V}} w_i \cdot f_i(\mathbf{z}) \right) + \left( \sum_{\{i,j\} \in \mathcal{E}} w_{i,j} \cdot f_{i,j}(\mathbf{z}) \right) \right]. \end{aligned}$$

(Since it is straightforward to derive the posterior distribution for collective classification, we omit it here.) If we assume that the domains of the variables are discrete, then it is common to define the features as basis vectors indicating the state of the assignment. For example, if $|\mathcal{Z}_i| = k$ for all $i$, then $f_i(\mathbf{z})$ is a length-$k$ binary vector, whose $j^{\text{th}}$ entry is equal to one if $z_i$ is in the $j^{\text{th}}$ state and zero otherwise; similarly, $f_{i,j}(\mathbf{z})$ has length $k^2$ and the only nonzero entry corresponds to the joint state of $(z_i, z_j)$. To make this MRF templated, we simply replace all $w_i$ with a single $w_{\text{single}}$, and all $w_{i,j}$ with a single $w_{\text{pair}}$.

It is important to note that the data graph does not necessarily correspond to the graph of the MRF; there is some freedom in how one defines the relational features $\{f_{i,j}\}_{\{i,j\} \in \mathcal{E}}$. However, when using a pairwise MRF for collective classification, it is natural to define a relational feature for each edge in the data graph. Defining $f_{i,j}$ as a function of $(Y_i, Y_j)$ models the dependence between labels. One may alternately model $(X_i, Y_i, X_j, Y_j)$ to capture the pairwise interactions of both features and labels.

### 1.5.3 Approximate Inference in Graphical Models

Exact inference in general graphical models is intractable, depending primarily on the structure of the underlying graph. Specifically, inference is exponential in the graph's *treewidth*. For structures with low treewidth—such as chains and trees—exact inference can be relatively fast. Unfortunately, these tractable settings are rare in collective classification, so inference is usually computed approximately. In this section, we review some commonly used approximate inference algorithms for directed and undirected graphical models.

#### 1.5.3.1 Gibbs Sampling

Gibbs sampling is a general framework for approximating a distribution, when the distribution is presumed to come from a specified family, such as Gaussian, Poisson, etc. It is a *Markov chain Monte Carlo* (MCMC) algorithm, in that it iteratively samples from the current estimate of the distribution, constructing a Markov chain that converges to the target (stationary) distribution. Gibbs sampling is efficient because it samples from the conditional distributions of the individual variables, instead of the joint distribution over all variables. To make this more concrete, we examine Gibbs sampling for inference in directed graphical models, in the context of collective classification.

Pseudocode for a Gibbs sampling algorithm (based on [10, 25]) is given in Algorithm 3. At a high level, the algorithm works by iteratively sampling from the posterior distribution of each $Y_i$, i.e., random draws from $P(Y_i \,|\, \mathbf{X}, \mathbf{Y}_{\mathcal{N}_i})$. Like iterative classification, it initializes the posteriors using some function of the local and neighboring features, as well as any observed neighboring labels; this could be the (normalized) output of a local predictor. It then iteratively samples from each of the current posteriors and uses the sampled values to update the probabilities. This process is repeated until the posteriors converge, or until a maximum number of iterations is reached. Upon terminating, the samples for each node are averaged to obtain the approximate marginal label probabilities, $P(Y_i = y)$ (which can be used for prediction by choosing the label with the highest marginal probability). In practice, one typically sets aside a specified number of initial iterations as "burn-in" and averages over the remaining samples. In the limit of infinite data, the estimates should asymptotically converge to the true distribution.

**Algorithm 3** Gibbs Sampling

```
1: for $i = 1, \ldots, n$ do {bootstrapping}
2:   Initialize $P(Y_i \,|\, \mathbf{X}, \mathbf{Y}_{\mathcal{N}_i})$ using local features $X_i$ and the features $\mathbf{X}_{\mathcal{N}_i}$ and *observed*
     labels $\mathbf{Y}^{\ell}_{\mathcal{N}_i} \subseteq \mathbf{Y}_{\mathcal{N}_i}$ of its neighbors
3: end for
4: for $i = 1, \ldots, n$ do {initialize samples}
5:   $\mathcal{S}_i \leftarrow \emptyset$
6: end for
7: repeat {sampling}
8:   $\pi \leftarrow \textsc{GenPerm}(n)$ {generate permutation $\pi$ over $1, \ldots, n$}
9:   for $i = 1, \ldots, n$ do
10:    Sample $s \sim P(Y_i \,|\, \mathbf{X}, \mathbf{Y}_{\mathcal{N}_i})$
11:    $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup s$
12:    Update $P(Y_i \,|\, \mathbf{X}, \mathbf{Y}_{\mathcal{N}_i})$ using $\mathcal{S}_i$
13:  end for
14: until convergence or maximum iterations reached
15: for $i = 1, \ldots, n$ do {compute marginals}
16:   Remove first $T$ samples (i.e., burn-in) from $\mathcal{S}_i$
17:   for $y \in \mathcal{Y}_i$ do
18:     $P(Y_i = y) \leftarrow \frac{1}{|\mathcal{S}_i|} \sum_{s \in \mathcal{S}_i} \mathbb{1}[y = s]$
19:   end for
20: end for
```

Gibbs sampling is a popular method of approximate (marginal) inference in directed graphical models, such as BNs and PRMs. While each iteration of Gibbs sampling is relatively efficient, many iterations are required to obtain an accurate estimate of the distribution, which may be impractical. Thus, there is a trade-off between the running time and the accuracy of the approximate marginals.

#### 1.5.3.2 Loopy Belief Propagation (LBP)

For certain undirected graphical models, exact inference can be computed efficiently via *message passing*, or *belief propagation* (BP), algorithms. These algorithms follow a simple iterative pattern: each variable passes its "beliefs" about its neighbors' marginal distributions, then uses the incoming messages about its own value to updates its beliefs. Convergence to the true marginals is guaranteed for tree-structured MRFs, but is not guaranteed for MRFs with cycles. That said, BP can still be used for approximate inference in general, so-called "loopy" graphs, with a minor modification: messages are discounted by a constant factor (sometimes referred to as *damping*). This algorithm is known as *loopy belief propagation* (LBP).

**Algorithm 4** Loopy Belief Propagation

```
1: for $\{i,j\} \in \mathcal{E}$ do {initialize messages}
2:     for $y \in \mathcal{Y}_j$ do
3:         $m_{i \to j}(y) \leftarrow 1$
4:     end for
5: end for
6: repeat {message passing}
7:     for $\{i,j\} \in \mathcal{E}$ do
8:         for $y \in \mathcal{Y}_j$ do
9:             $m_{i \to j}(y) \leftarrow \alpha \sum_{y'} \phi_{i,j}(y', y) \phi_i(y') \prod_{k \in \mathcal{N}_i \setminus j} m_{k \to i}(y')$
10:        end for
11:    end for
12: until convergence or maximum iterations reached
13: for $i = 1, \ldots, n$ do {compute marginals}
14:     for $y \in \mathcal{Y}_i$ do
15:         $P(Y_i = y) \leftarrow \alpha \phi_i(y) \prod_{j \in \mathcal{N}_i} m_{j \to i}(y)$
16:     end for
17: end for
```

The LBP algorithm shown in Algorithm 4 [45, 46] assumes that the model is a pairwise MRF with singleton potentials defined on each $(X_i, Y_i)$ and pairwise potentials on each adjacent $(Y_i, Y_j)$. We denote by $m_{i \to j}(y)$ the message sent by $Y_i$ to $Y_j$ regarding the belief that $Y_j = y$; $\phi_i(y)$ denotes the $i^{\text{th}}$ local potential function evaluated for $Y_i = y$, and similarly for the pairwise potentials; $\alpha \in (0, 1]$ denotes a constant discount factor. The algorithm begins by initializing all messages to one. It then iterates over the message passing pattern, wherein $Y_i$ passes its beliefs $m_{i \to j}(y)$ to all $j \in \mathcal{N}_i$, using the incoming messages $m_{k \to i}(y)$, for all $k \in \mathcal{N}_i \setminus j$, from the previous iteration. Similar to Gibbs sampling, the algorithm iterates until the messages stabilize, or until a maximum number of iterations is reached, after which we compute the approximate marginal probabilities.

## 1.6 Feature Construction

Thus far, we have used rather abstract problem representations, using an arbitrary data graph $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$, feature variables $\mathbf{X}$ and label variables $\mathbf{Y}$. In practice, how one maps a real-world problem to these representations can have a greater impact than the choice of model or inference algorithm. This process, sometimes referred to as *feature construction* (or *feature engineering*), is perhaps the most challenging aspect of data mining. In this section, we explore various techniques, motivated by concrete examples.

### 1.6.1 Data Graph

Network structure is *explicit* in certain collective classification problems, such as categorizing users in a social network or pages in a website. However, there are other problems that exhibit *implicit* network structure.

An example of this is image segmentation, a common computer vision problem. Given an observed image—i.e., an $m \times n$ matrix of pixel intensities—the goal is to label each pixel as being part of a certain object, from a predetermined set of objects. The structure of the image naturally suggests a grid graph, in which each pixel $(i, j)$ is a node, adjacent to (up to) four neighboring pixels. However, one could also define the neighborhood using the diagonally adjacent pixels, or use wider concentric rings. This decision reflects on one's prior belief of how many pixels directly interact with $(i, j)$; that is, its Markov blanket.

A related setting is part-of-speech (POS) tagging in natural language processing. This task involves tagging each word in a sentence with a POS label. The linear structure of a sentence is naturally represented by a path graph; i.e., a tree in which each vertex has either one or two neighbors. Though, one could also draw edges between words separated by two hops, or, more generally, $n$ hops, depending on one's belief about sentence construction.

The data graph can also be inferred from distances. Suppose the task is to predict which individuals in a given population will contract the flu. Infection is obviously related to proximity, so it is natural to construct a network based on geographic location. Distances can be thresholded to create unweighted edges (e.g., "close" or "not close"), or they can be incorporated as weights (if the model supports it).

Structure can sometimes be inferred from other structure. An example of this is found in document classification in citation networks. In a citation network, there are *explicit* links from papers citing other papers. There are also *implicit* links between two papers that cite the same paper. These "co-citation" edges complete a triangle between three connected nodes.

Links can also be discovered as part of the inference problem [5, 26]. Indeed, collective classification and link prediction are complimentary tasks, and it has been shown that coupling these predictions can improve overall accuracy [29].

It is important to note that not all data graphs are *unimodal*—that is, they may involve multiple types of nodes and edges. In citation networks, authors write papers; authors are affiliated with institutions; papers cite other papers; and so on.

### 1.6.2 Relational Features

Methods based on local classifiers, such as ICA, present a unique challenge to relational feature engineering: while most local classifiers require fixed-length feature vectors, neighborhood sizes are rarely uniform in real relational data. For example, a paper can have any number of authors. If the number of neighbors is bounded by a reasonably small constant (a condition often referred to as *bounded degree*), then it is possible to represent neighborhood information using a fixed-length vector. However, in naturally occurring graphs, such as social networks, this condition is unlikely.

One solution is to aggregate neighborhood information into a fixed number of statistics. For instance, one could count the number of neighbors exhibiting a certain attribute; for a set of attributes, this amounts to a histogram. For numerical or ordinal attributes, one could also use the mean, mode, minimum or maximum. Another useful statistic is the number of triangles, which reflects the connectivity of the neighborhood. More complex, domain-specific aggregates are also possible. Within the inductive logic programming community, aggregation has been studied as a means for *propositionalizing* a relational classification problem [17, 19, 20]. In the machine learning community, Perlich and Provost [33, 34] have studied aggregation extensively.

Aggregation is also useful for defining relational features in graphical models. Suppose one uses a pairwise MRF for social network classification. For each pair of nodes, one could obviously consider the similarities of their local features; yet one could also consider the similarities of their neighborhood structures. A simple metric is the number of common neighbors. To compensate for varying neighborhood sizes, one could normalize the intersection by the union, which is known as *Jaccard similarity*,

$$J(\mathcal{N}_i, \mathcal{N}_j) \triangleq \frac{\mathcal{N}_i \cap \mathcal{N}_j}{\mathcal{N}_i \cup \mathcal{N}_j}.$$

This can be generalized to the number (or proportion) of common neighbors who exhibit a certain attribute.

## 1.7 Applications of Collective Classification

Collective classification is a generic problem formulation that has been successfully applied to many application domains. Over the course of this chapter, we have already discussed a few popular applications, such as document classification [41, 44], image segmentation [1] and POS tagging [22]. In this section, we briefly review some other uses.

One such problem, which is related to POS tagging, is *optical character recognition* (OCR). The goal of OCR is to automatically convert a digitized stream of handwritten characters into a text file. In this context, each node represents a scanned character; its observed features $X_i$ are a vectorized pixel grid, and the label $Y_i$ is chosen from the set of ASCII (or ISO) characters. While this can be predicted fairly well using a local classifier, it has been shown that considering intra-character relationships can be beneficial [42], since certain characters are more (or less) likely to occur before (or after) other characters.

Another interesting application is activity detection in video data. Given a recorded video sequence (say, from a security camera) containing multiple actors, the goal is to label each actor as performing a certain action (from a predefined set of actions). Assuming that bounding boxes and tracking (i.e., identity maintenance) are given, one can bolster local reasoning with spatiotemporal relational reasoning. For instance, it is often the case that certain actions associate with others: if an actor is crossing the street, then other actors in the proximity are likely crossing the street; similarly, if one actor is believed to be talking, then other actors in the proximity are likely either talking or listening. One could also reason about action transitions: if an actor is walking at time $t$, then it is very likely that they will be walking at time $t + 1$; however, there is a small probability that they may transition to a related action, such as crossing the street or waiting. Incorporating this high-level relational reasoning can be considered a form of collective classification. This approach has been used in a number of publications [7, 16] to achieve current state-of-the-art performance on benchmark datasets.

Collective classification is also used in computational biology. For example, researchers studying protein-protein interaction networks often need to annotate proteins with their biological function. Discovering protein function experimentally is expensive. Yet, protein function is sometimes correlated with interacting proteins; so, given a set of labeled proteins, one can reason about the remaining labels using collective methods [23].

The final application we consider is viral marketing, which is interesting for its relationship to *active* collective classification. Suppose a company is introducing a new product to a population. Given the social network and the individuals' (i.e., local) demographic features, the goal is to determine which customers will be interested in the product. The mapping to collective classification is straightforward. The interesting subproblem is in how one acquires labeled training data. Customer surveys can be expensive to conduct, so companies want to acquire the smallest set of user opinions that will enable them to accurately predict the remaining user opinions. This can be viewed as an active learning problem for collective classification [3].

## 1.8 Conclusion

Given the recent explosion of relational and network data, collective classification is quickly becoming a mainstay of machine learning and data mining. Collective techniques leverage the idea that connected (related) data objects are in some way correlated, performing joint reasoning over a high-dimensional, structured output space. Models and inference algorithms range from simple iterative frameworks to probabilistic graphical models. In this chapter, we have only discussed a few; for greater detail on these methods, and others we did not cover, we refer the reader to Macskassy and Provost [25] and Sen et al. [37]. Many of the algorithms discussed herein have been implemented in NetKit-SRL[3], an open-source toolkit for mining relational data.

**Acknowledgements**

This material is based on work supported by the National Science Foundation under Grant No. 0746930 and Grant No. IIS1218488.

[3]http://netkit-srl.sourceforge.net

# Bibliography

[1] Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., Ng, A.: Discriminative learning of markov random fields for segmentation of 3d scan data. In: International Conference on Computer Vision and Pattern Recognition (2005)

[2] Bach, S.H., Huang, B., London, B., Getoor, L.: Hinge-loss markov random fields: Convex inference for structured prediction. In: Uncertainty in Artificial Intelligence (2013)

[3] Bilgic, M., Getoor, L.: Effective label acquisition for collective classification. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 43–51 (2008), winner of the KDD'08 Best Student Paper Award.

[4] Bilgic, M., Mihalkova, L., Getoor, L.: Active learning for networked data. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10) (2010)

[5] Bilgic, M., Namata, G.M., Getoor, L.: Combining collective classification and link prediction. In: Workshop on Mining Graphs and Complex Structures in Int. Conf. of Data Mining (2007)

[6] Chakrabarti, S., Dom, B., Indyk, P.: Enhanced hypertext categorization using hyperlinks. In: International Conference on Management of Data. pp. 307 – 318 (1998)

[7] Choi, W., Shahid, K., Savarese, S.: What are they doing?: Collective activity classification using spatio-temporal relationship among people. In: VS (2009)

[8] Cortes, C., Mohri, M., Pechyony, D., Rastogi, A.: Stability analysis and learning bounds for transductive regression algorithms. CoRR abs/0904.0814 (2009)

[9] Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: International Joint Conference on Artificial Intelligence. (1999)

[10] Geman, S., Geman, D.: Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1990)

[11] Getoor, L.: Advanced Methods for Knowledge Discovery from Complex Data, chap. Link-based classification. Springer (2005)

[12] Getoor, L., Friedman, N., Koller, D., Taskar, B.: Learning probabilistic models of link structure. Journal of Machine Learning Research. (2002)

[13] Getoor, L., Segal, E., Taskar, B., Koller, D.: Probabilistic models of text and link structure for hypertext classification. In: IJCAI Workshop on Text Learning: Beyond Supervision (2001)

[14] Jensen, D., Neville, J., Gallagher, B.: Why collective inference improves relational classification. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2004)

[15] Ji, M., Sun, Y., Danilevsky, M., Han, J., Gao, J.: Graph regularized transductive classification on heterogeneous information networks. In: Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part I. pp. 570–586. ECML PKDD'10, Springer-Verlag, Berlin, Heidelberg (2010)

[16] Khamis, S., Morariu, V.I., Davis, L.S.: Combining per-frame and per-track cues for multi-person action recognition. In: European Conference on Computer Vision (2012)

[17] Knobbe, A., deHaas, M., Siebes, A.: Propositionalisation and aggregates. In: Proceedings of the Fifth European Conference on Principles of Data Mining and Knowledge Discovery (2001)

[18] Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)

[19] Kramer, S., Lavrac, N., Flach, P.: Propositionalization approaches to relational data mining. In: Dzeroski, S., Lavrac, N. (eds.) Relational Data Mining. Springer-Verlag, New York (2001)

[20] Krogel, M., Rawles, S., Zeezny, F., Flach, P., Lavrac, N., Wrobel, S.: Comparative evaluation of approaches to propositionalization. In: International Conference on Inductive Logic Programming. (2003)

[21] Kuwadekar, A., Neville, J.: Relational active learning for joint collective classification models. In: Proceedings of the 28th International Conference on Machine Learning (2011)

[22] Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the International Conference on Machine Learning. pp. 282 – 289 (2001)

[23] Letovsky, S., Kasif, S.: Predicting protein function from protein/protein interaction data: a probabilistic approach. Bioinformatics 19, 197–204 (2003)

[24] Lu, Q., Getoor, L.: Link based classification. In: Proceedings of the International Conference on Machine Learning. (2003)

[25] Macskassy, S., Provost, F.: Classification in networked data: A toolkit and a univariate case study. Journal of Machine Learning Research. (2007)

[26] Macskassy, S.A.: Improving learning in networked data by combining explicit and mined links. In: Proceedings of the Twenty-Second Conference on Artificial Intelligence (2007)

[27] Macskassy, S.A.: Using graph-based metrics with empirical risk minimization to speed up active learning on networked data. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2009)

[28] Mcdowell, L.K., Gupta, K.M., Aha, D.W.: Cautious inference in collective classification. In: Proceedings of AAAI. (2007)

[29] Namata, G.M., Kok, S., Getoor, L.: Collective graph identification. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2011)

[30] Namata, G.M., London, B., Getoor, L., Huang, B.: Query-driven active surveying for collective classification. In: Workshop on Mining and Learning with Graphs (2012)

[31] Neville, J., Jensen, D.: Iterative classification in relational data. In: Workshop on Statistical Relational Learning, AAAI (2000)

[32] Neville, J., Jensen, D.: Relational dependency networks. Journal of Machine Learning Research. 8, 653–692 (2007)

[33] Perlich, C., Provost, F.: Aggregation-based feature invention and relational concept classes. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2003)

[34] Perlich, C., Provost, F.: Distribution-based aggregation for relational learning with identifier attributes. Machine Learning Journal 62(1-2), 65–105 (2006)

[35] Rattigan, M.J., Maier, M., Wu, D.J.B., Pei, X., Tan, J., Wang, Y.: Exploiting network structure for active inference in collective classification. In: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops. pp. 429–434. ICDMW '07, IEEE Computer Society (2007)

[36] Richardson, M., Domingos, P.: Markov logic networks. Machine Learning (2006)

[37] Sen, P., Namata, G.M., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. AI Magazine 29(3), 93–106 (2008)

[38] Sharara, H., Getoor, L., Norton, M.: Active surveying: A probabilistic approach for identifying key opinion leaders. In: The 22nd International Joint Conference on Artificial Intelligence (IJCAI '11) (2011)

[39] Talukdar, P.P., Crammer, K.: New regularized algorithms for transductive learning. In: ECML/PKDD (2). pp. 442–457 (2009)

[40] Taskar, B., Abbeel, P., Koller, D.: Discriminative probabilistic models for relational data. In: Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence. (2002)

[41] Taskar, B., Chatalbashev, V., Koller, D.: Learning associative markov networks. In: Proceedings of the International Conference on Machine Learning. (2004)

[42] Taskar, B., Guestrin, C., Koller, D.: Max-margin markov networks. In: Neural Information Processing Systems. (2003)

[43] Wu, M., Schölkopf, B.: Transductive classification via local learning regularization. Journal of Machine Learning Research - Proceedings Track 2, 628–635 (2007)

[44] Yang, Y., Slattery, S., Ghani, R.: A study of approaches to hypertext categorization. Journal of Intelligent Information Systems 18 (2002)

[45] Yedidia, J., Freeman, W., Weiss, Y.: Constructing free-energy approximations and generalized belief propagation algorithms. In: IEEE Transactions on Information Theory. pp. 2282–2312 (2005)

[46] Yedidia, J., W.T.Freeman, Weiss, Y.: Generalized belief propagation. In: Neural Information Processing Systems. vol. 13, pp. 689–695 (2000)

[47] Zhou, D., Bousquet, O., Lal, T.N., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: NIPS (2003)

[48] Zhu, X., Ghahramani, Z.: Learning from labeled and unlabeled data with label propagation. Tech. rep., Carnegie Melon University (2002)

[49] Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised learning using gaussian fields and harmonic functions. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03) (2003)