Simulated Iterative Classification A New Learning Procedure for Graph Labeling

Francis Maes, Stéphane Peters, Ludovic Denoyer, and Patrick Gallinari

LIP6 - University Pierre et Marie Curie
104 avenue du Président Kennedy, Paris, France

Abstract. Collective classification refers to the classification of interlinked and relational objects described as nodes in a graph. The Iterative Classification Algorithm (ICA) is a simple, efficient and widely used method to solve this problem. It is representative of a family of methods for which inference proceeds as an iterative process: at each step, nodes of the graph are classified according to the *current predicted labels* of their neighbors. We show that learning in this class of models suffers from a training bias. We propose a new family of methods, called Simulated ICA, which helps reducing this training bias by simulating inference during learning. Several variants of the method are introduced. They are both simple, efficient and scale well. Experiments performed on a series of 7 datasets show that the proposed methods outperform representative state-of-the-art algorithms while keeping a low complexity.

1 Introduction

A fundamental assumption that underlies most existing work in machine learning is that data is *independently and identically distributed* (*i.i.d.*). Web pages classification, WebSpam detection, community identification in social networks and peer-to-peer files analysis are typical applications where data is naturally organized according to a graph structure. In these applications, the elements to classify (Web pages or users of files for example) are interdependent: the label of one element may have a direct influence on other labels in the graph. Problems involving the classification of graph nodes are generally known as *graph labeling* problems [5] or as *collective classification* problems [11]. Due to the irrelevancy of the *i.i.d.* assumption, new models have been proposed recently to perform machine learning on such networked data.

Different variants of the graph labeling problem have been investigated. For *inductive graph labeling*, training and test are performed in distinct steps. The goal here is to learn classifiers able to label any node in new graphs or subgraphs. This is an extension of the classical *supervised classification task* to interdependent data. Typical applications include email classification, region or object labeling in images or sequence labeling. For *transductive graph labeling*, node labeling is performed in a single step where both labeled and unlabeled data are considered simultaneously. This corresponds to applications like WebSpam detection or social network analysis. Note that some problems like web

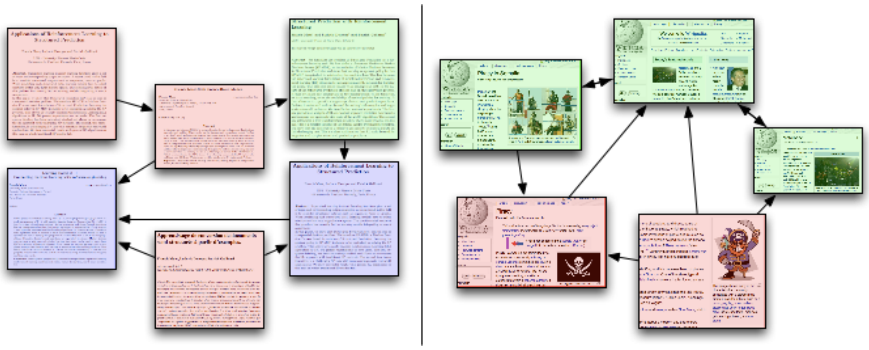

Fig. 1. Two examples of graph labeling problems. Left: categorization of scientific articles related by citation links. Right: classification of Web pages into relevant and non-relevant pages for a given query.

page classification may be handled under either the inductive or the transductive settings. In this article, we focus on the inductive graph labeling problem and we will use the name *collective classification* for this setting.

Graph labeling is often a hard problem and performing exact inference is generally prohibitive. Practical algorithms then rely on approximate inference methods. Many collective classification algorithms make use of local classifiers. Inference then amounts at iteratively labeling nodes: each iteration takes into account labels predicted at preceding steps. Several such local classifier techniques have been proposed [16,15]. Among them, the Iterative Classification Algorithm (ICA) has received a growing interest in the past years. It has been shown to be more robust and accurate than most alternative solutions, it is simple to understand and scales well *w.r.t.* the size of data, making it possible to label graphs containing thousands to millions of nodes.

Like most local collective classification methods, training and inference for ICA are performed differently. For the former, a local classifier is trained classically, using as inputs some node characteristics and its *correct* neighboring labels. Inference, on the opposite, is an iterative process, where the node labels are repeatedly re-estimated, given the *current estimated* labels of neighbors. Prediction errors on labels may then propagate during the iterations and the classifier will then have difficulties to generalize correctly. This is mainly caused by the bias between training – which assume perfect labels for neighboring nodes – and inference – which may iterate over wrong labels. In this paper, we build on this idea and introduce methods for reducing this training bias. The *Simulated Iterative Classification* (Sica) models proposed here introduce different ways to simulate the inference process during learning. The local classifier being trained on situations close to test ones, this will allow reducing the bias of classical ICA. We present different variants of this Sica algorithm. We also introduce the Sica$^+$ algorithm with *pass-dependent classifiers*, which relies on the idea of using a different local classifier for each iteration of the algorithm.

The underlying idea of this last model is similar to the one developed for the *Stacked Learning* method [8], however it proceeds differently, replacing the cross validation steps used for stacked learning by inference simulation. This family of techniques provides computationally efficient algorithms which outperform many other methods and which can be used for large scale graph labeling.

The contributions of this paper are twofold. Firstly, we propose a new collective classification algorithm. Its inference procedure is similar to ICA. Its learning procedure incorporates *inference simulation* avoiding the training bias of ICA. The Sica model has the same low inference complexity than ICA but provides higher performance. Several variants of the Sica method are introduced and compared. Secondly, we present a set of experiments on seven graph labeling datasets showing the efficiency of our approach.

The paper is structured as follows. Section 2 is an overview of related work. Section 3 defines the graph labeling problem formally and describes ICA. Section 4 introduces our contribution: *Simulated ICA*. We demonstrate the efficiency of the proposed approach with several experiments and comparisons with state-of-the-art models in Section 5. Finally, we conclude and discuss the generality of the proposed approach in Section 6.

2 Related Work

Two main directions have been developed independently for learning to label graphs.

The first one has been proposed for semi-supervised learning and is sometimes called *learning on data manifolds* [20]. The graph here reflects the intrinsic structure of the data and describes local constraints among data points [19]. Graph labeling is formalized as the minimization of an objective function over both labeled and unlabeled nodes. This objective function aims at finding a classifier corresponding to a good balance between an error minimization term defined on the labeled nodes and a local smoothness constraint which imposes close scores for connected nodes. All these methods rely on transductive learning. Different type of models have been developed along this idea:

– Models based on *label propogation* [21,2] only operate on the node labels without considering any other node feature. The labels are propagated iteratively from the labeled nodes to the rest of the graph. The models mainly differ by the regularization function they rely on.
– Models taking into account both the structure of the graph and the node features. Belkin et al. [3] developed a general framework allowing the use of both local node features and weighted links. Other models have been proposed by Zhang et al. [18] for web page classification. They have been adapted by Abernethy et al. [1] for the WebSpam detection task.

The second direction sometimes called collective or relational classification directly attacks the problem of graph labeling. It makes use of different models like ICA, Gibbs sampling, Relaxation Labeling and Loopy Belief Propagation (see

[16,15] for a review and a comparison of these methods). There are two main groups of methods:

- *Local classifier based methods* make use of a local classifier for classifying a node knowing both its content and the labels of its neighbors. For example, the *Iterative Classification* model [16] iteratively uses the base classifier during a fixed number of iterations. Gibbs sampling [9] aims at finding the best set of labels, by sampling each node label iteratively according to the probability distribution given by the local classifier.
- *Global models* try to optimize a global function over the graph. Since this is NP-hard, these methods propose different approximation algorithms in order to reduce the complexity of the optimization. The more popular methods are Loopy Belief Propagation [13] and Relaxation Labeling [10].

Note that methods of this second family are generally used in an inductive setting. They also suffer from the same training label bias as ICA since training and test operate differently. Finally, it is worth emphasizing Stacked Graphical Learning [8,12], a collective classification method developed independently. It makes use of a chain of classifiers, used iteratively to label the nodes. Each classifier uses as input the output of the preceding one. The originality of this algorithm comes from the way it learns the stacked classifiers using a cross-validation based approach. In the context of graph labeling, this algorithm was successfully used for WebSpam classification [6] or for layout document structuring [7].

Graph labeling problem is an instance of the more general framework of *structured prediction* or *structured output classification* [17]. In principle, any general structured prediction methods could be applicable to solve the graph labeling problem. However they have a high computational complexity and are not used for practical applications, especially on a large scale.

3 Supervised Graph Labeling

We use the following notations in the remainder of the paper. A directed graph is a couple $G = (X, E)$ where $X = (x_1, \dots, x_N)$ is a set of nodes and $E = \{(i,j)\}_{i,j \in [1..N]^2}$ is a set of directed edges. We denote (G, Y) the labeled graph G where $Y = (y_1, \dots, y_N)$ is the set of labels, with y_i the label corresponding to node x_i. Nodes in X are described through feature vectors $x_i \in \mathbb{R}^d$ where d is the number of features. We consider here the multiclass single-label problem: labels belong to a predefined set of possible labels $\mathcal{L} = (l_1, ..., l_L)$ where L is the number of possible labels[1].

For example, in a document classification task, nodes x_i may be documents described with vectors of word frequencies, edges (i, j) may correspond to citation links and labels y_i may be document categories.

We consider the inductive graph labeling problem. Given a training labeled directed graph (G, Y), the aim is to learn a model that is able to label the nodes

[1] Depending on the communities, labels may also be called *classes* or *categories* in the literature.

```
Input: A labeled graph $(G, Y)$
Input: A multiclass learning algorithm $\mathcal{A}$
Output: A base classifier $P(y_i|x_i, \mathcal{N}(x_i))$
foreach $x_i \in X$ do
    submit training example $((x_i, \mathcal{N}(x_i)), y_i)$ to $\mathcal{A}$
end
return the classifier trained by $\mathcal{A}$
```

Algorithm 1. ICA Learning algorithm

of any new graph G'. Note that this framework is rather general, since undirected graph can be seen as particular cases of directed graph and since G may contain multiple disconnected components, *i.e.* multiple different training graphs.

3.1 Iterative Classification

ICA is a graph labeling method based on iterative classification of graph nodes given both their local features and the labels of neighboring nodes. Let $\mathcal{N}(x_i)$ be the set of labels of neighbors of x_i. Typically, neighboring nodes are those directly connected to x_i, i.e:

$$\mathcal{N}(x_i) = \{y_j \mid (i,j) \in E\} \cup \{y_j \mid (j,i) \in E\}$$

ICA relies on the assumption that the probability $P(y_i = l|x_i, G)$ of a label can be approximated by the local probability $P(y_i|x_i, \mathcal{N}(x_i))$, which only depends on the associated content x_i and on the set of neighboring labels. Note that, since the number of neighboring labels $\mathcal{N}(x_i)$ is variable and depends on node x_i, neighbors information needs to be encoded as a fixed-length vector to enable the use of usual classification techniques. This is detailed and illustrated in Section 5.

Learning. The learning procedure of ICA is given in Algorithm 1. $P(y_i|x_i, \mathcal{N}(x_i))$ is estimated by a multiclass classifier. Possible multiclass base classifiers include neural networks, decision trees, naive Bayes or support vector machines. The base classifier is learned thanks to a learning algorithm $\mathcal{A}$ given a labeled training graph (G, Y). Learning the base classifier simply consists in creating one classification example per node in the graph and then running $\mathcal{A}$. In these classification examples, inputs are pairs $(x_i, \mathcal{N}(x_i))$ and outputs are correct labels $y_i \in \mathcal{L}$. Note that both batch learning algorithms and online learning algorithms may be used in conjunction with ICA.

Inference. ICA performs inference in two steps, as illustrated in Algorithm 2. The first step, *bootstrapping*, predicts an initial label for all the nodes of the graph. This step may either be performed by the base classifier – by removing neighboring labels information – or by another classifier trained to predict labels given the local features only. The second step is the *iterative classification* process

```
Input: A unlabeled graph $G = (V, E)$
Input: A classifier $P(y_i|x_i, \mathcal{N}(x_i))$
Output: The set of predicted labels $Y$

// Bootstrapping
foreach $x_i \in X$ do
    $y_i \leftarrow \text{argmax}_{l \in \mathcal{L}} P(y_i = l|x_i)$
end

// Iterative Classification
repeat
    Generate ordering $\mathcal{O}$ over nodes of $G$.
    foreach $i \in \mathcal{O}$ do
        $y_i \leftarrow \text{argmax}_{l \in \mathcal{L}} P(y_i = l|x_i, \mathcal{N}(x_i))$
    end
until all labels have stabilized or a threshold number of iterations have elapsed
return $Y$
```

Algorithm 2. ICA Inference algorithm

itself, which re-estimates the labels of each node several times, picking them in a random order and using the base classifier. ICA inference may converge exactly; if none of the labels change during an iteration, the algorithm stops. Otherwise, inference is usually stopped after a given number of iterations.

The main advantage of ICA is that both training and inference have linear complexities *w.r.t.* the number of nodes of the graphs. Furthermore, even if it is not guaranteed to converge, ICA has shown to behave well in practice and to give nice performance on a wide range of real-world problems [15].

4 Simulated ICA

When using ICA to infer the labels of a directed graph G, the base classifier is used repeatedly to predict the label of a node given its content and neighboring labels. Since it is very rare to reach perfect classification, the base classifier of ICA often makes some prediction errors during inference. Since prediction errors become inputs for later classification problems, ICA raises an important bias between training and inference; in the former case, neighboring labels are always correct, while they may be noisy in the latter case. This training/inference bias is illustrated in Figure 2.

The training/inference bias raised by ICA corresponds to a general problem that appears as soon as *predictions become inputs for later classification problems.* In the context of collective classification, the authors of Stacked Learning [8] identified the same training/inference bias. Both the Simulated ICA approach detailed below and Stacked Learning are motivated by the same concern: learning with intermediary predictions that are adapted towards the natural bias of the classifier.

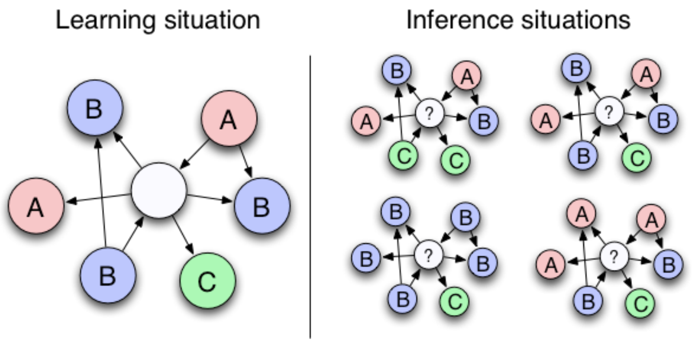

Fig. 2. Illustration of the learning/inference bias of ICA. Left: during training, only perfect neighboring labels are considered. Right: during inference, several neighboring labels may be wrong due to classification errors.

4.1 Learning Algorithm

In order to remove the learning/inference bias of ICA, the base classifier should be trained on situations representative of what happens during inference. In other words, the base classifier should be trained with corrupted neighboring labels. Furthermore, these labels should be biased towards the natural bias of inference: errors that are more frequent during inference should appear more during learning. Simulated ICA (Sica) relies on a simple, but powerful, idea to make training examples representative of inference behavior: *simulation*. We propose to simulate inference during learning, in order to create the training examples representative of the real use of the base classifier.

The general scheme of the Simulated ICA learning procedure is given by Algorithm 3. Sica is an iterative learning algorithm which repeatedly runs ICA inference, using labels which are *sampled* based on the currently learned classifier $P(y_i|x_i, \mathcal{N}(x_i))$. This sampling operation can be performed in different ways, which are detailed in Section 4.2. At each inference step, one training example is submitted to the learning algorithm $\mathcal{A}$, to train the base classifier for predicting the correct label y_i given the node x_i and the current neighboring labels. The key property of training examples in Sica is that they rely on *currently predicted neighboring labels* instead of assuming perfectly labeled neighbors, as it is the case of ICA.

Similarly to classical ICA, both batch and stochastic learning algorithms may be used inside Sica. In our experiments, we used stochastic descent learning algorithm to update the parameters of the base classifier.

4.2 Sampling Methods

When learning with Sica, the aim of simulation is to generate a maximum number of situations that are representative of ICA's inference behavior. Sica therefore relies on a *sampling* operation, whose role is to select the labels y_i used during learning. We propose three sampling strategies:

```
Input: A labeled graph $(G, Y)$
Input: A multiclass learning algorithm $\mathcal{A}$
Output: A classifier $P(y_i|x_i, \mathcal{N}(x_i))$
repeat
    // Bootstrapping
    foreach $x_i \in X$ do
        $y_i \leftarrow \text{argmax}_{l \in \mathcal{L}} P(y_i = l|x_i)$
    end
    // Iterative Classification
    repeat
        Generate ordering $\mathcal{O}$ over nodes of $G$.
        foreach $i \in \mathcal{O}$ do
            sample $y_i$ given $P(y_i = l|x_i, \mathcal{N}(x_i))$
            submit training example $((x_i, \mathcal{N}(x_i)), y_i)$ to $\mathcal{A}$
        end
    until iterative classification terminates
until learning has converged or a threshold number of iterations have elapsed
return the classifier trained by $\mathcal{A}$
```

Algorithm 3. Simulated ICA Learning algorithm

– SICA-DET (Deterministic). The simplest way to perform sampling in SICA consist in selecting the labels y_i that are *predicted* by the current classifier $P(y_i|x_i, \mathcal{N}(x_i))$. Formally, the sampling operation used in SICA-DET is defined as follows:

$$y_i = \underset{l \in \mathcal{L}}{\text{argmax}}\, P(y_i = l|x_i, \mathcal{N}(x_i))$$

– SICA-UNI (Uniform Noise). In order to increase the range of generated inference situations, one simple variant of SICA called SICA-UNI, consists in introducing stochasticity into the inference process by selecting labels randomly with a small probability. Formally, the sampling operation used in SICA-UNI is defined as follows:

$$y_i = \begin{cases} \text{a random label } l \in \mathcal{L} & \text{with probability } \epsilon \\ \text{argmax}_{l \in \mathcal{L}} P(y_i = l|x_i, \mathcal{N}(x_i)) & \text{with probability } 1 - \epsilon \end{cases}$$

where $\epsilon \in [0, 1]$ is a parameter controlling the tradeoff between random sampling and SICA-DET style sampling.

– SICA-PROB (Probabilistic). Instead of using uniform noise, a more natural alternative consists in sampling labels from the $P(y_i|x_i, \mathcal{N}(x_i))$ distribution:

$$y_i = \text{sample } l \in \mathcal{L} \text{ from } P(y_i = l|x_i, \mathcal{N}(x_i))$$

The advantages of SICA-PROB and SICA-UNI over SICA-DET are twofold. Firstly, stochasticity enables to generate a wider range of inference situations and thus creates more training examples for the base classifier. This may thus lead to more robust classifiers, especially when using few training nodes. Secondly, stochasticity may contribute to make training conditions closer to test conditions. In this sense, simulating the inference procedure with additional noise is an alternative to the cross-validation approach of Stacked Learning. In both cases, the aim is to learn with intermediary predictions that correctly reflect the behavior of the model on testing data. Stacked Learning creates the intermediary predictions by cross-validation, while we propose to directly modify the predicted labels on training data. We show in Section 5 that Sica-Prob and Sica-Uni frequently outperform Stacked Learning experimentally.

4.3 One Classifier Per Pass

ICA operates in several passes, where each node of the graph is re-estimated during one pass. Since the problem of first estimating the labels may slightly differ from the problem of re-estimating the labels at the second pass or at the third pass, the current pass number may have a direct influence on the base classifier. Instead of learning a unique classifier $P(y_i|x_i, \mathcal{N}(x_i))$, Sica can be modified to take the current pass number, t, into account, by learning a classifier $P(y_i|x_i, \mathcal{N}(x_i), t)$.

Sica-Det, Sica-Uni and Sica-Prob can be modified by learning *one distinct classifier per pass*. This leads to three new variants of Sica that are denoted Sica$^+$-Det, Sica$^+$-Uni and Sica$^+$-Prob in the remainder of this paper. As an example, our experiments use Sica with 5 maximum inference passes, which leads to a set of 5 slightly different classifiers, each one specialized for a given inference pass. Concretely, using one classifier per pass makes it possible to learn finer inference-dependent behavior. This idea is similar to stacking: in both Sica$^+$ and Stacked Learning, there is one classifier per inference pass and each of these classifiers is trained to compensate the errors of previous classifiers.

Sica and Sica$^+$ both perform learning and inference in the same way. The only difference concerns the number of parameters which is multiplied by the number of inference passes by using one disctinct classifier per pass.

5 Experiments

In this section, we describe experimental comparisons between Simulated ICA and various state-of-the-art models for inductive graph labeling.

5.1 Datasets

We performed experiments on four datasets of webpages and on three datasets of scientific articles, whose characteristics are summarized in Table 1. In the former datasets, nodes correspond to webpages, links represent web hyperlinks and the aim is to predict the category of each webpage. In the latter datasets, nodes are scientific articles, links are citation links and the aim is to predict the

Table 1. This table shows different charasteristics (numbers of nodes, links, features and classes) of the seven different datasets used in our experiments. The four first datasets are small scale graphs from WebKB, the two following ones are medium scale graphs and the last one is our large scale dataset.

Dataset	**Nodes**	**Links**	**Features**	**Classes**
		Small scale – WebKB		
Cornell	195	304	1703	5
Texas	187	328	1703	5
Washington	230	446	1703	5
Wisconsin	265	530	1703	5
		Medium scale		
CiteSeer	3312	4715	3703	6
Cora-I	2708	5429	1433	7
		Large scale		
Cora-II	36954	136024	11816	11

category of each article. All nodes are described by feature vectors, with features that correspond to the most frequent words and that indicate whether or not the associated words appear. Cora-II was introduced by A. McCallum and was preprocessed by keeping only the words appearing in at least 10 documents[2]. The other datasets were preprocessed by Getoor et al[3].

5.2 Experimental Setup

State-of-the-art models. In order to evaluate Simulated ICA, we have implemented three state-of-the-art graph labeling models: Iterative Classification (Ica), Gibbs Sampling (Gs) and Stacked Learning. Our implementation of Stacked Learning uses 5-fold cross-validation during learning to create the intermediary predictions. Stack2 is the simplest Stacked Learning model, where the labels are first estimated based on the content only and are then re-estimated by taking both the initial predictions and the content into account. Stack3 is a Stacked Learning model with three stacks: labels are first estimated using Stack2 and are then re-estimated with a classifier that uses both the content and the predictions of Stack2. Similarly, Stack4 and Stack5 are Stacked Learning models that respectively rely on Stack3 and Stack4 to provide intermediary predictions.

Baselines. We have also compared Simulated ICA with two baselines named *Content-Only* (Co) and *Optimistic* (Opt). The former is a classifier ignoring the graph structure and taking only the content of nodes into account during classification. The latter is a classifier which assumes the availability of perfect neighboring labels for each node, during both *training and inference*. Note that

[2] Cora-II is available at `http://www.cs.umass.edu/~mccallum/code-data.html`

[3] Datasets available at `http://www.cs.umd.edu/~sen/lbc-proj/LBC.html`, see [14] for more details.

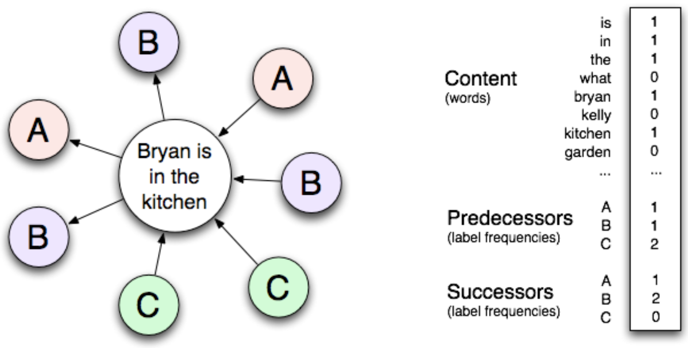

Fig. 3. A feature function to jointly describe content of nodes and neighboring labels. Feature vectors contain three parts. The first one describes the content of the node and the other two describe the labels of the node neighbors. For each label, there is a feature which counts the number of predecessor (*resp.* successors) with this label.

due to this dependency, the OPT baseline is not a "true model" able to generalize to new unlabeled graphs.

Base classifier. In order to make the comparison fair, we used the *same base classifier*, the *same learning algorithm* and the *same tuning procedure* for all models. The base classifiers are L2-regularized maximum-entropy classifiers [4] learned with stochastic gradient descent. For each model, we tried ten different regularizer values and kept the best-performing ones.

In order to take simultaneously the content of nodes and their neighboring labels, the base classifiers rely on a feature function that jointly maps contents and associated neighboring labels to scalar vectors. The feature function used in our experiments is illustrated in Figure 3.

Data splitting. In order to evaluate the generalization abilities of the models, we have split each dataset into one training graph and one testing graph. As [16], we have used a random sampling strategy: both the training and the testing graphs are random subsets of the whole graphs. When splitting graphs randomly, it is often the case that some links connect train nodes to test nodes. The simplest way to deal with these *train-test links* is simply to ignore them; this approach is called *Test Links Only*. Since many links may be discarded with the *Test Links Only* approach, we have also adopted the alternative approach proposed by [14] called *Complete Links* in which no link is suppressed and in which we assume, during inference, to know the correct labels of training nodes connected to testing nodes. For the purpose of comparison with [14], we used 10-folds cross-validation with the *Complete Links* strategy. Note that, if we used 10-folds with *Test Links Only*, 90% of the links involving testing nodes would be discarded. In order to make the average number of links connected to testing nodes and to training nodes equal, we used *Test Links Only* with two-fold cross-validation.

Table 2. This table shows the accuracy of different models on Cora-I, CiteSeer and the four WebKB databases. The graphs were split randomly into two folds by using *Test Links Only* strategy. We used 50% of the dataset as a training set and took the mean test accuracy over ten runs.

	Small scale – WebKB				Medium scale	
	Cornell	Texas	Washington	Wisconsin	Cora-I	CiteSeer
Co	71.79	82.14	80.52	85.21	74.73	71.73
Ica	72.91	82.46	81.22	84.76	78.69	73.14
Gs	73.02	82.67	81.04	84.98	78.67	72.8
Stack2	73.02	82.35	81.74	84.38	78.85	73.25
Stack3	73.02	82.46	81.13	84.76	79.73	73.34
Stack4	73.02	82.67	81.3	84.83	79.53	73.25
Stack5	72.91	82.35	81.13	84.76	79.81	73.22
Sica-Det	73.53	82.24	81.83	85.21	78.77	72.92
Sica-Uni	**74.24**	83.21	82.35	85.36	79.29	73.21
Sica-Prob	74.04	**83.42**	**82.43**	**85.74**	79.32	73.47
Sica$^+$-Det	71.48	80.32	80	83.32	79.18	73.59
Sica$^+$-Uni	72.71	82.35	81.13	84.45	79.59	73.70
Sica$^+$-Prob	73.42	82.35	81.91	85.06	**80.01**	**74.02**
Opt	72.5	82.78	82.17	84.46	83.16	76.18

5.3 Results

In our experiments, the parameters was the same as these in [16] for the baselines (Co, Opt, Ica, Gs and Stack): the maximum number of training iterations was set to 100, for Iterative Classification we used a maximum of 100 inference passes and for Gibbs Sampling we performed 1000 samples of each node label. For the Sica inference, we used a maximum of 5 passes. Moreover, we fixed the uniform noise percentage in Sica-Uni and Sica$^+$-Uni to 10%.

Comparisons with state-of-the-art models. We compared the six variants of Sica described in Section 4 with the baselines described previously on the small and medium scale datasets. Firstly, we have split each dataset into two halves by using the *Test Links Only* strategy. Each experiment was performed 10 times with different random splits and we report the mean test accuracies in Table 2. As expected, Sica models outperform Ica on all datasets. More interestingly, our models also outperform Gibbs Sampling and Stacked Learning in nearly all cases. In particular, the Sica-Uni and Sica-Prob models that rely on stochasticity outperform Stacked Learning on five datasets out of six, whereas this tendency is less clear for Sica-Det. As discussed in Section 4, adding a perturbation to the predicted labels proves to be a good alternative to the heavy cross-validation based prediction process adopted by Stacked Learning. Among the two perturbation approaches, Sica-Prob most-of-time reaches better results than Sica-Uni. A deeper comparison of these sampling approaches in given below. Using one classifier per pass (Sica$^+$-Det, Sica$^+$-Uni and Sica$^+$-Prob)

Table 3. This table shows the accuracy of different models on Cora-I, CiteSeer and the four WebKB databases. Here, the graphs were split randomly into ten folds by using *Complete Links* strategy. Then, we did a 10-folds cross-validation and took the mean test accuracy.

	Small scale – WebKB				Medium scale	
	Cornell	Texas	Washington	Wisconsin	Cora-I	CiteSeer
Co	79.5	86.64	84.35	89.4	77.43	72.98
Ica	79.47	87.22	85.65	89.79	88.52	77.63
Gs	80.5	87.22	85.22	89.79	88.18	77.47
Stack2	78.92	**89.91**	87.39	89.42	88.07	76.72
Stack3	78.42	88.27	**88.26**	89.03	88.18	77.35
Stack4	78.97	88.3	86.96	89.8	88.4	77.23
Stack5	78.45	88.83	87.83	89.42	88.4	77.08
Sica-Det	**81.55**	87.75	85.65	89.79	88.37	76.27
Sica-Uni	**81.55**	88.27	85.65	89.79	88.26	76.48
Sica-Prob	81.53	87.75	86.52	**90.16**	88.37	76.33
Sica$^+$-Det	79.5	86.7	84.78	89.42	**88.74**	77.75
Sica$^+$-Uni	80.03	86.70	86.52	89.42	88.63	77.93
Sica$^+$-Prob	81.05	87.22	85.65	89.79	88.66	**78.02**
Opt	79.97	87.75	86.09	89.77	88.85	78.08

improves over the basic versions of Sica on the two medium scale datasets (+ 0.69% on Cora-I and + 0.55% on CiteSeer). On the small datasets, using one classifier per pass slightly deteriorates the results. We believe that this is due to estimation problems related to the large number of parameters on these approaches.

The second set of experiments aims at comparing our results to those of [16]. Here, each dataset was split into ten folds by using the *Complete Links* strategy. Table 3 gives the 10-fold cross-validation accuracies (90% training nodes and 10% testing nodes) for all models and all datasets. As previously, our models most-of-the-time outperform Ica and Gs. On small datasets, Stacked Learning is competitive with our models. However, these results should be taken with a grain of salt, since, when using 90% training nodes with *Complete Links*, a large majority of testing-node neighbors are in the training set. Consequently, the various methods that take wrong labels into account during learning (Sica and Stacked Learning) have a more limited interest in this case.

Impact of the uniform noise percentage. Next, we evaluated the impact of uniform noise percentage on Sica-Uni. Figure 4 compares the three sampling methods on our two medium scale datasets. With a reasonable noise percentage (below 40%), the accuracy of Sica-Uni is between that of Sica-Det and that of Sica-Prob. Once more, this confirms the contribution of stochasticity in the simulation process of Sica.

In most cases, Sica-Prob outperforms Sica-Uni. Sica-Prob has another key advantage over the latter: it does not rely on additional hyper-parameter,

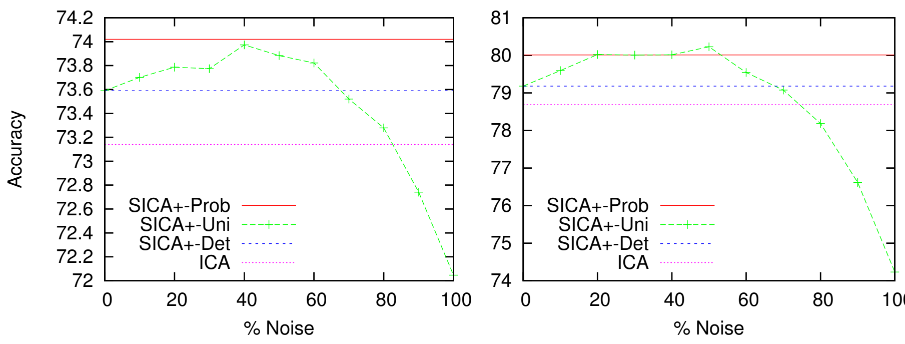

Fig. 4. Accuracy for varying percentage of uniform noise in SICA-UNI on CITESEER (on the left) and CORA-I (on the right)

Table 4. Test accuracies and training/inference time on CORA-II. The first two columns give the mean test accuracy of the models with the *Test Links Only* and *Complete Links* splitting strategies. The last two columns show approximate training time and inference time. The results for STACK5 are incomplete due to the excessive time requirement of this method (since we average our results over 10 runs and try 10 different regularizer values, the experiment would need more than three months to complete).

	Test Links Only	Complete Links	Training time	Inference time
CO	49	49	2 min	300 ms
ICA	51.9	58.05	6 min	20 s
GS	44.43	55.42	6 min	10 min
STACK2	54.46	56.28	13 min	1 s
STACK3	56.07	57.99	1 h	1.5 s
STACK4	**56.67**	58.52	4.5 h	2 s
STACK5	-	-	20 h	2.5 s
SICA-DET	56.02	58.52	5 min	4 s
SICA-UNI	56.20	**59.57**	3 min	4 s
SICA-PROB	56.3	59.14	3 min	4 s
SICA$^+$-DET	55.5	58.87	5 min	4 s
SICA$^+$-UNI	56.25	59.56	3 min	4 s
SICA$^+$-PROB	56.2	58.91	2 min	4 s
OPT	66.40	67.71	3.5 min	600 ms

which makes its tuning much easier. With a too high noise percentage, information in neighboring labels becomes irrelevant and thus accuracy drops down to the one of a *Content-Only* classifier.

Large scale. In order to show that our model can deal with large-scale graphs, we performed a set of experiments on the CORA-II database. For each model, we performed 10 runs with 20% training nodes and 80% training nodes selected randomly. The mean test accuracies are reported in Table 4. The first two columns give scores respectively for *Test Links Only* and *Complete Links*. The last two columns give the CPU time needed to train a single model and the time needed to fully label the test graph[4]. Experiments were performed on standard 3.2 Ghz computer.

Our approaches clearly outperform Ica and Gs on Cora-II (up to +3% improvement) and behave similarly to Stacked Learning with a much lower training time. Indeed, since each stack involves making 5 folds and learning a model on each sub-fold recursively, Stack models have a training time which is exponential *w.r.t.* the number of stacks. Instead, all our models – that use 5 inference passes – were learned in a few minutes.

6 Conclusion

In this paper, we have introduced the Simulated Iterative Classification Algorithm (Sica), a new learning algorithm for ICA. The core idea of Sica is to simulate ICA's inference during learning. We argued that simulation is a simple and efficient way to create training examples that are representative of *real inference situations*. We have shown that the proposed approach outperforms state-of-the-art models (Iterative Classification, Gibbs Sampling and Stacked Learning) on a wide range of datasets. Furthermore, we have shown that the model scales well, which makes it possible to label graphs containing thousands to millions of nodes.

Our future work will primarily focus on generalizing the idea of simulation during learning to semi-supervised graph labeling problems. We believe that one promising approach is to develop (fast and scalable) incremental inference algorithms, that takes both the labeled and the unlabeled nodes into account, and to learn them using simulation.

One key characteristic of ICA is that it relies on a classifier whose inputs depend on its previous predictions. Although this paper is focused on supervised graph labeling problems, we believe that the proposed idea of simulating inference during learning is relevant to a wider class of problems where predictions become inputs for later classification problems. In order to tackle error propagation problems in such algorithms involving classifier chains, simulation is a key solution, which appears to be both simple and efficient.

Acknowledgement

We would like to thank the French National Research Agency (ANR) for financial support under the project *MADSPAM 2.0*.

[4] Note that Sica models were carefully optimized in our implementation, which explains their relative low training times.

References

1. Abernethy, J., Chapelle, O., Castillo, C.: Witch: A new approach to web spam detection. Technical report, Yahoo! Research (2008)
2. Agarwal, S.: Ranking on graph data. In: ICML 2006, pp. 25–32. ACM, New York (2006)
3. Belkin, M., Niyogi, P., Sindhwani, V.: Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. Journal of Machine Learning Research 7, 2399–2434 (2006)
4. Berger, A.L., Pietra, S.D., Della Pietra, V.J.: A maximum entropy approach to natural language processing. Computational Linguistics 22(1), 39–71 (1996)
5. Castillo, C., Davison, B.D., Denoyer, L., Gallinari, P. (eds.): Proceedings of the Graph Labelling Workshop and Web Spam Challenge (2007)
6. Castillo, C., Donato, D., Gionis, A., Murdock, V., Silvestri, F.: Know your neighbors: web spam detection using the web topology. In: SIGIR 2007, pp. 423–430. ACM, New York (2007)
7. Chidlovskii, B., Lecerf, L.: Stacked dependency networks for layout document structuring. In: SAC, pp. 424–428 (2008)
8. Cohen, W.W., de Carvalho, V.R.: Stacked sequential learning. In: IJCAI, pp. 671–676 (2005)
9. Hastings, W.K.: Monte carlo sampling methods using markov chains and their applications. Biometrika 57(1), 97–109 (1970)
10. Hummel, R.A., Zucker, S.W.: On the foundations of relaxation labeling processes, pp. 585–605 (1987)
11. Jensen, D., Neville, J., Gallagher, B.: Why collective inference improves relational classification. In: ACM SIGKDD 2004, pp. 593–598. ACM, New York (2004)
12. Kou, Z., Cohen, W.W.: Stacked graphical models for efficient inference in markov random fields. In: SDM (2007)
13. Kschischang, F.R., Frey, B.J.: Iterative decoding of compound codes by probability propagation in graphical models. IEEE Journal on Selected Areas in Communications 16, 219–230 (1998)
14. Lu, Q., Getoor, L.: Link-based classification using labeled and unlabeled data. In: ICML: Workshop from Labeled to Unlabeled Data (2003)
15. Macskassy, S.A., Provost, F.: Classification in networked data: A toolkit and a univariate case study. J. Mach. Learn. Res. 8, 935–983 (2007)
16. Sen, P., Namata, G.M., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. Technical Report CS-TR-4905, University of Maryland, College Park (2008)
17. Taskar, B., Chatalbashev, V., Koller, D., Guestrin, C.: Learning structured prediction models: A large margin approach. In: ICML 2005, Bonn, Germany (2005)
18. Zhang, T., Popescul, A., Dom, B.: Linear prediction models with graph regularization for web-page categorization. In: KDD 2006: Proceedings of the 12th ACM SIGKDD, pp. 821–826. ACM, New York (2006)
19. Zhou, D., Schölkopf, B.: Regularization on discrete spaces. In: Kropatsch, W.G., Sablatnig, R., Hanbury, A. (eds.) DAGM 2005. LNCS, vol. 3663, pp. 361–368. Springer, Heidelberg (2005)
20. Zhou, D., Schölkopf, B., Hofmann, T.: Semi-supervised learning on directed graphs. In: NIPS, pp. 1633–1640. MIT Press, Cambridge (2005)
21. Zhu, X., Ghahramani, Z.: Learning from labeled and unlabeled data with label propagation. Technical report (2002)