Meta-Prediction for Collective Classification

Luke K. McDowell¹, Kalyan Moy Gupta², and David W. Aha³

¹Dept. of Computer Science; U.S. Naval Academy; Annapolis, MD 21402
²Knexus Research Corp.; Springfield, VA 22153
³Navy Center for Applied Research in Artificial Intelligence;
  Naval Research Laboratory (Code 5514); Washington, DC 20375
lmedowel@usna.edu | kalyan.gupta@knexusresearch.com | david.aha@nrl.navy.mil

Abstract
When data instances are inter-related, as are nodes in a social network or hyperlink graph, algorithms for collective classification (CC) can significantly improve accuracy. Recently, an algorithm for CC named Cautious ICA (ICAC) was shown to improve accuracy compared to the popular ICA algorithm. ICAC improves performance by initially favoring its more confident predictions during collective inference. In this paper, we introduce ICAMC, a new algorithm that outperforms ICAC when the attributes that describe each node are not highly predictive. ICAMC learns a meta-classifier that identifies which node label predictions are most likely to be correct. We show that this approach significantly increases accuracy on a range of real and synthetic data sets. We also describe new features for the meta-classifier and demonstrate that a simple search can identify an effective feature set that increases accuracy.

Introduction
In many classification tasks, the instances to be classified (such as web pages or people in a social network) are related in some way. Collective classification (CC) is a methodology that jointly classifies such instances (or nodes). CC algorithms can attain higher accuracies than non-collective methods when nodes are interrelated (Neville and Jensen 2000; Taskar, Abbeel, and Koller 2002). Several CC algorithms have been studied, including relaxation labeling (Chakrabarti, Dom, and Indyk 1998), the Iterative Classification Algorithm (ICA) (Sen et al. 2008), loopy belief propagation (LBP) (Taskar et al. 2002), and Gibbs sampling (Jensen, Neville, and Gallagher 2004).

We focus on ICA because it is a popular and computationally efficient algorithm that has good classification performance (Sen et al. 2008). It makes initial label predictions for each node vi, then iteratively re-computes them based on the predictions for every node that links to vi. Recently, a variant of ICA named Cautious ICA (ICAC) (McDowell et al. 2007, 2009) was shown to often attain higher accuracies than ICA. ICAC is based on the observation that, since some label predictions will be incorrect, ICA’s use of all predictions may sometimes decrease accuracy. To counter this effect, ICAC initially uses only some label predictions. By “cautiously” choosing only those predictions that appear more likely to be correct, ICAC can increase accuracy vs. ICA.

In this paper, we introduce Meta-Cautious ICA (ICAMC), which is exactly like ICAC except in how it selects the set of predicted labels to use during classification. In particular, ICAMC learns a meta-classifier to predict the likelihood that a label prediction is correct. By carefully constructing a meta-training set from the original training set, ICAMC can learn this classifier and use it to select more reliable predicted labels than ICAC, increasing accuracy.

Our contributions are as follows. First, we present ICAMC, a novel algorithm that can significantly increase accuracy compared to ICAC, especially when the attributes that describe each node are not very predictive. Second, we introduce a technique to improve accuracy by generating more training examples for ICAMC’s meta-classifier. Third, we describe new features for the meta-classifier and demonstrate that, while the most effective meta-features for ICAMC are task-dependent, a simple search identifies an effective set that increases accuracy. Empirical evaluations using real and synthetic datasets support our claims.

We next review CC and the ICA and ICAC algorithms. Then we introduce ICAMC. Finally, we present our experimental evaluation and discuss future research issues.

Collective Classification
Assume we are given a graph G = (V,E,X,Y,C), where V is a set of nodes, E is a set of (possibly directed) edges, each x_i ∈ X is an attribute for node v_i ∈ V, each Y_i ∈ Y is a label variable for v_i, and C is the set of possible labels. We are also given a set of “known” label values Y^K for nodes V^K ⊆ V, so that Y^K = {Y_i | v_i ∈ V^K}. Finally, assume that we are given a training graph G Tr, which is defined similarly to G except that every node in G Tr is a “known” node. Then the task is to infer Y_i = Y_i^K for the nodes in G whose labels are unknown. For each node v_i, let y_i be the true label and ̄y_i be the predicted label.
For example, consider the task of predicting whether a web page belongs to a professor or a student. Conventional supervised learning approaches ignore the links and classify each page using attributes derived from its content (e.g., words present in the page). In contrast, a technique for relational classification explicitly uses the links to construct additional features for classification (e.g., for each page, include as features the words from hyperlinked pages). These relational features can increase classification accuracy, though not always (Chakrabarti et al. 1998). Alternatively, even greater (and usually more reliable) increases can occur when the class labels of the linked pages are used to derive relevant relational features (Jensen et al. 2004). However, using features based on these labels is challenging because some or all of these labels are initially unknown. Thus, their labels must first be predicted (without using relational features) and then re-predicted in some manner (using all features). This process of jointly inferring the labels of interrelated nodes is known as collective classification (CC).

We next describe two existing collective inference algorithms, ICA and ICAC, and then introduce ICAMC. Each algorithm relies on a given node classifier \( \Phi \), which predicts the class label \( \hat{y}_i \) for each node \( v_i \) using both attributes and relations.

### ICA: Inference using all predicted labels

Figure 1 shows pseudocode for ICA, ICAC, and ICAMC (depending on \( \text{AlgType} \)). Step 1 is a “bootstrap” step that predicts the class label \( \hat{y}_i \) for each node in \( V \) using only attributes \( (\text{conf}_i, \text{rfi}_i) \) records the confidence of this prediction, but ICA does not use it). ICA then iterates (step 2). During each iteration, it selects all available label predictions (step 3), computes the relational features’ values based on them (step 4), and then re-predicts the class label of each node using both attributes and relational features (step 5). Step 6 is ignored for ICA. After iterating, step 7 returns the final set of predicted class labels and their confidence values.

### ICAC: Inference using some predicted labels

In steps 3-4 of Figure 1, ICA assumes that the predicted node labels are all equally likely to be correct. When \( \text{AlgType} = \text{ICAC} \) is instead ICAC, the inference becomes more cautious by only considering more confident predictions. Specifically, step 3 “commits” into \( Y \) only the most confident \( m \) of the currently predicted labels; other labels are considered missing and are ignored. Step 4 computes the relational features using only the committed labels, and step 5 performs classification using this information. Step 3 gradually increases the fraction of predicted labels that are committed per iteration. Node label assignments committed in an iteration \( h \) are not necessarily committed again in future iterations (and may in fact change).

ICAC requires a confidence measure \( \text{conf}_i \) (Figure 1) to rank the current label predictions. As with prior work (Neville and Jensen 2000, McDowell et al. 2007), we set \( \text{conf}_i \) to be the posterior probability of the most likely class.

### Improving ICAC with Meta-Caution

To address this potential problem with ICAC, we created ICAMC. They are identical except that ICAMC uses a separate “meta classifier” to predict how likely each prediction \( \hat{y}_i \) is to be correct. Below we describe ICAMC’s use of this meta-classifier, methods for generating its training data, and methods for constructing its features.

### ICAMC: Inference using predicted correct labels

Figure 1 shows that ICAMC changes ICAC only in step 6. In particular, after using the node classifier to predict the label \( \hat{y}_i \) (and associated confidence \( \text{conf}_i \)) for every node, ICAMC computes the meta-feature values and then uses the meta-classifier \( \Phi_{\text{mfi}} \) to predict how likely \( \hat{y}_i \) is to be correct. These predictions serve as the new confidence values that are then used in Step 3 of the next iteration to select the...
committed set \( Y' \). If the meta-classifier’s confidence predictions more accurately identify those nodes whose labels are correctly predicted (compared to ICAC’s simple confidence values), then accuracy should increase.

**Generating meta-training data**

Learning the meta-classifier requires constructing appropriate meta-training data, which we represent as a set of vectors. Figure 2 shows the pseudocode for this task, whose algorithm employs a holdout graph (a subset of the training set) with nodes \( V \), edges \( E \), attributes \( X \), and true labels \( Y \). For each of \( T \) trials, step 3 randomly selects \( lp\% \) of the nodes to be known; this value is chosen to replicate the fraction of known labels that are present in the test set. It then executes ICAC on the graph, given the known nodes (step 4). This yields the set \( Z'' \), which contains the label predictions and associated confidence values for each node in \( V'' \). Using these and the expected class distribution \( \Phi \) (from the training set), it then generates a meta-training vector per node (steps 5-7). This vector includes eight meta-features (described later) and a Boolean value that indicates whether prediction \( \hat{y}_i \) is correct. This training data is later used to learn the meta-classifier that predicts the correctness of the \( \hat{y}_i \) estimates given the values of the meta-features.

We set \( T=10 \) to conduct ten trials with different known nodes each time. The goal is to reduce the bias that might otherwise occur due to the particular selection of \( Y'' \) in step 3. We later compare this with the one-trial approach (\( T=1 \)).

**ICA\(_{MC}\) needs useful meta-features to predict when the node classifier has correctly classified a node. The constructed features are based on two key premises. First, we assume that the data exhibits relational autocorrelation (correlation of class labels among interrelated nodes, Jensen et al., 2004) for use by the node classifier. Thus, each node’s predicted label will be influenced by the predictions of its neighboring labels. Second, since ICA\(_{MC}\) (like ICAC) exploits only some of the predicted labels during each iteration, not all neighbor labels will affect the prediction for \( v_i \). We assume that the accuracy of prediction \( \hat{y}_i \) for iteration \( j \) is affected only by the neighbors of \( v_i \) that were included in the committed set \( Y' \) during that same iteration. Let \( N_i \) refer to the set of such neighbors for \( v_i \).

Based on these two premises and additional intuitions described below, we designed eight features for this initial study of ICA\(_{MC}\). The first three features are based on ones used by Biligic and Getoor (2008) for a related problem that is discussed later. Future work should examine these choices and others in more detail.

Suppose the CC algorithm predicts \( \hat{y}_i \) to be the label for node \( v_i \), with confidence \( conf \). Then \( v_i \)’s features are:

1. **Local score**: The CC algorithm’s predictions should differ from those of an attribute-only classifier (e.g., \( M_{A} \) in Figure 1), or there is no point in executing CC. However, if \( M_{A} \) and the node classifier \( M_{AR} \) agree on a prediction, then it is more likely to be correct. This heuristic is captured by using, for each \( v_i \), \( M_{A} \)’s confidence value for the \( \hat{y}_i \) that was predicted by \( M_{AR} \). “Known” nodes are assumed to be fully correct (score of 1), though this could be reduced to account for possible noise:

   \[
   \text{lf}_i = \begin{cases} 
   \frac{1}{N_i} \sum_{v_j \in N_i} \text{lf}_j, & v_i \notin V'' \\
   1, & v_i \in V''
   \end{cases}
   \]

2. **Relational score**: If a node is surrounded by nodes whose predictions are more likely (e.g., have high \( \text{lf} \) scores), then its prediction is also more likely:

   \[
   \text{rf}_i = \frac{1}{|N_i|} \sum_{v_j \in N_i} \text{lf}_j
   \]

3. **Global score**: Let \( \text{Prior}(c) \) be the fraction of training nodes with class label \( c \), and \( \text{Posterior}(c) \) be the fraction of test set labels predicted as \( c \) by the CC algorithm. If \( \text{Posterior}(c) \) is much higher than \( \text{Prior}(c) \), then many nodes with predicted label \( c \) may be incorrect. Thus, the global score measures whether class \( y_i \) is over or underestimated in the posterior distribution:

   \[
   \text{gf}_i = \frac{1}{2} \left( \frac{1}{2} - \frac{\text{Prior}(\hat{y}_i) - \text{Posterior}(\hat{y}_i)}{\text{Prior}(\hat{y}_i) + \text{Posterior}(\hat{y}_i)} \right)
   \]

4. **Node confidence**: If the node classifier is confident in some prediction \( \hat{y}_i \) (high posterior probability), then this suggests that \( \hat{y}_i \) is more likely to be correct:

   \[
   \text{cf}_i = \begin{cases} 
   \text{conf}, & v_i \notin V'' \\
   1, & v_i \in V''
   \end{cases}
   \]

   If only this feature is used, ICA\(_{MC}\) devolves to ICAC.

5. **Neighbor confidence**: As with the relational score, more confident neighbor predictions suggest that a node’s prediction is more likely to be correct:

   \[
   \text{nf}_i = \frac{1}{|N_i|} \sum_{v_j \in N_i} \text{cf}_j
   \]

**Figure 2**: Pseudocode to generate training vectors for the meta classifier used by ICA\(_{MC}\).
6. Neighbor agreement: If most of node \( v_i \)’s neighbors have the same predicted label, this may indicate that \( \hat{y}_i \) is more likely to be correct. Let \( \text{count}_1(N_i) \) and \( \text{count}_2(N_i) \) indicate the count of the two most frequent label predictions in \( N_i \). If the former value is large and the latter is small, then neighbor agreement is high:

\[
naf_i = \frac{1}{|N_i|} \left( \text{count}_1(N_i) - \text{count}_2(N_i) \right)
\]

7. Known neighbors: Having many “known” neighbors increases the chances that a node’s prediction is correct:

\[
\text{known neighbors} = \left| N_i \cap V^K \right| + \frac{1}{2} \left| \forall v_j \exists v \in (N_i \cap N_j) \right|
\]

Each of these eight features may not be useful for every dataset. However, ICA\textsuperscript{MC} needs only some of the features to be useful – the meta-classifier (we use logistic regression) will learn appropriate parameters for each feature based on their predictive accuracy on the meta-training data. Also, features that provide no benefit are discarded by the feature search process described later.

### Evaluation

**Hypotheses.** By default, ICA\textsuperscript{MC} uses feature search and ten randomized training data trials. This ICA\textsuperscript{MC} attains higher accuracies than ICA\textsuperscript{C} (Hypothesis #1), ICA\textsuperscript{MC} without such trials (#2), ICA\textsuperscript{MC} without feature search (#3), and ICA\textsuperscript{MC} with just the three features used by Bilgic and Getoor (#4).

**Data Sets.** We used the following data sets (see Table 1):

1. **Cora** (see Sen et al. 2008): A collection of machine learning papers categorized into seven classes.
2. **CiteSeer** (see Sen et al. 2008): A collection of research papers drawn from the CiteSeer collection.
3. **WebKB** (see Neville and Jensen 2007): A collection of web pages from four computer science departments.
4. **Synthetic:** We generate synthetic data using Sen et al.’s (2008) graph generator. Similar to their defaults, we use a degree of homophily of 0.7 and a link density of 0.4.

### Table 1: Data sets summary

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>WebKB</th>
<th>Syn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total nodes</td>
<td>2708</td>
<td>3312</td>
<td>1541</td>
<td>n.a.</td>
</tr>
<tr>
<td>Avg. # nodes per test set</td>
<td>400</td>
<td>400</td>
<td>385</td>
<td>250</td>
</tr>
<tr>
<td>Avg. links per node</td>
<td>2.7</td>
<td>2.7</td>
<td>6</td>
<td>3.3</td>
</tr>
<tr>
<td>Class labels</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Non-rel. features avail.</td>
<td>1433</td>
<td>3703</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>Non-rel. features used</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Relational features used</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Folds</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>

**Feature Representation.** Our node representation includes relational features and non-relational attributes, as described below.

**Non-relational (content) attributes:** The real datasets are all textual. We use a bag-of-words representation for the textual content of each node, where the feature corresponding to a word is assigned true if it occurs in the node and false otherwise.

Our version of the WebKB dataset has 100 words available. For Cora and CiteSeer, we used information gain to select the 100 highest-scoring words, based on McDowell et al. (2007), which reported that using more did not improve performance. Our focus is on the case where relatively few attributes are available (or the attributes are not very predictive) as may occur in large real-world networks (cf., Macskassy and Provost 2007, Gallagher et al. 2008). Thus, for most of our experiments we randomly select 10 of the 100 available words to use as attributes. We also briefly discuss results when using 100 attributes. For the synthetic data, ten binary attributes are generated using the technique described by McDowell et al. (2009). This model has a parameter \( ap \) (attribute predictiveness) that ranges from 0.0 to 1.0; it indicates how strongly predictive the attributes are of the class label. We evaluate \( ap \) using the values \( \{0.2, 0.4, 0.6\} \).

**Relational features:** Each relational feature value is a multiset. For instance, a possible feature value is \( \{3 A, 2 B, 1 C, 1 missing\} \), which indicates that a node links to 3 other nodes whose predicted label is A, 2 nodes whose prediction is B, and 1 node labeled missing. During inference, each label in the multiset (excluding missing labels) is separately used to update the probability that a node has label c. This is the “independent value” approach that was introduced by Neville et al. (2003), used by Neville and Jensen (2007), and shown to be superior to “count” or “proportion” features by McDowell et al. (2009). See Neville et al. (2003) for more details.

For Cora and CiteSeer, we compute a “multiset” feature using only incoming links, and a separate such feature using only outgoing links. For WebKB, we also compute one such feature using “co-citation” links (a co-citation link exists between nodes \( i \) and \( j \) if some node \( k \) links to both of them). For the synthetic data, the links are undirected, so there is a single relational feature.

**Classifiers.** For the node classifier, we used a naïve Bayes classifier. McDowell et al. (2009) reported that, using multiset features, it attained higher accuracies than did alternatives such as logistic regression. For the meta-classifier, we used logistic regression, as did Bilgic and Getoor (2008). Future work should consider other choices.

**Test Procedure.** We conducted an n-fold cross-validation study for each tested algorithm. For WebKB, we treated each of the four schools as a separate fold. For Cora and CiteSeer, we created five disjoint test sets by using “similarity-driven snowball sampling” (McDowell et al. 2009). This is similar to the approach of Sen et al. (2008).
For all 3 datasets we tested on one graph, trained on two others, and used the remaining two (one for WebKB) as a holdout set for learning the meta-classifier and performing the meta-feature search.

For the synthetic data, we performed 25 separate trials. For each trial we generated three disjoint graphs: one test set, one training set, and one holdout set.

We randomly selected \( p=10\% \) of each test set to form \( V^k \) (nodes with known labels). This is a “sparsely labeled” task, which is common in real data (Gallagher et al. 2008).

To search for which of the eight meta-features to use with ICAMC, we use the simple, greedy Backwards Sequential Elimination (BSE) algorithm (Kittler, 1986). It evaluates accuracy on the holdout set with ICAMC, then recursively eliminates any meta-feature whose removal increases accuracy. To increase robustness, accuracy is averaged over ten executions of ICAMC, each time using a different set of initial “known” labels (as done for \( T=10 \) in Figure 2). The final set of meta-features is used for testing.

**Tested Algorithms.** We tested ICA, ICAC, and ICAMC. In addition, to assess the utility of ICAMC’s design decisions, we also tested three of its ablated variants:

1. “1 trial instead of 10”: this uses only one randomized trial to collect meta-training data (i.e., \( T=1 \) in Figure 2) and only one evaluation trial for the meta-feature search.
2. “No meta-feature search”: This skips search and uses all eight meta-features that were previously described.
3. “Only Bilgic meta-feats”: This uses just features #1, #2, and #3—the set used by Bilgic and Getoor (2008).

**Performance Measure.** We compared all the algorithms on their average classification error rate on the test sets.

**Analysis.** We performed independent analyses for each prediction task and joint analyses by pooling the observations, either for all the real data sets or for all the synthetic data sets. Our analysis uses one-tailed paired t-tests accepted at the 95\% confidence level.

**Results.** Table 2 displays the classification error rates averaged over all the folds for each algorithm. For each (data set, algorithm) pair, the best result is shown in bold.

**Table 2: Average % classification error rate**

<table>
<thead>
<tr>
<th>Core Algorithms</th>
<th>“Real” datasets</th>
<th>Synthetic data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cora CS Web KB</td>
<td>ap=0.2 ap=0.4  ap=0.6</td>
</tr>
<tr>
<td>ICA</td>
<td>51.5† 61.0† 60.3†</td>
<td>53.3† 35.9† 22.6†</td>
</tr>
<tr>
<td>ICAC</td>
<td>36.2† 37.6† 32.5†</td>
<td>38.8† 27.8† 18.3†</td>
</tr>
<tr>
<td>ICAMC</td>
<td>31.3 35.3 24.5</td>
<td>31.9 25.0 16.4</td>
</tr>
<tr>
<td>Gain*</td>
<td>4.9 2.3 8.0</td>
<td>6.9 2.8 1.9</td>
</tr>
</tbody>
</table>

**Variants of ICAMC**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 trial instead of 10</td>
<td>No meta-feat. search</td>
</tr>
<tr>
<td></td>
<td>35.4† 35.8 30.0†</td>
<td>35.9† 37.6† 31.5†</td>
</tr>
<tr>
<td></td>
<td>36.4† 27.5† 17.2</td>
<td>33.5 24.9 16.2</td>
</tr>
<tr>
<td>Only Bilgic meta-feats</td>
<td>42.1† 47.1† 26.4</td>
<td>37.3† 27.8† 18.2†</td>
</tr>
</tbody>
</table>

* indicates significantly worse behavior than ICAMC. † indicates gain from meta-caution (ICAC – ICAMC).

For the real data, ICA MC provides gains for all three datasets, where the largest gain is with WebKB. WebKB has more complex and numerous linking patterns (Macskassy and Provost 2007). For this reason, ICAMC’s careful selection of which neighboring labels to use for prediction may be especially important with WebKB.

We repeated these experiments with real data using 2, 5, or 20 attributes (instead of 10) and found similar results. In every case pooled analyses found a significant gain for ICAMC over ICAC (average gains ranging from 3.2–6.9\%), with the largest gains occurring with WebKB. As with the synthetic data, these gains diminish when the attributes are more predictive. For instance, when 100 attributes are used the gains of ICAMC remained but were small (0.2-1.0\%) and statistically insignificant. These results suggest that ICAMC is especially helpful when the attributes alone are not very predictive, and at least does no harm otherwise.

**Result 2: ICAMC with randomized trials and meta-feature search outperforms simpler variants**

The bottom of Table 2 shows results with the variants of ICAMC that do not use multiple randomized trials or do less or no meta-feature search. ICAMC outperforms the “1 trial instead of 10” and “Only Bilgic meta-feats” variants, often significantly, and pooled analyses find that ICAMC outperforms both, for the real and for the synthetic data. Thus, we accept Hypotheses #2 and #4. ICAMC also significantly outperforms the variant that uses all eight meta-features (“No meta-feat. search”) for the real data, but not for the synthetic data (perhaps because simpler, undirected linking patterns were used in the synthetic data). Thus, we reject Hypothesis #3.

Despite the rejection of one hypothesis, ICAMC always outperformed all three variants (or lagged by at most 0.2\%) and significantly outperformed all three variants on the real datasets. Some of the variants that simplify ICAMC’s search process sometimes performed notably worse than even ICAC. Together, these results suggest that the complete ICAMC, with randomized trials and feature search, is the most robust performer.
Discussion

ICA\textsubscript{MC} increased accuracy compared to ICA and ICA\textsubscript{C}. However, why does ICA\textsubscript{MC}’s meta-classifier more effectively identify reliable predictions than does ICA\textsubscript{C}’s node classifier? First, the meta-classifier’s task is simpler: choosing between two values (correct or incorrect) vs. between all possible class labels. Second, the meta-classifier can use additional information, such as the number of known labels, which has no obvious utility for predicting a particular label, but does help estimate the correctness of the resultant prediction. Finally, using two different classifiers helps to reduce the bias due to using the Naïve Bayes node classifier alone.

Meta-feature search often significantly increased ICA\textsubscript{MC}’s accuracy. However, is the same set of features almost always chosen? On average, the “global score” and “node confidence” features were selected most often, and “known neighbor” least often. This varied substantially, however, with some features selected 90% of the time for one dataset and never for another. These results, combined with the results from Table 2, suggest that search is essential to make ICA\textsubscript{MC} robust across different data, even if the default set of meta-features is further refined.

We are not aware of any other work that uses a meta-classifier to improve the operation of a CC inference algorithm, although Bilgic and Getoor (2008) did use a similar predictor to identify the least likely CC label predictions (in order to “purchase” the correct labels for them). In contrast, we seek the most likely predictions (to favor them for inference). They considered three features for this different task, which our search algorithm selected for ICA\textsubscript{MC} 62%, 67%, and 91% of the time, respectively. Thus, their features are also useful for our task, although the results of the previous section show that using only those features leads to very poor performance for ICA\textsubscript{MC}.

Compared to ICA\textsubscript{C}, ICA\textsubscript{MC} requires additional computation: to execute ICA\textsubscript{C} when collecting meta-training data, to execute ICA\textsubscript{MC} for feature selection, and to train the meta-classifier for each combination of meta-features that are considered. However, in many real-world graphs each node links to at most \( k \) other nodes, in which case each of these steps is linear in the number of nodes. In addition, once the meta-classifier is learned, ICA\textsubscript{MC} requires little additional time for inference compared to ICA\textsubscript{C} (i.e., it needs only one additional execution of the meta-classifier per iteration).

Conclusion

We demonstrated that Meta-Cautious ICA (ICA\textsubscript{MC}) significantly outperforms ICA\textsubscript{C} for many tasks. Moreover, we showed that aspects of ICA\textsubscript{MC} – in particular, its use of multiple randomized training data trials and its use of search for selecting meta-features – were essential to achieving performance that was robust across a range of datasets. Since ICA\textsubscript{C} has already been shown to be a very effective CC algorithm, these results suggest that ICA\textsubscript{MC} should be seriously considered for CC applications, particularly when attributes alone do not yield high predictive accuracy.

Further work is needed to confirm our results using other datasets, meta-features, and classifiers, and to consider how meta-caution might be extended to other CC algorithms. In addition, we intend to consider techniques for further reducing the time complexity of ICA\textsubscript{MC} compared to ICA\textsubscript{C}.

Acknowledgements

We thank the Naval Research Laboratory for supporting this work, and Brian Gallagher, Gita Sukthankar, and the anonymous reviewers for their helpful comments.

References


