Reverse Engineering Gene Regulatory Networks Using Sampling and Boosting Techniques

Turki Turki[1,2(✉)] and Jason T.L. Wang[2(✉)]

[1] Department of Computer Science, King Abdulaziz University, P.O. Box 80221, Jeddah 21589, Saudi Arabia
tturki@kau.edu.sa

[2] Bioinformatics Program and Department of Computer Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA
{ttt2,wangj}@njit.edu

Abstract. Reverse engineering gene regulatory networks (GRNs), also known as network inference, refers to the process of reconstructing GRNs from gene expression data. Biologists model a GRN as a directed graph in which nodes represent genes and links show regulatory relationships between the genes. By predicting the links to infer a GRN, biologists can gain a better understanding of regulatory circuits and functional elements in cells. Existing supervised GRN inference methods work by building a feature-based classifier from gene expression data and using the classifier to predict the links in GRNs. Observing that GRNs are sparse graphs with few links between nodes, we propose here to use under-sampling, over-sampling and boosting techniques to enhance the prediction performance. Experimental results on different datasets demonstrate the good performance of the proposed approach and its superiority over the existing methods.

Keywords: Graph mining · Sampling methods · Boosting techniques · Supervised learning · Applications in biology and medicine

1 Introduction

Gene regulation is a series of processes that control gene expression and its extent. The connections among genes and their regulatory molecules, usually transcription factors, and a descriptive model of such connections, are known as gene regulatory networks (GRNs). Elucidating GRNs is crucial to understand the inner workings of the cell and the interactions among genes. Furthermore, GRNs could be the basis to infer more complex networks, encompassing gene, protein, and metabolic spaces, as well as the entangled and often over-looked signaling pathways that interconnect them [2,3,12,13,16].

Existing GRN inference methods can be broadly categorized into two groups: unsupervised and supervised [23]. Unsupervised methods infer GRNs based solely on gene expression data. The accuracy of these methods is usually low. By contrast, supervised methods use machine learning algorithms and training data

P. Perner (Ed.): MLDM 2017, LNAI 10358, pp. 63–77, 2017.
DOI: 10.1007/978-3-319-62416-7_5

to achieve higher accuracy. These methods work as follows. We represent a GRN by a directed graph in which each node is a gene or transcription factor, and a directed link or edge from node A to node B indicates that gene A regulates the expression of gene B. The training data contains a partially known network with known present edges and absent edges between nodes. These known present edges are positive training examples, and the known absent edges are negative training examples. We train a machine learning algorithm using the training data and apply the trained model to predict the remaining unknown edges in the network. With the predicted present and absent edges, we are able to infer or construct a complete GRN.

GRNs are always sparse graphs. The ratio between the number of gene interactions (i.e., edges or links) and the number of genes (i.e., nodes) falls between 1.5 and 2.75 regardless of the differences in phylogeny, phenotypic complexity, life history, and the total number of genes in an organism [14]. Thus, all GRNs have relatively few present edges and a lot of absent edges. This means there are few positive examples and a lot of negative examples when modeling the GRN inference problem as the link prediction problem described above. This poses an imbalanced classification problem in which the positive class (i.e., the minority class) is much smaller than the negative class (i.e., the majority class). However, existing supervised GRN inference methods [9,17] do not take into consideration the imbalanced datasets, and hence their performance is unsatisfactory.

In this paper, we present a new approach to supervised GRN inference. We tackle the imbalanced classification problem by using sampling techniques, including under-sampling and over-sampling, to obtain a balanced training set. This balanced training set, containing the same number of positive and negative training examples, is used to train a machine learning algorithm to make predictions. Furthermore, we develop several boosting techniques to enhance the prediction performance. As our experimental results show later, this new approach outperforms the existing supervised GRN inference methods [9,17].

The rest of this paper is organized as follows. Section 2 transforms the GRN inference problem to a link prediction problem in which we infer a GRN by predicting the links in the GRN. We then present several sampling and boosting techniques for enhancing the link prediction performance. Section 3 reports experimental results, comparing our approach with the existing methods [9,17]. Section 4 concludes the paper and points out some directions for future research.

2 Methods

2.1 Problem Statement

We are given n genes where each gene has p expression values. The gene expression profile of these n genes is denoted by $G \subseteq R^{n \times p}$, which contains n rows, each row corresponding to a gene, and p columns, each column corresponding to an expression value [9,17,18,20,22]. In addition, we are given known regulatory relationships or links among some genes. Suppose these known regulatory relationships are stored in a matrix $X \subseteq R^{m \times 3}$, which forms the training dataset.

X contains m rows, where each row shows a known regulatory relationship between two genes, and three columns. The first column shows a transcription factor (TF). The second column shows a target gene. The third column shows the label, which is $+1$ if the TF is known to regulate the expression of the target gene or -1 if the TF is known not to regulate the expression of the target gene. The matrix X represents a partially observed or known gene regulatory network for the n genes. If the label of a row in X is $+1$, then the TF in that row regulates the expression of the target gene in that row, and hence that row represents a directed link or edge of the network. That row is a positive training example. If the label of a row in X is -1, then there is no link between the corresponding TF and target gene in that row. That row is a negative training example. The positive and negative training examples in X are used to train a machine learning or classification algorithm. There are much more negative training examples than positive training examples in X.

The test dataset contains ordered pairs of genes (g_1, g_2) where the regulatory relationship between g_1 and g_2 is unknown. Given a test example, i.e., an ordered pair of genes (g_1, g_2) in the test dataset, the goal of *link prediction* is to use the trained classifier to predict the label of the test example. The predicted label is either $+1$ (i.e., a directed link is predicted to be present from g_1 to g_2) or -1 (i.e., a directed link is predicted to be absent from g_1 to g_2). Here, the present link means g_1 (a transcription factor) regulates the expression of g_2 (a target gene) whereas the absent link means g_1 does not regulate the expression of g_2.

2.2 Feature Vector Construction

To perform training and prediction, we construct a feature matrix $D \subseteq R^{q \times 2p}$ with q feature vectors based on the gene expression profile G. Let g_1 and g_2 be two genes. Let $g_1^1, g_1^2, ..., g_1^p$ be the gene expression values of g_1 and $g_2^1, g_2^2, ..., g_2^p$ be the gene expression values of g_2. The feature vector of the ordered pair of genes (g_1, g_2), denoted D_d, is stored in the feature matrix D and constructed by concatenating their gene expression values as follows:

$$D_d = (g_1^1, g_1^2, ..., g_1^p, g_2^1, g_2^2, ..., g_2^p) \tag{1}$$

Thus, the ordered pair of genes (g_1, g_2) corresponds to a point in $2p$-dimensional space. Each training and test example is represented by a $2p$-dimensional feature vector. For a positive training example, the label of its feature vector is $+1$. For a negative training example, the label of its feature vector is -1. For a test example, the label of its feature vector is unknown and to be predicted. This feature vector construction method has been widely used by existing supervised GRN inference methods [9,17,18,20,22].

2.3 Under-Sampling

Given is a training dataset X that is the union of two disjoint subsets X_+ and X_-. X_+ is the minority class, containing positive training examples (i.e., known

present links). X_- is the majority class, containing negative training examples (i.e., known absent links). X_+ is much smaller than X_-. Our under-sampling method works as follows [10,11,15]. It samples a random subset $X_-^s \subseteq X_-$ such that the size of X_-^s is equal to the size of X_+ (i.e., $|X_-^s| = |X_+|$). Thus, $X^s = X_+ \cup X_-^s$ forms a balanced dataset. We then use X^s to train a machine learning algorithm. The trained model will be used to predict the labels of test examples.

2.4 Over-Sampling

Our over-sampling method is based on SMOTE [4,6,10]. Given is the training dataset $X = X_+ \cup X_-$ as described above. Our over-sampling method creates a new dataset X_{++} that contains all examples in X_+ and many synthetic examples generated as follows. For each example $x_i \in X_+$, we select h-nearest neighbors of x_i, where

$$h = \left\lfloor \frac{1.1 \times |X_-|}{|X_+|} \right\rfloor \tag{2}$$

Here, Euclidean distances are calculated to find the h-nearest neighbors. Denote these h-nearest neighbors as x_r, $1 \leq r \leq h$.

A new synthetic example x_{new} along the line between x_i and x_r, $1 \leq r \leq h$, is created as follows:

$$x_{new} = x_i + (x_r - x_i) \times \delta \tag{3}$$

where $\delta \in (0, 1)$ is a random number. We add x_{new} to X_{++}. We continue generating and adding such synthetic examples to X_{++} until X_{++} is larger than X_-. Then a random subset $X_{++}^s \subseteq X_{++}$ is selected such that the size of X_{++}^s is equal to the size of X_- (i.e., $|X_{++}^s| = |X_-|$). Thus, $X^s = X_{++}^s \cup X_-$ forms a balanced dataset. We then use X^s to train a machine learning algorithm. The trained model will be used to predict the labels of test examples.

2.5 Boosting

We further improve the performance of our link prediction algorithms through boosting. Boosting algorithms such as AdaBoost [8,21,25,26], described below, have been used in various domains with great success. We use a weighted decision tree [1] as the base learning algorithm and create a strong classifier through an iterative procedure as follows. Let X be the set of training examples $\{x_1, x_2, ..., x_m\}$. The label associated with example x_i is y_i such that

$$y_i = \begin{cases} +1 \text{ if } x_i \text{ is a positive example (i.e., present link)} \\ -1 \text{ if } x_i \text{ is a negative example (i.e., absent link)} \end{cases} \tag{4}$$

Initially, in iteration 1, each example is assigned an equal weight, i.e., $W_1(x_i) = \frac{1}{m}$, $1 \leq i \leq m$. In iteration k, $1 \leq k \leq K$, AdaBoost generates a base learner (i.e., model) H_k by calling the base learning algorithm on the training set X with weights W_k. Then H_k is used to classify each training example x_i as either

+1 (i.e., x_i is a predicted present link) or −1 (i.e., x_i is a predicted absent link). That is,

$$H_k(x_i) = \begin{cases} +1 \text{ if } H_k \text{ classifies } x_i \text{ as a positive example (i.e., present link)} \\ -1 \text{ if } H_k \text{ classifies } x_i \text{ as a negative example (i.e., absent link)} \end{cases} \quad (5)$$

Let $E_k = \{x_i | H_k(x_i) \neq y_i\}$. The error ε_k of H_k is:

$$\varepsilon_k = \sum_{x_i \in E_k} W_k(x_i) \quad (6)$$

The weight α_k of H_k is:

$$\alpha_k = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_k}{\varepsilon_k} \right) \quad (7)$$

AdaBoost then updates the weight of each training example x_i, $1 \leq i \leq m$, as follows:

$$\begin{aligned} W_{k+1}(x_i) &= \begin{cases} \frac{W_k(x_i)}{Z_k} \times e^{-\alpha_k} & \text{if } H_k(x_i) = y_i \\ \frac{W_k(x_i)}{Z_k} \times e^{\alpha_k} & \text{if } H_k(x_i) \neq y_i \end{cases} \\ &= \frac{W_k(x_i) \exp(-\alpha_k y_i H_k(x_i))}{Z_k} \end{aligned} \quad (8)$$

where Z_k is a normalization factor chosen so that W_{k+1} is normally distributed. The weights of incorrectly classified examples will increase in iteration $k + 1$. Then, in iteration $k + 1$, AdaBoost generates a base learner H_{k+1} by calling the base learning algorithm again on the training set X with weights W_{k+1}. Such a process is repeated K times. Using this technique, each weak classifier H_{k+1} should have greater accuracy than its predecessor H_k. The final, strong classifier H is derived by combining the votes of the weighted weak classifiers H_k, $1 \leq k \leq K$, where the weight α_k of a weak classifier H_k is calculated as shown in Eq. (7).

Specifically, given an unlabeled test example $\hat{x}$, $H(\hat{x})$ is calculated as follows:

$$H(\hat{x}) = sign\Big(\sum_{k=1}^{K} \alpha_k H_k(\hat{x}) \Big) \quad (9)$$

The *sign* function indicates that if the sum of the results of the weighted K weak classifiers is greater than or equal to zero, then H classifies $\hat{x}$ as +1 (i.e., $\hat{x}$ is a predicted present link); otherwise H classifies $\hat{x}$ as −1 (i.e., $\hat{x}$ is a predicted absent link).

We propose to extend AdaBoost by modifying Eq. (8) to obtain the following variants:

Boost I:

$$\begin{aligned} W_{k+1}(x_i) &= \begin{cases} \frac{e^{-\alpha_k}}{Z_k} & \text{if } H_k(x_i) = y_i \\ \frac{e^{\alpha_k}}{Z_k} & \text{if } H_k(x_i) \neq y_i \end{cases} \\ &= \frac{\exp(-\alpha_k y_i H_k(x_i))}{Z_k} \end{aligned} \quad (10)$$

Boost II:

$$W_{k+1}(x_i) = \frac{\exp\Big(- \Big(\sum_{j=1}^{k} \alpha_j H_j(x_i) \Big) y_i \Big)}{Z_k} \tag{11}$$

Boost III:

$$W_{k+1}(x_i) = \begin{cases} \frac{C \times e^{\alpha_k}}{Z_k} & \text{if } H_k(x_i) = -1 \text{ and } y_i = +1 \\ \frac{e^{-\alpha_k}}{Z_k} & \text{if } H_k(x_i) = +1 \text{ and } y_i = +1 \\ \frac{e^{\alpha_k}}{Z_k} & \text{if } H_k(x_i) = +1 \text{ and } y_i = -1 \\ \frac{e^{-\alpha_k}}{Z_k} & \text{if } H_k(x_i) = -1 \text{ and } y_i = -1 \end{cases} \tag{12}$$

where C is the number of examples in the majority class (i.e., negative class) divided by the number of examples in the minority class (i.e., positive class) in the training set.

Each one of the above variants is taken as a new boosting technique. Boost I is a simplified version of AdaBoost. For each training example x_i, $1 \leq i \leq m$, Boost I does not consider $W_k(x_i)$ when calculating $W_{k+1}(x_i)$. Boost II is an accumulative version of AdaBoost. It considers all weak classifiers H_j, $1 \leq j \leq k$, obtained in the previous k iterations when calculating the weight $W_{k+1}(x_i)$. Specifically, Boost II will increase the weight of a training example x_i in iteration $k+1$ if the majority of the weak classifiers obtained in the previous k iterations incorrectly classify x_i. Boost III can be regarded as a cost-sensitive boosting technique. For an imbalanced dataset, positive examples (i.e., those with labels of $+1$) are much fewer than negative examples (i.e., those with labels of -1). Our objective here is to improve the classification performance on the minority (i.e., positive) class. Hence we introduce the cost C, giving more weights to misclassified examples in the minority class where the examples are classified as negative though they have labels of $+1$. For a training example x_i that is correctly classified in iteration k, we decrease its weight in iteration $k+1$ so that the next classifier H_{k+1} pays less attention to x_i while focusing more on the other examples that are incorrectly classified in iteration k.

2.6 The Proposed Approach

Figure 1 presents an overview of our approach. In (A), we are given a training set containing imbalanced labeled links. These labeled links include few positive examples (i.e., known present links with labels of $+1$) and a lot of negative examples (i.e., known absent links with labels of -1). In addition, we are given a test set in which each test example is an unlabeled ordered gene pair. We construct feature vectors for both training examples and test examples as described in Sect. 2.2. In (B), we apply a sampling technique, either under-sampling as described in Sect. 2.3 or over-sampling as described in Sect. 2.4, to the training set to obtain a balanced training set. In (C), we apply a boosting technique as described in Sect. 2.5 to the balanced training set to learn K models (weak classifiers). These models predict the labels of the test examples. In (D), we take

the weighted majority vote from the weak classifiers as shown in Equation (9) to make final predictions of the labels of the test examples. A test example is a predicted present link if its predicted label is +1; a test example is a predicted absent link if its predicted label is −1.

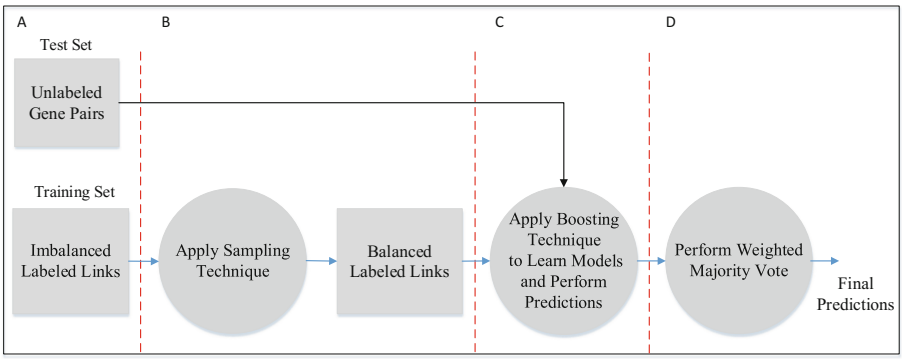

Fig. 1. The proposed approach for link prediction in gene regulatory networks.

3 Experiments and Results

3.1 Datasets

We used GeneNetWeaver [19] to generate the datasets related to yeast and E. coli. GeneNetWeaver has been widely used to generate benchmark datasets for evaluating GRN inference tools. Specifically we built four different networks for each organism where the networks contained 50, 100, 150, 200 genes (or nodes) respectively. Table 1 presents details of the yeast networks, showing the number of nodes (edges, respectively) in each network. The present edges or links in a network form positive examples. The absent edges or links in a network form negative examples. Table 2 presents details of the E. coli networks.

Table 1. Yeast networks used in the experiments.

Network	Directed	#Nodes	#Edges	#Positive examples	#Negative examples
Yeast 50	Yes	50	63	63	2387
Yeast 100	Yes	100	281	281	9619
Yeast 150	Yes	150	333	333	22017
Yeast 200	Yes	200	517	517	39283

For each network, we generated three files of gene expression data. These files were labeled as knockouts, knockdowns and multifactorial, respectively.

Table 2. E. coli networks used in the experiments.

Network	Directed	#Nodes	#Edges	#Positive examples	#Negative examples
E. coli 50	Yes	50	68	68	2382
E. coli 100	Yes	100	177	177	9723
E. coli 150	Yes	150	270	270	22080
E. coli 200	Yes	200	415	415	39385

A knockout is a technique to deactivate the expression of a gene, which is simulated by setting the transcription rate of this gene to zero [9]. A knockdown is a technique to reduce the expression of a gene, which is simulated by reducing the transcription rate of this gene by half [9]. Multifactorial perturbations are simulated by randomly increasing or decreasing the activation of the genes in a network simultaneously [9]. Totally there were twelve gene expression datasets for yeast and E. coli respectively.

3.2 Experimental Methodology

We compared our proposed approach with existing supervised GRN inference methods [9,17]. The existing methods employ support vector machines (SVM) and use the same feature vector construction method as ours (cf. Sect. 2.2); however, they lack sampling and boosting techniques. Table 3 lists the abbreviations of the fifteen algorithms that we evaluated and compared in this study where twelve algorithms are boosting-related and three algorithms are SVM-related.

The performance of each algorithm was evaluated through 10-fold cross validation. The positive examples (negative examples, respectively) were evenly distributed to the ten folds. When testing a fold, the Area Under the ROC Curve (AUC) of an algorithm was calculated where the AUC is defined as

$$\text{AUC} = \frac{1}{2} \times \left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} \right) \tag{13}$$

Here TP (FP, TN, FN, respectively) denotes the number of true positives (false positives, true negatives, false negatives, respectively) for the test set. A true positive (true negative, respectively) is a predicted present link (a predicted absent link, respectively) that is indeed a known present link (a known absent link, respectively). A false positive (false negative, respectively) is a predicted present link (a predicted absent link, respectively) that is in fact a known absent link (a known present link, respectively). For each algorithm, the mean AUC, denoted MAUC, over the ten folds was computed and recorded. The higher MAUC an algorithm has, the better performance that algorithm achieves.

3.3 Experimental Results

We first conducted experiments to evaluate the performance of SVM with different kernel functions, including the linear kernel, polynomial kernel of degree 2,

Table 3. Abbreviations of the fifteen algorithms studied in this paper.

Abbreviation	Algorithm
AdaBoost	AdaBoost technique
AdaBoost+U	AdaBoost with under-sampling technique
AdaBoost+O	AdaBoost with over-sampling technique
Boost I	Boost I technique
Boost I+U	Boost I with under-sampling technique
Boost I+O	Boost I with over-sampling technique
Boost II	Boost II technique
Boost II+U	Boost II with under-sampling technique
Boost II+O	Boost II with over-sampling technique
Boost III	Boost III technique
Boost III+U	Boost III with under-sampling technique
Boost III+O	Boost III with over-sampling technique
SVM	SVM technique
SVM+U	SVM with under-sampling technique
SVM+O	SVM with over-sampling technique

Gaussian kernel, and sigmoid kernel. It was observed that the Gaussian kernel performed the best. In subsequent experiments, we fixed the SVM kernel at the Gaussian kernel.

Figure 2 shows the AMAUC values of the three SVM-related algorithms, namely SVM, SVM+U, SVM+O, on the twelve yeast datasets used in the experiments. For each algorithm, the AMAUC was calculated by taking the average of the MAUC values the algorithm received over the twelve yeast datasets. It can be seen from Fig. 2 that SVM+U performed better than SVM+O and SVM. Figure 3 shows the AMAUC values of the twelve boosting-related algorithms on the twelve yeast datasets used in the experiments. It can be seen from Fig. 3 that Boost III+U performed the best on the twelve yeast datasets.

Table 4 shows the MAUC values of Boost III+U and SVM+U, and compares them with the existing approaches using SVM only [9,17] on the twelve yeast datasets. For each dataset, the algorithm with the best performance (i.e., the highest MAUC) is shown in bold. It can be seen from Table 4 that Boost III+U has the best overall performance on the yeast datasets, and beats the existing approaches using SVM only [9,17].

Figure 4 shows the AMAUC values of the three SVM-related algorithms, namely SVM, SVM+U, SVM+O, on the twelve E. coli datasets used in the experiments. It can be seen that SVM+O outperformed SVM+U and SVM. Figure 5 shows the AMAUC values of the twelve boosting-related algorithms on the twelve E. coli datasets used in the experiments. It can be seen from Fig. 5 that Boost II+U performed the best among the twelve boosting-related algorithms.

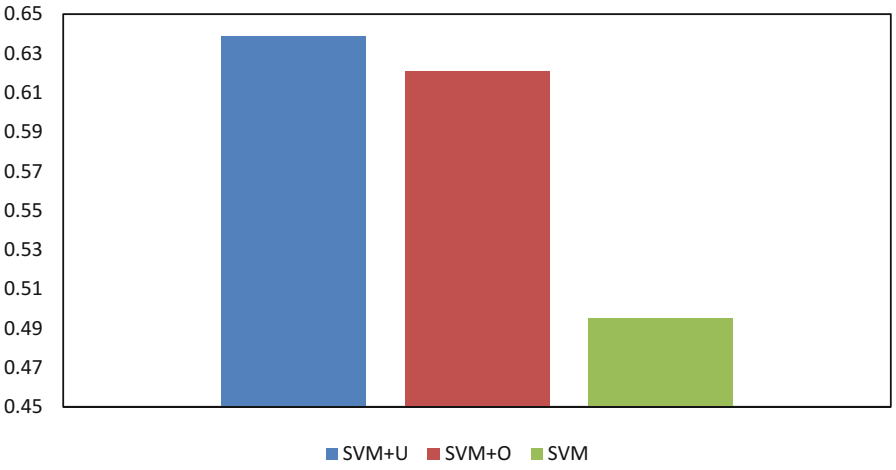

Fig. 2. AMAUC values of three SVM-related algorithms on twelve yeast datasets.

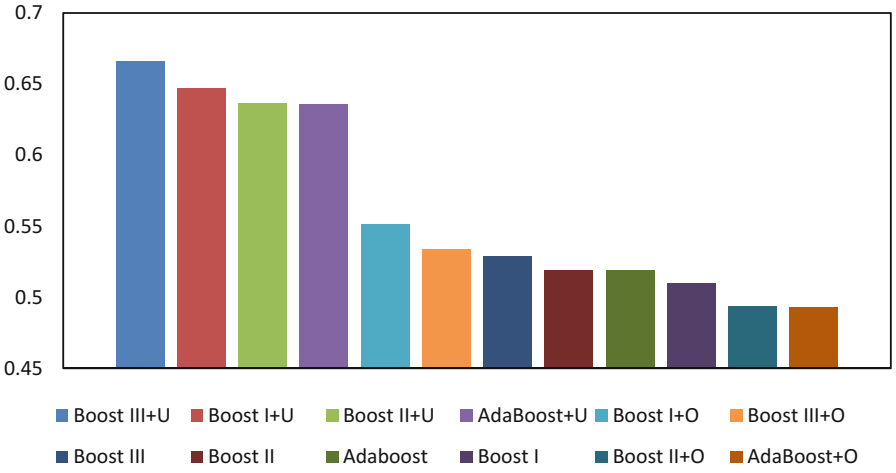

Fig. 3. AMAUC values of twelve boosting-related algorithms on twelve yeast datasets.

Table 5 shows the MAUC values of Boost II+U and SVM+O, and compares them with the existing approaches using SVM only [9,17] on the twelve E. coli datasets. It can be seen from Table 5 that Boost II+U has the best overall performance on the E. coli datasets, and beats the existing approaches using SVM only [9,17].

To summarize, one of our proposed boosting methods coupled with the undersampling technique achieves the best performance among all the fifteen algorithms studied in this paper on the yeast and E. coli datasets respectively. For the yeast datasets, this proposed boosting method is Boost III. For the E. coli

Table 4. MAUC values of three algorithms on twelve yeast datasets.

Dataset	Boost III+U	SVM+U	SVM
Yeast 50 knockouts	0.709	**0.802**	0.534
Yeast 50 knockdowns	0.664	**0.790**	0.529
Yeast 50 multifactorial	0.679	**0.736**	0.526
Yeast 100 knockouts	**0.729**	0.638	0.487
Yeast 100 knockdowns	**0.706**	0.690	0.489
Yeast 100 multifactorial	**0.701**	0.639	0.472
Yeast 150 knockouts	**0.633**	0.529	0.486
Yeast 150 knockdowns	**0.673**	0.519	0.494
Yeast 150 multifactorial	**0.680**	0.538	0.474
Yeast 200 knockouts	0.599	**0.622**	0.490
Yeast 200 knockdowns	0.612	**0.623**	0.490
Yeast 200 multifactorial	**0.608**	0.540	0.471
AMAUC	**0.666**	0.638	0.495

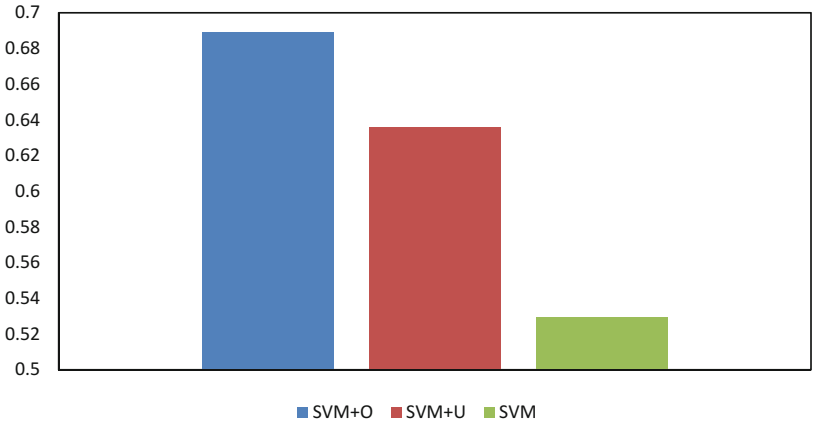

Fig. 4. AMAUC values of three SVM-related algorithms on twelve E. coli datasets.

datasets, this proposed boosting method is Boost II. Both boosting methods coupled with the under-sampling technique are superior to the existing approaches using SVM only [9,17].

Our boosting techniques are based on a weighted decision tree [1]. We have combined the boosting techniques with other machine learning algorithms including random forests [5], SVM with the linear kernel, SVM with the sigmoid kernel, SVM with the Gaussian kernel, and SVM with the polynomial kernel of degree 2. However, the performance of these other machine learning algorithms is inferior to the performance of the weighted decision tree used in this paper.

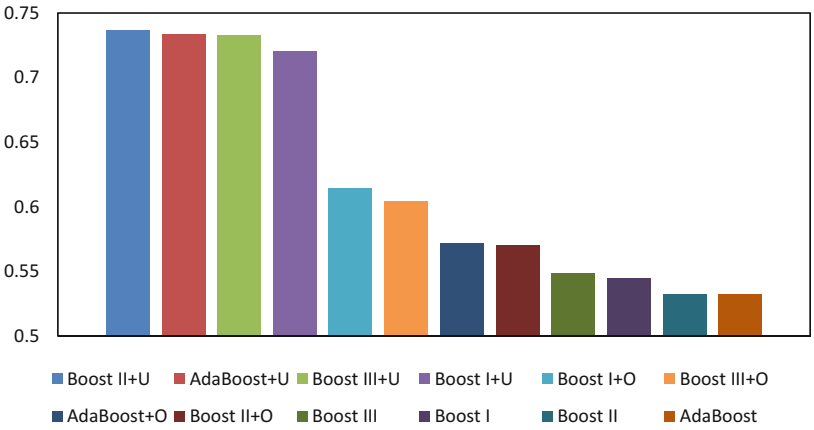

Fig. 5. AMAUC values of twelve boosting-related algorithms on twelve E. coli datasets.

Table 5. MAUC values of three algorithms on twelve E. coli datasets.

Dataset	Boost II+U	SVM+O	SVM
E. coli 50 knockouts	**0.872**	0.767	0.669
E. coli 50 knockdowns	**0.866**	0.776	0.505
E. coli 50 multifactorial	**0.879**	0.819	0.706
E. coli 100 knockouts	**0.770**	0.708	0.493
E. coli 100 knockdowns	**0.758**	0.712	0.492
E. coli 100 multifactorial	**0.750**	0.711	0.490
E. coli 150 knockouts	**0.625**	0.567	0.498
E. coli 150 knockdowns	**0.597**	0.596	0.500
E. coli 150 multifactorial	**0.636**	0.584	0.508
E. coli 200 knockouts	**0.702**	0.683	0.504
E. coli 200 knockdowns	**0.705**	0.682	0.495
E. coli 200 multifactorial	**0.678**	0.666	0.495
AMAUC	**0.736**	0.689	0.529

As a consequence, the results from the other machine learning algorithms are not reported here.

To evaluate the effectiveness of the proposed sampling and boosting techniques, we have also tested and compared the following four algorithms: (i) the weighted decision tree without boosting and sampling techniques; (ii) the weighted decision tree with boosting techniques only; (iii) the weighted decision tree with sampling techniques only; and (iv) the weighted decision tree with both boosting and sampling techniques. The results from the yeast and E. coli datasets are similar. For example, for the yeast datasets, the AMAUC value

for the weighted decision tree without boosting and sampling techniques is 0.45. This is lower than the existing approaches using SVM only (with AMAUC being 0.495 as shown in Table 4). When the weighted decision tree is coupled with only the Boost III technique, its AMAUC value is 0.53. When the weighted decision tree is coupled with only the under-sampling technique, its AMAUC value is 0.61. When the weighted decision tree is coupled with both Boost III and under-sampling techniques, its AMAUC value is 0.666 as shown in Table 4, which is much higher than the AMAUC value of 0.45 achieved by the weighted decision tree without boosting and sampling techniques.

4 Conclusion and Future Work

In this paper we present a new approach to gene network inference through regulatory link prediction. Our approach uses a weighted decision tree as the base learning algorithm coupled with sampling and boosting techniques to improve prediction performance. Experimental results demonstrated the superiority of the proposed approach over existing methods [9,17], and the effectiveness of our sampling and boosting techniques.

Moving forward, we are extending the techniques described here to miRNA-mediated regulatory networks. In addition to their importance as regulatory elements in gene expression, the capacity of miRNAs to be transported from cell to cell implicates them in a panoply of pathophysiological processes that include antiviral defense, tumorigenesis, lipometabolism and glucose metabolism [25,26]. This role in disease complicates our understanding of translational regulation via endogenous miRNAs. In addition, miRNAs seem to be present in different types of foods with potential implications on human health and disease. Understanding the biogenesis, transport and mechanisms of action of miRNAs on their target genes would result in possible therapies. Thus, reverse engineering of miRNA-mediated regulatory networks would contribute to human health improvement and disease treatment. We are exploring new ways for inferring such networks. We are also investigating other algorithms such as guided regularized random forests (GRRF) [7,24] and compare them with the weighted decision tree employed here.

References

1. Alhammady, H., Ramamohanarao, K.: Using emerging patterns to construct weighted decision trees. IEEE Trans. Knowl. Data Eng. **18**(7), 865–876 (2006)
2. Altaf-Ul-Amin, M., Afendi, F.M., Kiboi, S.K., Kanaya, S.: Systems biology in the context of big data and networks. In: BioMed Research International 2014 (2014)
3. Altaf-Ul-Amin, M., Katsuragi, T., Sato, T., Kanaya, S.: A glimpse to background and characteristics of major molecular biological networks. In: BioMed Research International 2015 (2015)
4. Batista, G.E., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explor. Newslett. **6**(1), 20–29 (2004)

5. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. CRC Press, New York (1984)
6. Bunkhumpornpat, C., Sinapiromsaran, K., Lursinsap, C.: Safe-Level-SMOTE: safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS, vol. 5476, pp. 475–482. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01307-2_43
7. Deng, H., Runger, G.: Gene selection with guided regularized random forest. Pattern Recogn. **46**(12), 3483–3489 (2013)
8. Freund, Y., Schapire, R.E.: A desicion-theoretic generalization of on-line learning and an application to boosting. In: Vitányi, P. (ed.) EuroCOLT 1995. LNCS, vol. 904, pp. 23–37. Springer, Heidelberg (1995). doi:10.1007/3-540-59119-2_166
9. Gillani, Z., Akash, M., Rahaman, M., Chen, M.: CompareSVM: supervised, support vector machine (SVM) inference of gene regularity networks. BMC Bioinformatics **15**, 395 (2014). http://dx.doi.org/10.1186/s12859-014-0395-x
10. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **21**(9), 1263–1284 (2009)
11. He, H., Ma, Y.: Imbalanced Learning: Foundations, Algorithms, and Applications. Wiley, Piscataway (2013)
12. Ideker, T., Krogan, N.J.: Differential network biology. Mol. Syst. Biol. **8**(1), 565 (2012)
13. Kibinge, N., Ono, N., Horie, M., Sato, T., Sugiura, T., Altaf-Ul-Amin, M., Saito, A., Kanaya, S.: Integrated pathway-based transcription regulation network mining and visualization based on gene expression profiles. J. Biomed. Inform. **61**, 194–202 (2016)
14. Leclerc, R.D.: Survival of the sparsest: robust gene networks are parsimonious. Mol. Syst. Biol. **4**(1), 213 (2008)
15. Liu, X.Y., Wu, J., Zhou, Z.H.: Exploratory undersampling for class-imbalance learning. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **39**(2), 539–550 (2009)
16. Margolin, A.A., Wang, K., Lim, W.K., Kustagi, M., Nemenman, I., Califano, A.: Reverse engineering cellular networks. Nat. Protoc. **1**(2), 662–671 (2006)
17. Mordelet, F., Vert, J.P.: SIRENE: supervised inference of regulatory networks. Bioinformatics **24**(16), i76–i82 (2008). http://bioinformatics.oxfordjournals.org/content/24/16/i76.abstract
18. Patel, N., Wang, J.T.L.: Semi-supervised prediction of gene regulatory networks using machine learning algorithms. J. Biosci. **40**(4), 731–740 (2015). http://dx.doi.org/10.1007/s12038-015-9558-9
19. Schaffter, T., Marbach, D., Floreano, D.: GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods. Bioinformatics **27**(16), 2263–2270 (2011)
20. Turki, T., Bassett, W., Wang, J.T.L.: A learning framework to improve unsupervised gene network inference. In: Perner, P. (ed.) MLDM 2016. LNCS (LNAI), vol. 9729, pp. 28–42. Springer, Cham (2016). doi:10.1007/978-3-319-41920-6_3
21. Turki, T., Ihsan, M., Turki, N., Zhang, J., Roshan, U., Wei, Z.: Top-k parametrized boost. In: Prasath, R., O'Reilly, P., Kathirvalavakumar, T. (eds.) MIKE 2014. LNCS (LNAI), vol. 8891, pp. 91–98. Springer, Cham (2014). doi:10.1007/978-3-319-13817-6_10
22. Turki, T., Wang, J.T.L.: A new approach to link prediction in gene regulatory networks. In: Jackowski, K., Burduk, R., Walkowiak, K., Woźniak, M., Yin, H. (eds.) IDEAL 2015. LNCS, vol. 9375, pp. 404–415. Springer, Cham (2015). doi:10.1007/978-3-319-24834-9_47

23. Turki, T., Wang, J.T.L., Rajikhan, I.: Inferring gene regulatory networks by combining supervised and unsupervised methods. In: 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016, Anaheim, CA, USA, 18–20 December 2016
24. Wang, S., Yao, X.: Relationships between diversity of classification ensembles and single-class performance measures. IEEE Trans. Knowl. Data Eng. **25**(1), 206–219 (2013). http://dx.doi.org/10.1109/TKDE.2011.207
25. Zhong, L., Wang, J.T.L., Wen, D., Aris, V., Soteropoulos, P., Shapiro, B.A.: Effective classification of microRNA precursors using feature mining and AdaBoost algorithms. OMICS: J. Integr. Biol. **17**(9), 486–493 (2013)
26. Zhong, L., Wang, J.T.L., Wen, D., Shapiro, B.A.: Pre-miRNA classification via combinatorial feature mining and boosting. In: 2012 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2012, Philadelphia, PA, USA, 4–7 October 2012, Proceedings, pp. 1–4 (2012). http://doi.ieeecomputersociety.org/10.1109/BIBM.2012.6392700