Training Iterative Collective Classifiers with Back-Propagation

Shuangfei Fan
Dept. of Electrical & Computer Engineering
Virginia Tech, Blacksburg, VA 24061
sophia23@vt.edu

Bert Huang
Dept. of Computer Science
Virginia Tech, Blacksburg, VA 24061
bhuang@vt.edu

ABSTRACT

We propose a new method for training iterative collective classifiers for labeling nodes in network data. The iterative classification algorithm (ICA) is a canonical method for incorporating relational information into the classification process. Yet, existing methods for training ICA models rely on computing relational features using the true labels of the nodes. This method introduces a bias that is inconsistent with the actual prediction algorithm. In this paper, we introduce a variant of ICA, ICA with back-propagation (BPICA) as a procedure analogous to recurrent neural network prediction, which enables gradient-based strategies for optimizing over model parameters. We demonstrate that by training BPICA, we more directly optimize the training loss of collective classification, which translates to improved accuracy and robustness on real network data. This robustness enables effective collective classification in settings where local classification is very noisy, settings that previously were particularly challenging for ICA and variants.

Keywords

Collective classification; iterative classification algorithm; back-propagation; graph mining

1. INTRODUCTION

Data science tasks often require reasoning about networks of connected entities, such as social and information networks. In classification tasks, the connections among network nodes can have important effects on node-labeling patterns, so models that perform classification in networks should consider network structure to fully represent the underlying phenomena. For example, when classifying individuals by their personality traits in a social network, a common pattern is that individuals will communicate with like-minded individuals, suggesting that predicted labels should also tend to be uniform among connected nodes. Collective classification methods aim to make predictions based on this relationship between network structure and labels. In this paper, we improve a training procedure for a collective classification framework that will enable an algorithm to more directly optimize the performance of trained collective classifiers.

DOI: 10.1145/1235

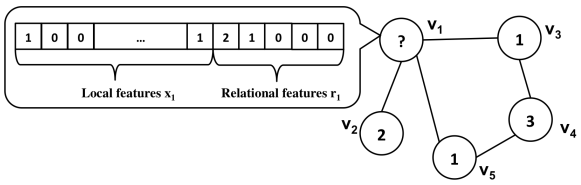


Figure 1: Illustration of the iterative classification algorithm (ICA). ICA repeatedly computes relational features based on current estimates of node labels, then classifies each nodes using a concatenated input vector of its local features and its relational features.

Iterative classification is a framework that enables a variety of basic machine learning methods to incorporate information from networks. The base machine learning method can be any standard classifier that takes information describing each example as input and then outputs a label. The iterative classification algorithm (ICA) operates by using previous predictions about neighboring nodes as inputs to the current predictor, as illustrated in Figure 1. For example, if the base predictor is a logistic regression model, one input feature may be the average predicted class probability of a node's neighbors. This pipeline creates a feedback loop that allows models to pass information through the network and capture the effect of network structure on classification. Despite the fact that the feedback loop is the most important aspect of ICA, existing approaches train these models in a manner that ignores the feedback-loop structure. In this paper, we introduce BPICA, which corrects this discrepancy between the learning and prediction algorithms by incorporating principles used in deep learning and recurrent neural networks into the training process.

Existing learning algorithms for iterative classification resort to an approximation based on the unrealistic assumption that the predicted labels of neighbors are their true classes [13, 19]. This assumption is overly optimistic. If it were true, iteration would be unnecessary. Because the assumption is overly optimistic, it causes the learned models to cascade and amplify errors when the assumption is broken in early stages of prediction. In contrast, the actual ICA predictive procedure uses predicted neighbor labels as feedback for each subsequent prediction, which means that if the model was trained expecting these predicted labels to be perfect, it will

not be robust to situations where predictions are noisy or inaccurate. In this paper, we correct this faulty assumption and develop an algorithm that trains models for iterative classification by treating the intermediate predictions as latent variables. We compute gradients to the classification loss function using iterated applications of the chain rule, leading to a method similar to the back-propagation approach for training neural networks.

To compute gradients for ICA using the chain rule, we break down the ICA process into differentiable (or sub-differentiable) operations. In many cases, the base classifier is differentiable with respect to its parameters. For example, if it is a logistic regression, it has a well-studied gradient. ICA also involves aggregation of predictions from neighboring nodes, which can be casted as a sparse matrix multiplication, through which gradients can be propagated. Finally, because the same base-classifier parameters should be used at all iterations of ICA, we can use methods from recurrent neural networks such as back-propagation through time (BPTT) [24] to compute the combined gradient. In contrast with existing strategies for training ICA, the resulting training optimization more closely mimics the actual procedure that ICA uses for prediction.

We evaluate the proposed gradient-based training algorithm on data sets where collective classification has been shown to be helpful in previous studies. These data sets are citation networks [21], where documents are connected in a network if one cites the other, and they are to be classified according to their main topic area. We demonstrate that BPICA is able to train classifiers that are robust to situations where local predictions are inaccurate.

The remainder of the paper is organized as follows. In Section 2, we review related work that our contribution builds upon. In Section 3, we review the iterative classification algorithm and introduce the notation we will use to describe our approach. In Section 4, we introduce BPICA and describe its back-propagation learning procedure. In Section 5, we describe experiments evaluating the behavior of BPICA on node classification data sets, demonstrating it as an effective of robust collective classification.

2. RELATED WORK

Node classification is one of the fundamental tasks in analysis of network data [5, 12]. *Collective classification* addresses this task by making joint classifications of connected nodes [11, 20, 23]. Gibbs sampling (GS) is another approach for collective classification using the iterative classification framework [16, 21]. The idea is to estimate the full joint distribution by iteratively doing classification using relational information for fixed iterations. However, it can require thousands of iterations to converge, so it is often more expensive than ICA. ICA and GS has been shown repeatedly to be an effective framework for collective classification [10, 14, 16, 19, 21]. ICA is a central algorithm in the research fields of graph mining and statistical relational learning, and since it is an algorithm that makes joint predictions, it is a key example of structured prediction.

One of the more natural motivations for collective classification comes from the study of social networks, where the phenomenon of *homophily*—the tendency of individuals to interact with other similar individuals—has been identified. For example, studies have shown homophily in age, gender, race and ethnicity, and geographical location [2, 17]. Nevertheless, the types of dependencies that can exist in networks are not limited to assortative relationships—where connected nodes tend to be similar. Methods such as ICA enable models to encode both assortative and non-assortative phenomena.

Through the interpretation of non-terminal classifications as latent variables, ICA can be related to deep learning methods. Since the same classifier is used to predict each latent layer, ICA is most related to recurrent neural networks (RNNs), which feature a similar feedback loop in which an output of a neural network is used as its input in a subsequent iteration. RNNs were introduced decades ago [1, 3], but they have recently become prominent because of their effectiveness at modeling sequences, such as those occurring in natural language processing, e.g., [6, 7, 18, 22]. A now standard method for gradient optimization of RNN parameters is known as back-propagation through time [8, 9, 24], which unrolls recurrent networks and computes gradients for the parameters separately before combining them into a single update.

3. THE ITERATIVE CLASSIFICATION ALGORITHM

In this section, we review the *iterative classification algorithm* (ICA) and the standard method for training its parameters. ICA is a framework for node classification in networks. ICA is given a decorated graph $G = \{V, E, X\}$, where $V = \{v_1, \ldots, v_n\}$, E contains pairs of linked nodes $(v_i, v_j) \in E$, and each node is associated with a respective feature vector $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ with each vector $x_i \in \mathbb{R}^d \equiv \mathcal{X}$. Using these inputs, ICA outputs a set of predictions $\{y_1, \ldots, y_n\}$ classifying each of the nodes in V into a discrete label space $\mathcal{Y}$. Throughout this paper, we will consider the multi-class setting, in which the labels can take one of k class-label values. In particular, we use the so-called "one-hot" representation for multi-class labels, such that $\mathcal{Y}$ is space of binary vectors with exactly one nonzero entry, indicating the class membership. ICA makes the label predictions by iteratively applying classification functions that label nodes based on their *local features* x_i and their dynamic *relational features* $R = \{r_1, \ldots, r_n\}$, each in a common space $r_i \in \mathcal{R}$. In other words, the classifier is a function g that maps $\mathcal{X} \times \mathcal{R}$ to $\mathcal{Y}$.

The dynamic relational features are computed based on the current estimated labels of each node. They enable the classifier to reason about patterns of labels among connected nodes. For example, a common dynamic relational feature is the average prediction of neighboring nodes. I.e.,

$$\mathbf{r}_i = \frac{1}{|\{(i,j) \in E|\}|} \sum_{j:(i,j)\in E} \mathbf{y_j}. \tag{1}$$

Using any such aggregation statistic creates a relational feature vector that has dimensionality k, where each entry is the occurrence rate of its corresponding label in the node's neighbors.

Since the dynamic relational features are computed based on the output of the classifier, the entire process is iterated: (1) all nodes are labeled by the classifier using the current dynamic relational features, then (2) the dynamic relational features are updated based on the new labels. These two phases are repeated either until the predictions converge and do not change between iterations or until a cutoff point.

The ICA framework is general in that any classification function can be used and any definition of dynamic relational attributes can be used. In practice, researchers have used naive Bayes, logistic regression, support vector machines as the classifiers, and they have used averages, sums, and presence as relational features [10, 14, 19, 21].

The existing training procedure for ICA trains the classifiers by computing relational features using the training labels. Given a training set consisting of a set of nodes V, edges E, node features X, and ground-truth labels Y, one generates relational features R using the true training labels, creating fully instantiated, fully labeled inputs for the classifier. Then any appropriate training procedure for the base classifier is used to fit the model parameters.

In many settings, we consider real-valued local and relational features, so each node is described by a feature vector created by concatenating its local features with its relational features $[\mathbf{x}_i\ \mathbf{r}_i]$. In this case, it is convenient to notate the classification of the entire graph in terms of matrices. Let $\mathbf{X}$ denote the feature matrix for the graph, such that the ith row of $\mathbf{X}$, i.e., $\mathbf{x}_i$ is the (transposed) feature vector for node v_i. Similarly, let $\mathbf{R}$ denote the relational feature matrix, such that the ith row of $\mathbf{R}$ is the dynamic relational feature vector of node v_i. For convenience, we consider the case where one type of dynamic relational feature is used, meaning the dimensionality of $\mathbf{R}$ is n by k.

Common instantiations of ICA use linear, multi-class classifiers, in which the classifiers are based on weighted linear combination of the features. In these cases, the model parameters for the classifier typically take the form of weight matrices $\mathbf{U} \in \mathbb{R}^{d\times k}$ and $\mathbf{V} \in \mathbb{R}^{k\times k}$. There is a column in each matrix for each class, and each column contains a weighting of the local and relational feature dimensions, respectively. For example, a multi-class logistic regression classifier computes estimated label probabilities using the formulas

$$\begin{aligned}\tilde{\mathbf{Y}} &= \exp\left(\mathbf{XV} + \mathbf{RU}\right), \\ Y'_{i\ell} &= \frac{\tilde{Y}_{i\ell}}{\sum_{\ell'} \tilde{Y}_{i\ell'}}.\end{aligned} \tag{2}$$

Moreover, for neighbor-averaging relational features, it is also convenient to notate their computation as a matrix operation. Let $\mathbf{A}$ be an n by n weighted adjacency matrix, where its rows are normalized to unit sums. I.e.,

$$A_{ij} = \begin{cases} 1/n_i & \text{if } (i,j) \in E \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where n_i is the number of node v_i's neighbors, $n_i = |\{(i,\ell) \in E\}|$.

Previous ICA training procedures for such a classifier would fit the $\mathbf{U}$ and $\mathbf{V}$ matrices by setting $\hat{\mathbf{R}} = \mathbf{AY}$, then using $\mathbf{X}$ and $\hat{\mathbf{R}}$ as the input to a standard logistic regression training scheme, such as regularized maximum likelihood.

This training methodology is ideal in the situation where we expect a *perfect-classification fixed-point*. In such a fixed point, the relational features are computed using the true labels, and the classifiers perform perfectly, exactly predicting the true labels of the nodes; the relational features are computed, using the perfectly predicted labels, to be exactly the same features that are computed using the true training labels; since the relational features are computed using the exactly predicted true labels, the output is the same as in the first step, and the algorithm converges perfectly to the true labels. Using the matrix form of the relational feature computation above, this fixed point would be characterized as

$$\mathbf{Y} = g\left([\mathbf{X},\ \mathbf{AY}]\right). \tag{4}$$

Unfortunately, such a fixed point is unrealistic. In practice, the classifications can be inaccurate or made with low confidence due to a lack of reliable local information. Thus, training the model to expect that the neighbor labels be perfect creates an overconfidence that can lead to cascading errors. In the next section, we introduce our main contribution: an approach that more directly optimizes the loss associated with a variant of the ICA prediction process.

Algorithm 1 The Iterative Classification Algorithm

1: **Input:** Graph $G = \{V, E\}$ with node features X, number of iterations T, classifier g.
2: Initialize labels Y {e.g., random or uniform across all classes}
3: **for** t from 1 to T **do**
4: **for** each node in V **do**
5: Calculate r_i with current label estimates Y {e.g., Equation (1) or in the vector form in Section 3, $\mathbf{r}_i \leftarrow [\mathbf{AY}]_i$}.
6: Set $y_i \leftarrow g([x_i, r_i])$
7: **end for**
8: **end for**
9: **return** labels Y

4. ICA WITH BACK-PROPAGATION

In this section, we present our main contribution. We recast a slight variant of ICA as a series of differentiable operations such that a training loss function can be directly optimized.

Additionally to the fact that we cast the operations of an ICA variant as differentiable operations, the more important difference between the proposed training approach and the existing methods is that we enable training of the collective classification in a manner that more directly mimics how they will be applied. At test time, a collective classifier is often given only local features of nodes connected in a network. Thus, any relational features derived from neighbor labels is derived from *predicted* neighbor labels. A classifier considering these neighbor labels should therefore consider common patterns of misclassification of neighbor labels. The previous training procedure invites the classifier to become overly reliant on the verity of the neighbor labels.

4.1 Prediction

For linear classifiers, ICA can be viewed as the iteration of the following steps. First, the local and relational features are multiplied by the model weights. Second, the linear product prediction scores are *squashed*. For example, they could be converted to values in $[0, 1]$ via the *logistic* sigmoid function, or into a probability vector via the *soft-max* function, i.e., the normalized multi-class logistic. Third, the squashed predictions are aggregated into new dynamic relational features. The following three equations express these steps in matrix

form, respectively.

$$\begin{aligned}\mathbf{Z}^{(t)} &= \mathbf{X}\mathbf{V} + \mathbf{R}^{(t-1)}\mathbf{U} \\ \mathbf{P}^{(t)} &= f(\mathbf{Z}^{(t)}) \\ \mathbf{R}^{(t)} &= \mathbf{A}\mathbf{P}^{(t)}\end{aligned} \tag{5}$$

These updates are applied from iterations $t = 1$ to $t = T$.

A common squashing function is the logistic sigmoid function

$$f(z) = \frac{1}{1 + \exp(-z)}, \tag{6}$$

whose derivative is

$$f'(z) = f(z)(1 - f(z)). \tag{7}$$

For the final output, we use the soft-max squashing function to convert the prediction scores $\mathbf{Z}$ to a multinomial probability vector

$$Y'_{ij} = \frac{\exp\left(Z_{ij}^{(T)}\right)}{\sum_k \exp\left(Z_{ik}^{(T)}\right)}, \tag{8}$$

which is equivalent to the multi-class logistic regression formula in Equation (2).

4.2 Learning

Each of these steps is differentiable, and this variant of ICA iteratively performs linear products and nonlinear squashing functions. Figure 2 illustrates BPICA as a neural network. We can thus use a standard method for training such networks known as back-propagation through time, which unfolds the iterations of the algorithm (see Figure 2), computes gradients for each unfolded layer using feed-forward back-propagation, and combines the gradients to perform learning updates.

The main objective function is the negative log likelihood of the true labels

$$\begin{aligned}L(\mathbf{Z}) &= -\sum_{ij} Y_{ij} \log Y'_{ij} \\ &= \log\left(\sum_k \exp(Z_{ik}^{(T)})\right) - \sum_{ij} Y_{ij} Z_{ij}^{(T)}\end{aligned} \tag{9}$$

In order to capture more history information, errors can be propagated further and this process is called back-propagation through time.

Define the recursive error signal at t-th layer as:

$$\boldsymbol{\Delta}^{(t)} = \frac{\partial L}{\partial(\mathbf{Z}^{(t)})} \tag{10}$$

To apply BPTT training, a single entry of an error matrix in the previous layer $\delta_{pj}^{(t-1)}$ can be calculated by the error in the next layer:

$$\delta_{pj}^{(t-1)} = \sum_{m=1}^{k} \delta_{pm}^{(t)} \mathbf{U}_{mj} f'(\mathbf{Z}_{pj}^{(t-1)}) \mathbf{A}_{pp} \tag{11}$$

where p is the node index, j is the class index and k is the number of classes. Therefore, the errors are propagated from the output layer to the middle layers.

For convenience, we can rewrite Equation (11) in matrix form, for each layer (i.e., iterations) except for the last one, the error $\boldsymbol{\Delta}^{(t)}$ can be calculated backward through the following recursive definition

$$\boldsymbol{\Delta}^{(t-1)} = \mathbf{A}^\top \boldsymbol{\Delta}^{(t)} \mathbf{U}^\top \odot \mathbf{F}^{(t-1)} \tag{12}$$

where

$$\mathbf{F}_{ij}^{(i)} = f'_{ij}(\mathbf{Z}_{ij}^{(i)}). \tag{13}$$

For soft-max classifier, the error gradient $\boldsymbol{\Delta}^{(T)}$ for the last layer is

$$\boldsymbol{\Delta}^{(T)} = \frac{\partial L}{\partial(\mathbf{Z}^{(T)})} = \mathbf{Y}' - \mathbf{Y}. \tag{14}$$

While this recursive gradient computation is analogous to standard back-propagation equations for computing gradients of layered neural networks, an important difference here is the inclusion of the $\mathbf{A}$ matrix, which forms the relational features. The relational feature computation essentially acts as an extra neural layer between iterations, and this equation reflects how error propagates through that relational feature construction.

Then given these error signals at each layer, we can obtain the gradients for two unrolled weight matrices by applying chain rule.

$$\begin{aligned}\nabla(\mathbf{V})^{(t)} &= \frac{\partial(L)}{\partial(\mathbf{Z}^{(t)})} \frac{\partial(\mathbf{Z}^{(t)})}{\partial(\mathbf{V})} = \mathbf{X}^\top \boldsymbol{\Delta}^{(t)} \\ \nabla(\mathbf{U})^{(t)} &= \frac{\partial(L)}{\partial(\mathbf{Z}^{(t)})} \frac{\partial(\mathbf{Z}^{(t)})}{\partial(\mathbf{U})} = (\mathbf{R}^{(t-1)})^\top \boldsymbol{\Delta}^{(t)}\end{aligned} \tag{15}$$

Using the methodology of back-propagation through-time, the final derivative is the sum of each unrolled derivative. The entire gradient computation is summarized in Algorithm 2. Given the gradients, a variety of learning optimization procedures are possible, such as the simple gradient descent approach

$$\begin{aligned}\mathbf{V} &\leftarrow \mathbf{V} - \eta \sum_{\tau=0}^{T-1} \mathbf{X}^\top \boldsymbol{\Delta}^{(T-\tau)} \\ \mathbf{U} &\leftarrow \mathbf{U} - \eta \sum_{\tau=0}^{T-1} (\mathbf{R}^{(T-\tau-1)})^\top \boldsymbol{\Delta}^{(T-\tau)}\end{aligned} \tag{16}$$

where η is a learning rate that may decrease with further iterations of gradient descent. More advanced gradient-based methods are also easily applicable once the gradient computation is implemented. For example, one particularly effective strategy is the *adagrad* approach [4], which we use in our experiments as described in Section 5.

One important fact about this training procedure is that it results in a non-convex objective function. Even though the classifier is linear, the repeated squashing and recurrence of the labeling through the relational features leads to a non-convex objective. In experiments, we find that local solutions to the non-convex optimization produce effective models.

5. EXPERIMENTS

In this section, we describe experiments that test whether the proposed training method for BPICA is able to improve upon existing methods of training iterative classifiers. We explore scenarios where the local classifier produces inaccurate predictions, challenging the faulty assumption implied by training ICA and Gibbs sampling (GS) with relational

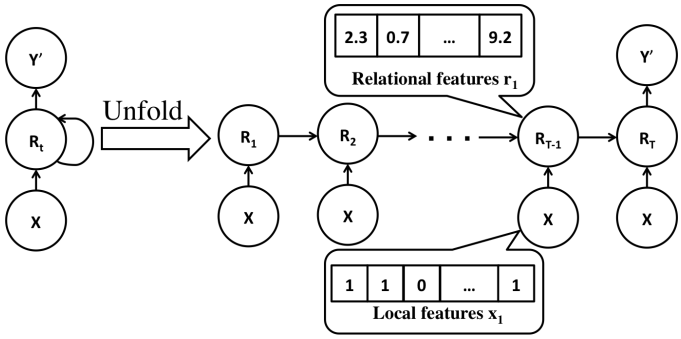


Figure 2: Structure of BPICA. On the left, the recurrent form illustrates the recursive computation of iterative classification. On the right, the unfolded structure explicitly considers each iteration as a separate operation, enabling gradients to be computed for each step.

Algorithm 2 BPICA Gradient Computation

Input: Graph $G = \{V, E\}$, training labels Y, number of iterations T.
Initialize parameter matrices $\mathbf{U}$, $\mathbf{V}$, set $\mathbf{R}^{(0)} = 0$.
Generate matrix $\mathbf{A}$ based on E {e.g., Equation (3)}
for t from 1 to T **do**
  $\mathbf{Z}^{(t)} = \mathbf{XV} + \mathbf{R}^{(t-1)}\mathbf{U}$.
  $\mathbf{P}^{(t)} = f(\mathbf{Z}^{(t)})$ where f is the activation function.
  $\mathbf{R}^{(t)} = \mathbf{AP}^{(t)}$
end for
Compute gradients of loss for output layer $\mathbf{\Delta^{(T)}}$
for t from T to 2 **do**
  $\mathbf{\Delta}^{(t-1)} \leftarrow \mathbf{A}^\top \mathbf{\Delta}^{(t)} \mathbf{U}^\top \odot \mathbf{F}^{(t-1)}$
end for
$\nabla(\mathbf{V}) \leftarrow \sum_{\tau=0}^{T-1} \mathbf{X}^\top \mathbf{\Delta}^{(T-\tau)}$
$\nabla(\mathbf{U}) \leftarrow \sum_{\tau=0}^{T-1} (\mathbf{R}^{(T-\tau-1)})^\top \mathbf{\Delta}^{(T-\tau)}$
return $\nabla(\mathbf{U})$ and $\nabla(\mathbf{V})$

features computed using the true labels. The results illustrate that our hypothesis is correct, identifying a variety of settings where BPICA better optimizes the training objective and produces more accurate predictions on held-out data.

5.1 Setup

For each experiment, we evaluate on four different approaches for node classification: (1) local prediction using only the local features; (2) ICA trained using the true labels; (3) GS trained using the true labels; and (4) BPICA trained using back-propagation. The local predictor is trained using a multi-class logistic regression loss function, ICA and GS are also trained using a multi-class logistic regression loss, except with the concatenated relational features computed from the true labels in addition to local features as input. BPICA is trained using the training labels only in computing the loss, but never as input to the classifier in any form.

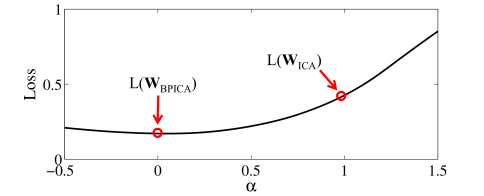
For each of the learning objectives, we optimize using the adagrad approach [4], in which gradients are rescaled based on the magnitude of previously seen gradients. For the gradient $\mathbf{g}_\tau$ at optimization iteration τ, one updates the variable θ with

$$\theta_\tau \leftarrow \theta_{\tau-1} - \eta \frac{\mathbf{g}_\tau}{\sqrt{\sum_{i=1}^{\tau} \mathbf{g}_i \odot \mathbf{g}_i}}, \tag{17}$$

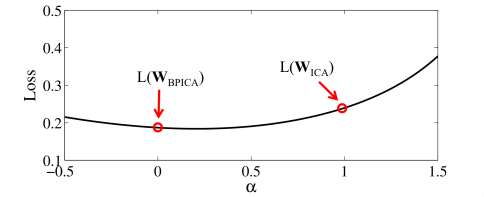
which is written with a slight abuse of notation in that the gradient is divided elementwise by the historical magnitude. Adagrad is one of many approaches that has been shown in practice to accelerate convergence of gradient-based optimization. Each training optimization is done with 1,000 iterations of adagrad and an initial learning rate of $\eta = 0.01$. We evaluate performance of each method using a range of regularization parameter settings from 1×10^{-3} to 1×10^{3}.

For each experimental trial, we perform snowball sampling to extract a random 1/5 of the nodes to hold out as a separate, isolated test network. We train on the induced graph of the 4/5 remaining nodes, and measure predictive accuracy on both the training graph and the held-out testing graph. The training accuracy should more closely reflect whether each method's training strategy effectively optimizes the model to fit the observed data, and the testing accuracy should additionally reflect how well the learned model generalizes. We compute both training and testing accuracy by feeding the learned model only the local features and link structure, meaning that though ICA and GS is typically *trained* with the training labels as input, we do not provide the labels to them when *evaluating* its training accuracy. Our hypothesis is that by directly computing the gradient of the actual prediction procedure for collective classification, BPICA will produce better training performance, which should translate to better testing performance.

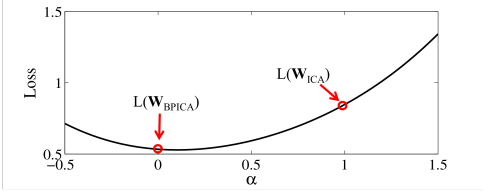
The problem BPICA aims is to solve the discrepancy between training ICA with relational features based on the true labels and the fact that at prediction time, ICA uses estimated labels. To illustrate this discrepancy, our experiments consider situations where local classification becomes more and more difficult. We generate versions of the data set where different fractions of the features are removed, in the range $[0.0, 0.9]$. For example, when this fraction is 0.5, half of the features are removed and made unavailable to the learning and prediction algorithms. In effect, the experiments are run on versions of the data sets where prediction is harder, and more relevantly, where the assumption that the predicted labels are exactly the true labels becomes more and more incorrect. If the training procedure of BPICA is truly more robust to this scenario, we expect its improvement over ICA to become more pronounced as (local) prediction becomes more difficult.



(a) Cora



(b) CiteSeer



(c) Facebook

Figure 3: Loss function comparison for BPICA and ICA on three data sets. Each curve is a cross-section of the high-dimensional training objective function. The plots contain the solutions from BPICA and the solution training ICA with the true training labels. The solutions of BPICA are close to local minima while the ICA weights are not.

5.2 Data

We experimented with three data sets, two bibliographic data sets and one social network data set. The Cora data set, the first bibliographic data set, is a collection of 2708 machine learning publications categorized into seven classes [21]. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3703 unique words.

The CiteSeer data set, the second bibliographic data set, is a collection of 3312 research publications crawled from the CiteSeer repository [21]. It consists of 3312 scientific publications categorized into six classes. Each publication in the data set is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

The social network data we use is the Facebook ego network data [15], which includes users' personal information, friend lists, and ego networks. For our experiments, we combined the ego networks and node features to form a single connected network, with 4039 users' anonymized profile data and links between them. We used the feature "education type," as a label, aiming to predict how many degrees each user has, i.e., the four classes are whether their highest level of education is secondary, undergraduate, or graduate, or if they omit this information from their profiles.

5.3 Empirical Evaluation of Loss

To illustrate that BPICA optimizes the training loss function better than training with the true labels, we can compare the training loss associated with the soft-max output loss, i.e., the regularized logistic-regression training log-likelihood. We first train the model weights using BPICA and the relational features computed using the true labels. We then define a tradeoff function

$$l(\alpha) = L(\mathbf{W}_{\text{BPICA}} + \alpha(\mathbf{W}_{\text{ICA}} - \mathbf{W}_{\text{BPICA}})) \qquad (18)$$

where L is the loss function, $\mathbf{W}_{\text{BPICA}} = [\mathbf{V}; \mathbf{U}]$ is the weight matrix from the model trained by BPICA, $\mathbf{W}_{\text{ICA}}$ is the weight matrix trained by the ICA strategy of using the true-label relational features. We apply the loss function to different weight matrices on the line that connects $\mathbf{W}_{\text{BPICA}}$ and $\mathbf{W}_{\text{ICA}}$. This line is a cross-section of the high-dimensional function that is the actual training loss. The most important points to note are (1) when $\alpha = 0$, which is exactly the loss of the BPICA weights $L(\mathbf{W}_{\text{BPICA}})$, and (2) when $\alpha = 1$, which is the loss of the ICA weights $L(\mathbf{W}_{\text{ICA}})$. The results are shown in Figure 3. For all three data sets—Cora, CiteSeer and Facebook—the loss obtained by BPICA $L(\mathbf{W}_{\text{BPICA}})$ appears to be a local minimum, the loss obtained by ICA training is not. Unsurprisingly, these results suggest that the training procedure of BPICA is a better strategy to optimize the loss function than using the true-label relational features, supporting the intuition and motivation behind our proposed approach.

5.4 Comparison of Prediction Accuracies

For many collective classification algorithms, the local features still play a significant role in how accurately they are able to predict. These methods often lack robustness to handle data that has weak local signal. In order to test whether BPICA is more robust to the loss of local signal, we weaken the local signal by randomly deleting subsets of local features and testing the learning algorithms on the deleted data. The results for these experiments are shown in Figures 4 to 6, where we plot the average accuracies for the best-scoring regularization parameters over 20 splits. We compared BPICA, ICA, GS, and the local classifier, which makes predictions only based on local features. The horizontal axis represents the fraction of deleted features, which ranges from 0 to 0.9, and the vertical axis represents the training or testing accuracies achieved by these algorithms.

The results for all three data sets suggest that BPICA is more robust to weak local signal than ICA, GS, and local classifier. For the Cora and CiteSeer data, as the fraction of deleted features increases, the training accuracies of BPICA stays stable until over 80% of local features have been deleted. The training accuracies of ICA, GS, and the local classifier drop earlier and much faster. When 90% of the features are deleted—meaning only 10% of the features can be used to train the model—the training accuracies of ICA, GS, and the local classifier drop to 0.5, however the accuracies of BPICA still remains around 0.9, showing that BPICA is able to train models to fully utilize the relational structure. The test accuracies of BPICA also show better performance than the other two methods as the number of local features reduces. Especially when over 60% local features are deleted, the local predictors become less reliable which causes ICA's accuracy

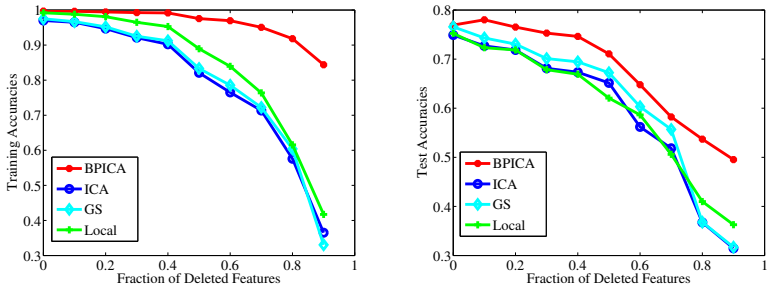


Figure 4: Performance of BPICA, ICA, GS and local classifier on Cora data. Each curve plots the average training or testing accuracy of the three different methods, using the best-scoring regularization parameters. The horizontal axis represents the fraction of local features that are removed. On training accuracy, BPICA dominates all other methods, demonstrating the effectiveness of directly optimizing the loss. On both training and testing accuracy, BPICA performs significantly better than the other methods when there is weak local signal.

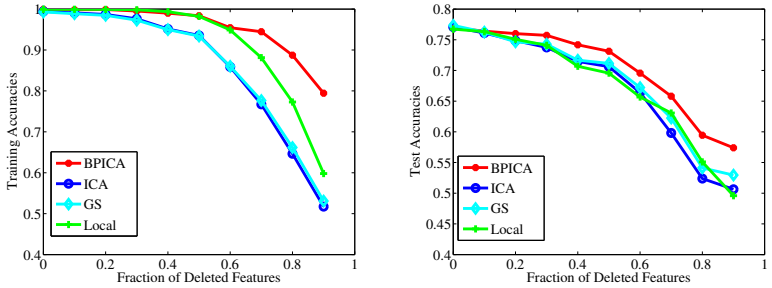


Figure 5: Performance of BPICA, ICA and local classifier on CiteSeer data. See Figure 4 for further discussion.

to significantly worsen, and BPICA is able to withstand the lack of attribute information. The Facebook results follow similar trends, where the training and test accuracies are always better than the ICA and local classifier. The differences in the testing accuracy between BPICA and ICA are statistically significant for all fractions of deleted features on the Cora tests, for 0.7 and 0.9 for the CiteSeer tests, and for all fractions 0.2 and higher on the Facebook tests.

One interesting effect not often reported in other research is the tendency for ICA trained using the true labels to produce predictors that perform *worse* than the local classifier, even on training data. This effect is exactly because of the discrepancy between the training regime and the actual prediction algorithm BPICA aims to correct. For example, in all three of our data sets, there are settings, especially when the local classifier is noisy, that ICA has worse training accuracy than the local classifier.

6. CONCLUSION AND DISCUSSION

We presented BPICA, a variant of the iterative classification algorithm that uses differentiable operations, enabling back-propagation of error gradients to directly optimize the model parameters. The concept of collective classification has long been understood to be a principled approach to classifying nodes in networks, but in practice, it often suffers from making only small improvements, or not improving at all. One cause for this could be the faulty training procedure that we correct in this paper. Our experiments demonstrate dramatic improvements in training accuracy, which translate to significant, but less dramatic improvements in testing performance. Thus, an important aspect to consider to further improve the effectiveness of collective classifiers is the generalization behavior of collective models. One future direction of research is exploring how a more direct training loss-minimization interacts with known generalization analyses, perhaps leading to further algorithm improvements. Another future direction we are exploring is how to apply similar approaches of direct loss minimization in transductive settings and how to expand the flexibility of the framework of BPICA to incorporate other variants of ICA.

References

[1] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54–65, 1994.

[2] M. Bilgic, G. M. Namata, and L. Getoor. Combining collective classification and link prediction. In *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on*, pages 381–386. IEEE, 2007.

[3] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *Neural Networks, IEEE Transactions on*, 5(2):240–254, 1994.

[4] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization.

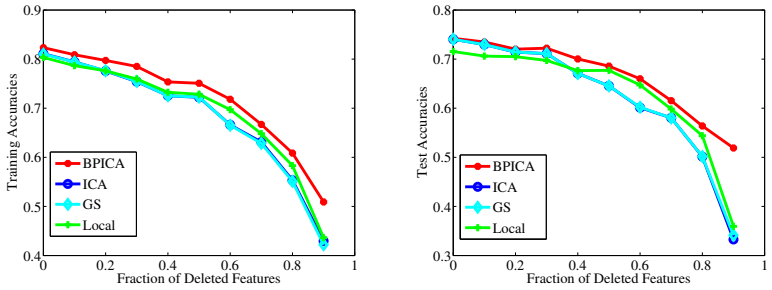


Figure 6: Performance of BPICA, ICA and local classifier on Facebook data. See Figure 4 for further discussion.

Journal of Machine Learning Research, pages 2122–2159, 2011.

[5] L. Getoor and C. P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[6] A. Graves. *Supervised sequence labelling*. Springer, 2012.

[7] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

[8] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[9] M. Ishikawa. Structural learning with forgetting. *Neural Networks*, 9(3):509–521, 1996.

[10] D. Jensen, J. Neville, and B. Gallagher. Why collective inference improves relational classification. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 593–598, 2004.

[11] X. Kong, X. Shi, and S. Y. Philip. Multi-label collective classification. In *SDM*, volume 11, pages 618–629. SIAM, 2011.

[12] J. Lindamood, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. Inferring private information using social network data. In *Proceedings of the 18th international conference on World wide web*, pages 1145–1146. ACM, 2009.

[13] B. London and L. Getoor. Collective classification of network data. In C. C. Aggarwal, editor, *Data Classification: Algorithms and Applications*. CRC Press, 2013.

[14] Q. Lu and L. Getoor. Link-based classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[15] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Neural Information Processing Systems*, 2012.

[16] L. K. McDowell, K. M. Gupta, and D. W. Aha. Cautious inference in collective classification. In *AAAI*, volume 7, pages 596–601, 2007.

[17] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, pages 415–444, 2001.

[18] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[19] J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.

[20] J. Neville and D. Jensen. Collective classification with relational dependency networks. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, pages 77–91. Citeseer, 2003.

[21] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[22] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *International Conference on Machine Learning (ICML)*, pages 129–136, 2011.

[23] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 485–492. Morgan Kaufmann Publishers Inc., 2002.

[24] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.