wRACOG: A Gibbs Sampling-Based Oversampling Technique

Barnan Das*, Narayanan C. Krishnan†, Diane J. Cook†
School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164-2752
Email: *barnandas@wsu.edu, †{ckn, cook}@eecs.wsu.edu

***Abstract*—As machine learning techniques mature and are used to tackle complex scientific problems, challenges arise such as the imbalanced class distribution problem, where one of the target class labels is under-represented in comparison with other classes. Existing oversampling approaches for addressing this problem typically do not consider the probability distribution of the minority class while synthetically generating new samples. As a result, the minority class is not well represented which leads to high misclassification error. We introduce wRACOG, a Gibbs sampling-based oversampling approach to synthetically generating and strategically selecting new minority class samples. The Gibbs sampler uses the joint probability distribution of data attributes to generate new minority class samples in the form of a Markov chain. wRACOG iteratively learns a model by selecting samples from the Markov chain that have the highest probability of being misclassified. We validate the effectiveness of wRACOG using five UCI datasets and one new application domain dataset. A comparative study of wRACOG with three other well-known resampling methods provides evidence that wRACOG offers a definite improvement in classification accuracy for minority class samples over other methods.**

***Keywords*—Imbalanced class distribution; oversampling; Gibbs sampling; Markov chain Monte Carlo (MCMC).**

I. Introduction

With a wide spectrum of industries drilling down into their data stores to collect previously unused data, data are being treated as the "new oil". Social media, e-commerce, and healthcare companies are refining and analyzing data with the goal of creating new products and/or adding value to existing ones. As machine learning techniques mature and find application in diverse data domains, new challenges arise from the data. One well-recognized challenge is the imbalanced class distribution problem, where one of the target classes (the *minority* class) is under-represented in comparison with the other class(es) (the *majority* class(es)). A widely accepted rule of thumb is: any dataset that contains less than $\sim 10\%$ minority class samples is considered to be imbalanced. The class imbalance problem exists in a variety of problem domains that include network intrusion detection [1], fraud detection [2], oil-spill detection [3], keyword extraction [4], any many more, where identifying members of the minority class is critical, sometimes more so than achieving optimal overall accuracy for the majority class.

The imbalance class distribution problem has received focused attention for over a decade. The common techniques that have been investigated include: *resampling*, which balances class priors of training data by either increasing the number of minority class data samples (*oversampling*) or decreasing the number of majority class data samples (*undersampling*); *cost-sensitive learning*, which assigns higher misclassfication cost of minority class samples than majority class; and *kernel-based learning* methods, which make the minority class samples more separable from the majority class by mapping the data to a high dimensional feature space. In addition to their ease of implementation, resampling methods have been preferred over other methods for a number of reasons [5]. First, because resampling occurs during preprocessing, the approach can be combined with others such as cost-sensitive learning, without changing the algorithmic anatomy [6]. Second, theoretical connections between resampling and cost-sensitive learning indicate that resampling can alter the misclassification costs of data points [7]. Third, empirical evidence demonstrates nearly identical performance between resampling and cost-sensitive learning techniques [8]. Although both under- and over-sampling techniques have been improving over the years, we focus our attention on oversampling because it is well suited for the application that motivates our investigation. Moreover, as many class imbalance problems involve an absolute rarity of minority class data samples [9], undersampling majority class examples is not advisable.

Existing oversampling approaches [10], [11] that add synthetic minority class samples to alleviate class imbalance problem typically rely on spatial location of these samples in the Euclidean space. These approaches harvest *local* information of minority class samples to generate new synthetic samples that are also assumed to belong to the minority class. Although this approach may be acceptable for data sets where a crisp decision boundary exists between the classes, spatial location-based synthetic oversampling is not suitable for data sets that have overlap between the minority and majority classes. Therefore, a better idea is to exploit *global* information of minority class samples, which can be done by considering the probability distribution of the minority class while synthetically generating new minority class samples.

In this paper, we introduce a Gibbs sampling-based oversampling method, called **w**rapper-based **RA**pidly **CO**nverging **G**ibbs sampler or wRACOG, for generating and strategically selecting new minority class samples. Specifically, the proposed algorithm generates new minority class samples via Gibbs sampling by using the joint probability distribution and interdependencies of data attributes. wRACOG iteratively selects samples from the Markov chain generated by the Gibbs sampler that have the highest probability of being misclassified by a learning model (*wrapper*). wRACOG is based on our previous algorithm RACOG, which also uses Gibbs sampling to generate a Markov chain, but selects samples from the Markov chain using a predefined *lag*.

We validate our approach using five UCI datasets, carefully modified to exhibit class imbalance, and one new dataset with inherent extreme class imbalance from a pervasive computing application domain. The performance of wRACOG is compared with RACOG as well as two existing oversampling techniques (SMOTE [10] and SMOTEBoost [11]) and an undersampling technique (RUSBoost [12]).

II. Related Work

Learning from imbalanced class datasets is a niche, yet critical area in supervised machine learning. Therefore a wide spectrum of related techniques have been proposed [13], [14] for improved classification of minority class samples. One of the most common solutions is cost sensitive learning (CSL) which subverts the underlying assumption of classifiers that all misclassification errors are equal. By assigning a sufficiently high cost to minority class samples, the algorithm devotes more attention to these points to learn an effective class boundary. The effectiveness of CSL methods has been validated both theoretically [7] and empirically [15]. However, the application of CSL methods is restricted due to the unavailability of cost-sensitive implementations of well-known learning algorithms and the need to manually determine the misclassification cost of minority class samples.

A second direction involves adapting the underlying classification algorithm to consider imbalanced classes, typically using kernel-based learning methods. For instance, Wu et al. [16] propose a kernel-boundary alignment (KBA) algorithm for adjusting the SVM class boundary. KBA is based on the idea of modifying the kernel matrix generated by a kernel function according to the imbalanced data distribution. Another technique proposed by Fung et al., called the k-category proximal support vector machine (PSVM) [17], transforms the soft-margin maximization paradigm into a simple system of k-linear equations for either linear or non-linear classifiers.

The most common approach, however, is to resample, or modify the dataset in a way that balances the class distribution. Determining the ideal class distribution is an open problem [18] and in most cases handled empirically. Naive resampling methods include oversampling the minority class by duplicating existing data points and undersampling the majority class by removing chosen data points. However, random over-sampling and under-sampling increases the possibility of overfitting and discarding useful information from the data, respectively. An intelligent way of oversampling is to synthetically generate new minority class samples. For example, SMOTE [10] constructs new minority class samples that are close to existing ones by selecting a point on the imaginary line segment connecting an existing minority class sample to its k-minority class nearest neighbors in the Euclidean space. In addition, adaptive synthetic sampling technique Borderline-SMOTE [19], generates synthetic samples only for those minority class samples that are closest to the decision boundary between the two classes. ADASYN [20], on the other hand, uses the density distribution of the minority class samples as a criterion to automatically decide the number of synthetic samples that need to be generated for each minority class sample. There are other methods that combine the powers of resampling and ensemble learning, SMOTEBoost [11], RUSBoost [12], and SMOTE-Bagging [21].

The fundamental principle of oversampling techniques such as SMOTE, Borderline-SMOTE and SMOTEBoost, is to generate new data samples that are spatially close to existing minority class samples in the Euclidean space. Ideally, the new data samples are representative of the entire minority class and not just of a local region. Therefore, in this paper we focus on satisfying the criteria of *global* representation of the minority class by generating multivariate samples from the joint probability distribution of the underlying random variables or attributes.

III. Motivation and Application

We are motivated to pursue this challenge by a problem in pervasive computing. Specifically, we are designing smart environments that perform health monitoring and assistance. One type of assistance that is valuable for individuals with cognitive impairment is automated prompts that aid with activity initiation and completion. We postulate that prompt timing can be automated by incorporating contextual information of an activity provided by a smart home.

To determine the ability of a machine learning algorithm to generate appropriate activity-aware prompts, we performed a study in our smart home with 128 volunteer participants, aged 50+, who are healthy older adults or individuals with mild cognitive impairment. The smart home is a two-story apartment equipped with sensors that monitor motion, door open/shut status, and usage of water, burner, and specific items throughout the apartment. Clinically-trained psychologists watch over a web camera as the participants perform 8 different activities. The psychology experimenter remotely issues a prompt when they determine that the individual is having difficulty initiating or completing an activity. Sensor events, denoted by the event date, time, sensor identifier, and message, are collected continuously, along with the prompt timings. A human annotator determines the beginning and end of an activity and its sub-steps from the sensor events and tags them with appropriate labels. On the basis of these annotations, the feature vector for every activity sub-step (which could be a collection of consecutive sensor events) present in the database is generated that consists of temporal, spatial and contextual features.

We can view automated prompting as a supervised learning problem in which each activity step is mapped to a "prompt" or "no-prompt" class label. Ground truth for the class labels is provided by the experimenter-based timing of manually-initiated prompts in the study. The *prompting* dataset[1], as we would like to call it, has 3980 examples with 17 features. Out of the 17 features, 4 are categorical and the rest are numeric. The difficulty that is faced for the prompting problem is that the majority of activity steps are "no-prompt" cases and standard machine learning algorithms will likely map all data points to this class, which defeats the purpose of the intervention. Our goal is to design solutions to the class imbalance problem that improve sensitivity for this prompting application.

We evaluate the applicability of our algorithm on five additional real-world datasets from the UCI machine learning repository that exhibit the class imbalance problem: *abalone*,

[1] Available at: http://ailab.wsu.edu/casas/datasets/prompting.zip

TABLE I. DESCRIPTION OF SELECTED DATASETS

Dataset	Size	Dim	% Min. Class	Description
Prompting	3,980	17	3.7437	Predicting intervention times in daily activities of older individuals with dementia.
Abalone	4,177	8	6.2006	Predicting age of large sea snails, abalone, from its physical measurements.
Car	1,728	6	3.9931	Evaluating car acceptability based on price, technology and comfort.
Nursery	12,960	8	2.5463	Nursery school application evaluation based on financial standing, parents' education and social health.
Letter	20,000	16	3.7902	Classifying English alphabets using image frame features.
Connect-4	5000	42	10.0000	Predicting first player's outcome in connect-4 game given 8-ply positions information .

car, *nursery*, *letter*, and *connect-4*. Characteristics of these datasets are summarized in Table I. The real-valued attributes were transformed into discrete sets using equal-width binning. The multi-class datasets were converted into binary classes by choosing a particular class label as the *minority* class and the rest of the class labels together as the *majority* class.

IV. PROPOSED METHOD: WRACOG

Our proposed approach to oversampling for imbalanced class distributions uses Gibbs sampling to generate new minority class data samples. Gibbs sampling is a type of Markov chain Monte Carlo (MCMC) method which originated with the Metropolis algorithm [22], [23]. A Markov chain is a collection of random variables with the property that, given the present, the future is conditionally independent of the past. A first-order Markov chain is defined as a series of random variables $z^{(1)}, ..., z^{(M)}$, such that the conditional independence property given in Equation 1 holds true for $m \in \{1, .., M-1\}$.

$$P(z^{(m+1)}|z^{(1)}, .., z^{(m)}) = P(z^{(m+1)}|z^{(m)}) \tag{1}$$

The equation can be represented as a directed graph that forms a chain as shown in Figure 1. A Markov chain is thus specified by the probability distribution of the initial variable $P(z^{(0)})$ and the conditional probabilities of the subsequent variables (also known as transition probabilities).

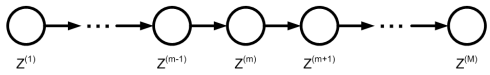

Fig. 1. A Markov chain

Therefore, the marginal probability of a variable of interest can be expressed in terms of the marginal probability of the previous variable and the transition probability from the previous variable to the current variable (see Equation 2).

$$P(z^{(m+1)}) = \sum_{z^{(m)}} P(z^{(m+1)}|z^{(m)})P(z^{(m)}) \tag{2}$$

A. Standard Gibbs Sampler

The goal of a Gibbs sampler is to create a Markov chain of random variables that converge to a target probability distribution. The approach is applicable in situations where a random variable Z has at least two dimensions ($z =< z_1, .., z_k >, k > 1$). At each sampling step, the algorithm considers univariate conditional distributions where each of the dimensions but one is assigned a fixed value. Rather than picking the entire collection of attribute values at once, a separate probabilistic choice is made for each of the k dimensions, where each choice depends on the values of the other $k-1$ dimensions and the previous value of the same dimension. Such conditional distributions are easier to model than the full joint distribution. Figure 2 shows the algorithm for the standard Gibbs sampler. As we would explain in this section, we exploit the univariate conditional distribution of each dimension, represented by $P\left(Z_i|z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right)$, to probabilistically choose attribute values that form a new synthetically generated sample.

Algorithm 1: Gibbs Sampler

1: $Z^{(0)} =< z_1^{(0)}, ..., z_k^{(0)} >$
2: **for** $t = 1$ to T
3: **for** $i = 1$ to k
4: $z_i^{(t+1)} \sim P\left(Z_i|z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right)$

Fig. 2. Gibbs Sampling

Implementation of Gibbs samplers are traditionally dependent on two factors. The first factor is the number of sample generation iterations that are needed for the samples to reach a stationary distribution, that is, when the marginal distribution of $Z^{(n)}$ is independent of n. In order to avoid the estimates being contaminated by values at iterations before that point (referred to as the ***burn-in***), earlier samples are discarded. The second factor is based on the fact that a sample generated during one iteration is highly dependent on the previous sample. This correlation between successive values, or *autocorrelation*, is avoided by defining a suitable ***lag***, or number of consecutive samples to be discard from the Markov chain following each accepted generated sample.

B. Factoring Joint Probability Distribution

A stumbling block in successful usage of Gibbs sampling is calculating the conditional distribution that appears in Step 4 of the algorithm (Figure 2) which is used to probabilistically determine attribute values for the new sample. A simplified version of the conditional distribution, shown in Equation 3, calculates a ratio between the joint probability of all attribute values for the random variable and a normalizing factor.

$$P\left(Z_i|z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right) = \frac{P\left(z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_i^{(t)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right)}{P\left(z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right)} \tag{3}$$

However, in cases of datasets which have extremely rare minority class samples and a sufficiently high dimensional feature vector, the probability of joint occurrences of other attribute values is extremely low due to an insufficient number of such minority class samples. Because the standard Gibbs sampling algorithm does not impose dependencies on attributes

Algorithm 2: Chow-Liu Dependence Tree Construction

1: Compute the mutual information between each pair of variables, $i \neq j$: $I_P(X_i; X_j) = \sum_{x,y} P(x,y) log \frac{P(x,y)}{P(x),P(y)}$
2: Build a complete undirected graph with variables in X as vertices and the weight of an edge connecting X_i and X_j by $I_P(X_i; X_j)$.
3: Build a maximum weighted spanning tree.
4: Transform the resulting undirected tree to a directed one by choosing a root variable and setting the direction of all edges to be outward from it.

Fig. 3. Chow-Liu Dependence Tree Construction

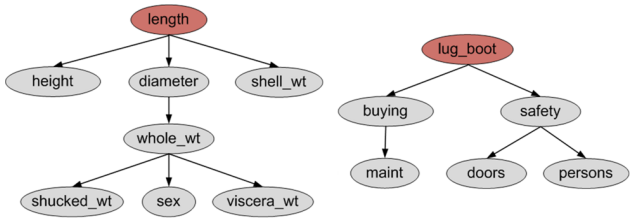

Fig. 4. Trees for (*left*) abalone and (*right*) car datasets

of the random variables, the sampling performed in Step 4 of the algorithm disallows exploration of the entire space of attributes and is therefore less likely to generate points that are consistent with such a minority class distribution. On the other hand, considering a full joint distribution will be prone to error because there are insufficient minority points to accurately estimate the probabilities that are used in Equation 3.

As we will see, the proposed algorithm explores the state space of attributes more thoroughly by factoring the large-dimensional joint distribution into a *directed acyclic graph* (DAG) that imposes explicit dependencies between attributes. Thus, the probability of a particular data sample, x, represented as a collection of attributes $\{x_i : 1 \leq i \leq M\}$, is computable as:

$$P(x) = \prod_i P(x_i | x_{parents(i)}) \tag{4}$$

Traditionally, a DAG is constructed by learning a Bayesian network using search techniques such as hill climbing or simulated annealing. An alternative, less computationally expensive approach, is to employ the Chow-Liu algorithm (1968) described in Figure 3, to construct a Bayesian tree of dependencies by reducing the problem of constructing a maximum likelihood tree to that of finding a maximal weighted spanning tree in a graph. Thus, the Chow-Liu algorithm runs in $O(D^2 log(D))$ time where D is the dimension of the data. Our algorithm employs this approach to finding a Bayesian tree among attribute dependencies. This allows each attribute (but the root) to have exactly one parent on which its value depends [24]. Figure 4 shows Bayesian trees formed from the *abalone* and *car* datasets, where attributes *length* and *lug_boot* are chosen as the roots of the Bayesian trees for *abalone* and *car*, respectively. Thus the joint probability distribution in Equation 3 is now factored as follows:

$$P\left(Z_i | z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right) = \frac{P\left(z_{root} \prod_x P(z_x | z_{parents(x)})\right)}{P\left(z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right)} \tag{5}$$

C. The RACOG Algorithm

We first describe the RACOG algorithm and eventually show what improvements have been incorporated in wRACOG. The **RA**pidy **CO**nverging **G**ibbs sampler (RACOG) uses Gibbs sampling to generate new minority class samples from the probability distribution of the minority class approximated using Chow-Liu algorithm. RACOG enhances standard Gibbs sampling by offering an alternative mechanism for choosing the initial values of the random variable Z (denoted by $Z^{(0)}$), that help generate samples which rapidly converge to the target (minority class) distribution, and by imposing dependencies among the attributes which form the k dimensions of the random variable.

Conventionally, the initial values of the random variable to "ignite" the Gibbs sampler are randomly chosen from the state space of the attributes. This approach requires a high burn-in period and an extremely large number of iterations for the sampler to converge with the target distribution. On the other hand, in RACOG, we choose the minority class data points as the set of initial samples and run the Gibbs sampler for every minority class sample. The total number of iterations for the Gibbs sampler is restricted by the desired *minority:majority* class distribution. Thus, RACOG produces multiple Markov chains, each starting with a different minority class sample, instead of one very long chain as done in conventional Gibbs sampling. As the initial samples of RACOG are chosen directly from the minority class samples, it helps in achieving faster convergence of the generated samples with the minority class distribution. This claim is validated with the help of a convergence test performed in Section VI.

There are arguments in the literature about the pros and cons of a single long Markov chain versus multiple shorter chain approaches. Geyer [25] argues that a single long chain is a better approach because if long burn-in periods are required or if the chains have high autocorrelations, using a number of shorter chains may result in chains that are too short to adequately represent the minority class. However, a single long chain requires a very large number of iterations to converge with the target distribution. Our experiments show that the argument made by Geyer does not hold when multiple chains are generated using the minority class samples as the initial samples of the Gibbs sampler.

Finally, to generate a new minority class sample, the value of an attribute i represented by $z_i^{(t+1)}$ is determined by randomly sampling from the distribution of the state space (all possible values) of attribute i represented by $P\left(Z_i | z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right)$. Figure 5 summarizes the RACOG algorithm that in $O(D^2 log(D) + N.T.D)$ time where, D = dimension of data, N = cardinality of minority class, and T = predetermined number of iterations.

While the RACOG algorithm enhances the traditional Gibbs sampler by making it suitable for class imbalanced data, this approach does not take into account the usefulness of the generated samples. As a result, RACOG might add minority

Algorithm 3: RACOG

function RACOG (*minority*, N, k, β, α, T)
Input: *minority* = minority class data points; N = size of *minority*; k = *minority* dimensions; β = burn-in period; α = lag; T = total number of iterations
Output: *new_samples* = new minority class samples
Construct Bayesian tree *BT* using Chow-Liu algorithm.
for $n = 1$ to N **do**
    $Z^{(0)} =$ *minority(n)*
    for $t = 1$ to T **do**
        for $t = 1$ to k **do**
            Simplify $P\left(Z_i | z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)}\right)$ using *BT* and Equation 3.
            $z_i^{(t+1)} \sim P(S_i)$ where S_i is the state space of attribute i.
        if $t > \beta$ **AND** t **mod** $(\alpha) = 0$
            new_samples = *new_samples* + $Z^{(t)}$
return *new_samples*

Fig. 5. The RACOG Algorithm

Algorithm 4: wRACOG

function wRACOG (*train*, *validation*, *wrapper*, *slide_win*, *threshold*, *slide_win*)
Input: *train* = training dataset enhanced at each iteration with new samples; *validation* = validation set on which trained model is tested at every iteration to track improvements; *wrapper* = classifier that is retrained on the enhanced dataset at every iteration; *slide_win* = sensitivities of previous iterations; *threshold* = threshold of standard deviation of sensitivities over *slide_win*
Output: *new_train* = final hypothesis encoded in the oversampled training set
Build *model* by training *wrapper* on *train*
Run Gibbs sampler on all minority class samples simultaneously
do
    Perform prediction on newly generated samples using *model*
    Add misclassified samples to form *new_train*
    Train *model* on *new_train* using *wrapper*
    Perform prediction on *validation* set using trained *model* and add sensitivity to *slide_win*
while ($\sigma(slide\_win) \geq$ *threshold*)
return *new_train*

Fig. 6. The wRACOG Algorithm

class samples that are redundant and have no contribution towards constructing a better hypothesis. In order to address this issue, we propose wRACOG, an enhancement to the RACOG algorithm.

D. The wRACOG algorithm

The enhanced RACOG algorithm, named wRACOG, is a wrapper-based technique over RACOG utilizing Gibbs sampling as the core data sampler. The purpose of introducing wRACOG is to get rid of *burn-in*, *lag* and predefined number of iterations associated with sample selection in Gibbs sampling. By performing iterative training on the dataset with newly generated samples, wRACOG selects those samples from the Markov chain that have the highest probability of being misclassified by the learning model generated from the previous version of the dataset.

While the RACOG algorithm generates minority class samples for a fixed (predefined) number of iterations, the wRACOG algorithm keeps on fine tuning its hypothesis at every iteration by adding new samples until there is no further improvement with respect to a chosen performance measure. As our goal is to improve the performance of classifiers on the minority class, wRACOG keeps on adding new samples until there is no further improvement in *sensitivity* (true positive rate) of the *wrapper* classifier (the core classifier that retrains at every iteration) of wRACOG. This process acts as the "stopping criterion" for the wRACOG algorithm. Please note that *sensitivity* is not an invariant choice for the stopping criteria. Other performance measures, such as *precision*, *F-measure*, etc. can also be used and the choice is entirely application dependent.

At each iteration of wRACOG, new minority class samples are generated by the Gibbs sampler. The model learned by the *wrapper* classifier on the enhanced set of samples produced in the previous iteration is used to make predictions on the newly generated set of samples. During classification, the class labels of the new samples are assumed to be the same as the minority class because the Gibbs sampler produces new samples from the probability distribution of the minority class. Those samples that are misclassified by the model are added to the existing set of data samples and a new model is trained using the *wrapper* classifier. At each iteration, the trained model performs prediction on a held out *validation* set and the sensitivity of the model is recorded. Generation of new samples stops once the standard deviation of sensitivities over the past iterations falls below a threshold. As the wRACOG algorithm might end up running many iterations, the standard deviation of sensitivities is calculated over a fixed number of most recent iterations (*slide_win*). We use the values *slide_win=10* and *threshold=0.02* for our experiments, determined by performing an empirical study on the datasets described in this paper. The wRACOG algorithm is summarized in Figure 6. wRACOG runs in $O(D^2 log(D) + B.N.D + B.W)$ time where, $D =$ dimension of data, $B =$ number of sample batches generated before stopping criteria is satisfied, $N =$ cardinality of minority class, and $W =$ time complexity of *wrapper* classifier.

Although wRACOG is similar to existing boosting techniques (such as AdaBoost) in the way misclassified samples are assigned higher weights to ensure their selection during random sampling in the next boosting iteration, there are a number of major differences. Firstly, while in traditional boosting, both training and prediction are performed on the same set of data samples, wRACOG trains the current hypothesis on the data samples from the previous iteration and performs prediction only on the newly generated samples from the Gibbs sampler. Secondly, there is no concept of changing weights of the samples before resampling, as the newly generated samples are directly added to the existing set of samples. Thirdly, wRACOG does not use weighted voting of multiple hypotheses learned at every iteration. Instead, it employs multiple iterations to fine tune a single hypothesis. We hypothesize that by applying this approach we can reduce the generation of redundant samples to converge more closely to the true distribution of the minority class, and also reduce the overhead of generating multiple hypotheses as is employed by traditional boosting techniques.

V. Experimental Setup

We hypothesize that Gibbs sampling, with additional modeling of attribute dependencies, will yield improved results over existing resampling methods for problems that exhibit class imbalance. We compare the performance of the classifiers on datasets preprocessed by wRACOG, RACOG, and three well-known sampling techniques, against the datasets with no preprocessing (henceforth named *Baseline*).

SMOTE: SMOTE [10] oversamples the minority class by creating "synthetic" samples based on spatial location of the samples in the Euclidean space. Oversampling is performed by considering each minority class sample and introducing synthetic examples along the line segments joining any or all of the k-minority class nearest neighbors. The k-nearest neighbors are randomly chosen depending upon the amount of oversampling that is required. Synthetic samples are generated in the following way: First, the difference between the minority class sample under consideration and its nearest neighbor is computed. This difference is multiplied by a random number between 0 and 1, and it is added to the sample under consideration to form a new sample. Consequently, by adding diversity to the minority class, this approach forces the decision boundary between the two regions to be crisper. However, as SMOTE does not rely on the probability distribution of the minority class as a whole, there is no guarantee that the generated samples would belong to the minority class, especially when the samples from the majority and minority classes overlap [26]. We use SMOTE to produce a 50:50 distribution of minority and majority class samples, which is considered as near optimum [27].

SMOTEBoost: By using a combination of SMOTE and a standard boosting procedure, SMOTEBoost [11] models the minority class by providing the learner with misclassified minority class samples from the previous boosting iteration and a broader representation of those samples achieved by SMOTE. The inherent skewness in the updated distribution is rectified by introducing SMOTE to increase the number of minority class samples according to the new distribution. SMOTEBoost maximizes the margin of the skewed class dataset and increases the diversity among the classifiers in the ensemble by producing a unique set of synthetic samples at every iteration. Although iterative learning of the weak learner by the boosting procedure attempts to form a hypothesis which improves the classification of minority class samples, the quality of the generated samples is still dependent on the spatial location of minority class samples in the Euclidean space, as is done by SMOTE. Moreover, if SMOTEBoost executes SMOTE k times (for k boosting iterations), generating $k \times (\#\textit{majority class samples} - \#\textit{minority class samples})$ samples is computationally expensive.

RUSBoost: RUSBoost [12] is very similar to SMOTEBoost, but claims to achieve better classification performance on the minority class samples by randomly under-sampling (RUS) the majority class. Although this method results in a simpler algorithm with a faster model training time, it fails to achieve desired performance (explained later in Section VI) as claimed by Seiffert et al., especially when the datasets have an absolute rarity of minority class samples. As most of the data sets under consideration have an absolute rarity of minority class samples, the class imbalance ratio has been set to 35:65 (minority:majority). The choice of class distribution is based on the empirical investigations performed by Khoshgoftaar et al. [28] which verify that a 35:65 class distribution would result in better classification performance than a 50:50 class distribution when samples from one class are extremely rare as compared to others.

wRACOG and its predecessor RACOG are compared with the aforementioned alternative sampling techniques. RACOG oversamples the minority class to achieve a 50:50 class distribution. Therefore, the total number of iterations is fixed and is determined on the basis of ($\#\textit{majority class samples} - \#\textit{minority class samples}$), *burn-in* and *lag*. A *burn-in* period of 100 and a *lag* of 20 iterations are chosen as the convention in the literature [29] to avoid autocorrelation among the samples generated by the Gibbs sampler. wRACOG, on the other hand, adds samples to the minority class until the standard deviation of sensitivity (true positive rate) over the 10 recent iterations falls below an empirically determined threshold. The classifiers presented in Table II are used as *wrapper* classifiers and are trained on an enhanced dataset at every iteration of wRACOG.

We choose four of the best-known classifiers in machine learning to evaluate the performance of the proposed method and other sampling approaches: decision tree, SVM, k-nearest neighbor and logistic regression. We performed parameter tuning of the chosen classifiers on the baseline data before pre-processing. The parameter values of the classifiers that performed best on the baseline datasets are used in conducting experiments with the existing and proposed sampling approaches. Table II lists parameter values for the corresponding classifiers.

TABLE II. Classifiers and parameter values

Classifier	Parameter Values
C4.5 Decision Tree	Confidence factor = 2, Minimum # instances per leaf = 2
SVM	Kernel = RBF, RBF kernel $\gamma = 0.01$
k-Nearest Neighbor	k =5, Distance measure = Euclidean
Logistic Regression	Log likelihood ridge value = $1 \times 10^{(-8)}$

All experiments are performed using 5-fold cross validation where prediction is performed on unsampled held-out data. We report the following performance measures in this paper: Sensitivity (*true positive rate*), G-mean ($\sqrt{\textit{true positive rate} \times \textit{true negative rate}}$), and Area Under ROC Curve (AUC-ROC).

VI. Results and Discussion

The primary limitation of classifiers that model imbalanced class datasets is in achieving desirable prediction accuracy for the minority class. That is, the sensitivity is typically low assuming that the minority class is represented as the positive class. The proposed approach places emphasis on boosting the sensitivity of the classifiers while maintaining a strong prediction performance for both of the classes, which is measured by G-mean. However, we understand that the choice of performance measure that needs to be boosted when dealing with class imbalanced dataset is tightly coupled with the application domain. Moreover, finding a trade-off between improving classifier performance on the minority class in isolation and on the overall dataset should ideally be left at the discretion of the domain expert.

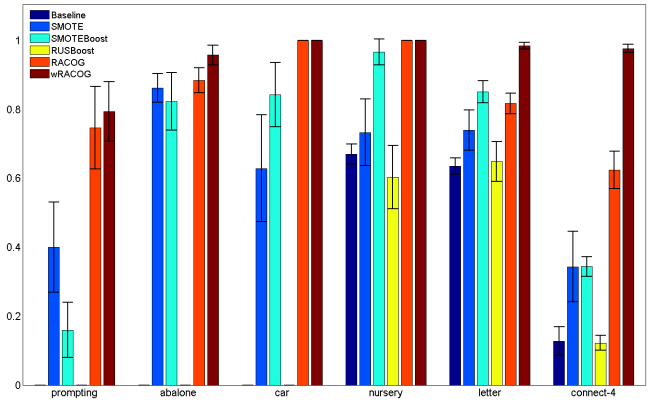

Fig. 7. Sensitivity for C4.5 Decision Tree

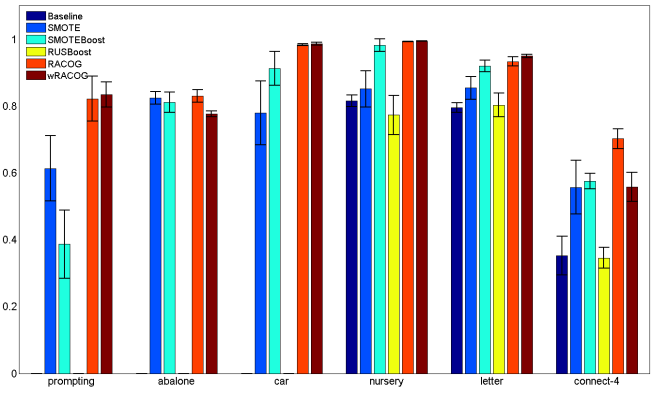

Fig. 8. G-mean for C4.5 Decision Tree

We compare the sensitivity of the C4.5 decision tree across all six approaches in Figure 7. From the figure it is quite evident that both wRACOG and RACOG perform better than the other methods, although the variation in performance of RACOG and wRACOG is not significant. RUSBoost fails by performing nowhere close to the oversampling techniques. The poor performance of RUSBoost can be attributed to the rarity of minority class samples in the datasets under consideration. When the minority class samples are already rare, random under-sampling of the majority class to achieve a 35:65 class distribution, as is done by RUSBoost at each iteration, makes the majority class samples rare as well. Thus, the weak leaner of RUSBoost does not learn anything on the majority class which increases the error rate of the hypothesis at every iteration. Both SMOTE and SMOTEBoost are good contenders, although SMOTEBoost outperforms SMOTE in most of the cases. Although wRACOG performs better than RACOG on four datasets, they perform equally well for the *car* and *nursery* datasets. We verify the statistical significance of these improvements using a *Student's t test*. RACOG and wRACOG do exhibit significant ($p < 0.05$) performance improvement over SMOTEBoost. Experimental results using other classifiers and statistical significance of the performance improvements (in bold) of RACOG and wRACOG over SMOTEBoost are given in Tables V, VI and VII.

Because the sampling techniques boost the minority class, there is a tendency for the false positive rate to increase (see Table III). However, the increase in false positive rate is not very significant and therefore it does not affect the G-mean scores. Figure 8 reports the G-mean scores of C4.5 decision tree on all the methods when tested with the six datasets. Clearly, RACOG and wRACOG result in a superior performance over SMOTE and SMOTEBoost. However, SMOTEBoost is a very strong contender.

In Figure 9, we plot the ROC curves produced by the different approaches on each of the 6 datasets when evaluated with a C4.5 decision tree. For the *prompting*, *abalone* and *car* datasets, Baseline and RUSBoost do not perform any better than random prediction. The performance of RACOG and wRACOG are clearly better than SMOTE and SMOTEBoost for the *prompting* dataset. However, there is no clear winner among them for the *abalone*, *car* and *nursery* datasets. *connect-4* has a very unique ROC plot which helps in analyzing the performance of the classifiers distinctly. This unique characteristic of *connect-4* is probably because all 42 attributes (representing each position of a 7×6 grid) of this dataset have only three possible categorical values: position taken by player A, position taken by player B, and blank, which gives the *win* or minority class a very distinct pattern. The AUC-ROCs are reported in Table IV. For the *letter* and *connect-4* datasets, the AUC-ROC for SMOTEBoost is higher than RACOG and wRACOG. However, no statistically significant improvement of RACOG and wRACOG was found over SMOTEBoost based on AUC-ROC. As there is no clear winner between RACOG and wRACOG on any of the performance measures we do not conduct any statistical significance test between them.

TABLE III. FALSE POSITIVE RATES FOR C4.5 DECISION TREE

Dataset	*prompting*	*abalone*	*car*	*nursery*	*letter*	*connect-4*
Baseline	0	0	0	0.0051	0.0031	0.0096
SMOTEBoost	0.0117	0.1959	0.0072	0.0008	0.0046	0.0356
wRACOG	**0.1178**	**0.3696**	**0.0265**	**0.0111**	**0.0832**	**0.0693**

Convergence Diagnostic Test: In the current work we explore the strengths of Markov chain Monte Carlo techniques, specifically Gibbs sampling, to generate new minority class samples. A major issue for the successful implementation of any MCMC technique is to determine the number of iterations required for the generated samples to converge to the target distribution. Researchers in econometrics [30] use formal methods such as the Raftery-Lewis test [29] to make this determination. Given outputs from a Gibbs sampler, the Raftery-Lewis test provides the answer to: *how long should we monitor the chain of samples?* Here, one specifies a particular quantile q of the distribution of interest (typically 2.5% and 97.5%, to give a 95% confidence interval), an accuracy σ of the quantile, and a power $1-\beta$ for achieving this accuracy on the specified quantile. These parameters are used to determine the burn-in (M), the total number of iterations required to achieve the desired accuracy for the posterior (N), the appropriate lag (k), and the number of iterations (N_{min}) that would be needed if the samples represented an *independent and identically distributed (iid)*[2] chain, which is not true in our case because of the autocorrelation structure present in the Markov chain of generated samples. These output values can be combined

[2] An *iid* sequence is a very special kind of Markov chain; whereas a Markov chain's future is allowed (but not required) to depend on the present state, an *iid* sequence's future does not depend on the present state at all.

TABLE IV. AUC-ROC FOR C4.5 DECISION TREE

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.5000 ± 0.0000	0.8681 ± 0.0359	0.8546 ± 0.0334	0.5000 ± 0.0000	0.8851 ± 0.0207	0.8717 ± 0.0404
abalone	0.5000 ± 0.0000	0.8693 ± 0.0272	0.8718 ± 0.0262	0.5000 ± 0.0000	0.8646 ± 0.0233	0.8727 ± 0.0208
car	0.5000 ± 0.0000	0.9595 ± 0.0320	0.9957 ± 0.0025	0.5000 ± 0.0000	0.9879 ± 0.0027	0.9904 ± 0.0067
nursery	0.9849 ± 0.0077	0.9515 ± 0.0219	0.9999 ± 0.0001	0.7999 ± 0.0456	0.9968 ± 0.0005	0.9967 ± 0.0005
letter	0.8745 ± 0.0056	0.9340 ± 0.0153	0.9865 ± 0.0080	0.8226 ± 0.0286	0.9659 ± 0.0102	0.9707 ± 0.0026
connect-4	0.6215 ± 0.0310	0.6907 ± 0.0335	0.8317 ± 0.0140	0.5552 ± 0.0092	0.7333 ± 0.0181	0.7115 ± 0.0282

TABLE V. PERFORMANCE COMPARISON OF ALL APPROACHES USING SVM

Sensitivity

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.0000 ± 0.0000	0.6667 ± 0.0667	0.2600 ± 0.0641	0.0000 ± 0.0000	$\mathbf{0.5600 \pm 0.1011}$	$\mathbf{0.5400 \pm 0.0723}$
abalone	0.0000 ± 0.0000	0.9462 ± 0.0251	0.8462 ± 0.0769	0.0000 ± 0.0000	$\mathbf{0.9385 \pm 0.0161}$	$\mathbf{0.9846 \pm 0.0161}$
car	0.0000 ± 0.0000	1.0000 ± 0.0000	0.9000 ± 0.1195	0.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
nursery	0.0000 ± 0.0000	0.9939 ± 0.0136	0.9848 ± 0.0186	0.7091 ± 0.0529	1.0000 ± 0.0000	1.0000 ± 0.0000
letter	0.4645 ± 0.0430	0.9026 ± 0.0409	0.9092 ± 0.0388	0.6487 ± 0.0396	0.9289 ± 0.0220	0.9364 ± 0.0166
connect-4	0.0000 ± 0.0000	0.6233 ± 0.0473	0.5680 ± 0.0383	0.0000 ± 0.0000	$\mathbf{0.7880 \pm 0.0390}$	$\mathbf{0.9867 \pm 0.0153}$

G-mean

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.0000 ± 0.0000	0.7653 ± 0.0419	0.5008 ± 0.0587	0.0000 ± 0.0000	$\mathbf{0.7184 \pm 0.0612}$	$\mathbf{0.7053 \pm 0.0430}$
abalone	0.0000 ± 0.0000	0.8049 ± 0.0079	0.8087 ± 0.0324	0.0000 ± 0.0000	0.8020 ± 0.0096	0.7850 ± 0.0140
car	0.0000 ± 0.0000	0.9629 ± 0.0028	0.9278 ± 0.0633	0.0000 ± 0.0000	0.9632 ± 0.0076	0.9660 ± 0.0075
nursery	0.0000 ± 0.0000	0.9742 ± 0.0065	0.9817 ± 0.0095	0.8403 ± 0.0320	0.9660 ± 0.0020	0.9760 ± 0.0042
letter	0.6804 ± 0.0316	0.9341 ± 0.0223	0.9484 ± 0.0195	0.8034 ± 0.0244	0.9351 ± 0.0115	0.9395 ± 0.0046
connect-4	0.0000 ± 0.0000	0.7306 ± 0.0282	0.7135 ± 0.0221	0.0000 ± 0.0000	$\mathbf{0.8007 \pm 0.0210}$	$\mathbf{0.8269 \pm 0.0351}$

AUC-ROC

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.5000 ± 0.0000	0.7734 ± 0.0384	0.8620 ± 0.0170	0.5000 ± 0.0000	0.7437 ± 0.0490	0.7332 ± 0.0316
abalone	0.5000 ± 0.0000	0.8156 ± 0.0100	0.8792 ± 0.0339	0.5000 ± 0.0000	0.8121 ± 0.0074	0.8054 ± 0.0102
car	0.5000 ± 0.0000	0.9636 ± 0.0027	0.9772 ± 0.0132	0.5000 ± 0.0000	0.9639 ± 0.0073	0.9666 ± 0.0073
nursery	0.5000 ± 0.0000	0.9744 ± 0.0065	0.9958 ± 0.0006	0.8530 ± 0.0264	0.9665 ± 0.0020	0.9763 ± 0.0041
letter	0.7315 ± 0.0213	0.9348 ± 0.0214	0.9915 ± 0.0050	0.8222 ± 0.0194	0.9351 ± 0.0114	0.9396 ± 0.0045
connect-4	0.5000 ± 0.0000	0.7402 ± 0.0238	0.8503 ± 0.0081	0.5000 ± 0.0000	0.8010 ± 0.0204	0.8210 ± 0.0203

TABLE VI. PERFORMANCE COMPARISON OF ALL APPROACHES USING K-NEAREST NEIGHBOR

Sensitivity

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.0533 ± 0.0298	0.6333 ± 0.0471	0.4667 ± 0.2055	0.0600 ± 0.0435	$\mathbf{0.7533 \pm 0.0767}$	$\mathbf{0.6600 \pm 0.0760}$
abalone	0.0000 ± 0.0000	0.8885 ± 0.0498	0.8308 ± 0.0896	0.0000 ± 0.0000	0.8846 ± 0.0360	$\mathbf{0.9423 \pm 0.0136}$
car	0.0429 ± 0.0639	0.7571 ± 0.1644	0.9286 ± 0.0505	0.0286 ± 0.0391	1.0000 ± 0.0000	1.0000 ± 0.0000
nursery	0.3121 ± 0.0409	0.9364 ± 0.0561	0.9727 ± 0.0166	0.3303 ± 0.0046	1.0000 ± 0.0000	1.0000 ± 0.0000
letter	0.7566 ± 0.0372	0.9118 ± 0.0374	0.8987 ± 0.0253	0.7250 ± 0.0225	$\mathbf{0.9211 \pm 0.0186}$	0.9189 ± 0.0166
connect-4	0.0360 ± 0.0134	0.7133 ± 0.0586	0.4940 ± 0.0654	0.0380 ± 0.0110	$\mathbf{0.6100 \pm 0.0436}$	$\mathbf{0.8833 \pm 0.0651}$

G-mean

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.2243 ± 0.0607	0.7547 ± 0.0268	0.6449 ± 0.1423	0.2144 ± 0.1318	$\mathbf{0.8272 \pm 0.0418}$	$\mathbf{0.7798 \pm 0.0405}$
abalone	0.0000 ± 0.0000	0.8301 ± 0.0198	0.8177 ± 0.0367	0.0000 ± 0.0000	0.8311 ± 0.0192	0.7841 ± 0.0127
car	0.1290 ± 0.1810	0.8093 ± 0.0729	0.9193 ± 0.0244	0.1069 ± 0.1464	0.8916 ± 0.0187	$\mathbf{0.9471 \pm 0.0071}$
nursery	0.5578 ± 0.0358	0.9262 ± 0.0206	0.9670 ± 0.0105	0.5741 ± 0.0309	0.9171 ± 0.0040	$\mathbf{0.9832 \pm 0.0025}$
letter	0.8683 ± 0.0213	0.9361 ± 0.0198	0.9366 ± 0.0122	0.8503 ± 0.0130	0.9350 ± 0.0102	0.9431 ± 0.0099
connect-4	0.1869 ± 0.0358	0.7097 ± 0.0295	0.6596 ± 0.0445	0.1932 ± 0.0276	$\mathbf{0.7153 \pm 0.0250}$	0.6911 ± 0.0184

AUC-ROC

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.8707 ± 0.0225	0.8670 ± 0.0228	0.8168 ± 0.0197	0.5290 ± 0.0214	$\mathbf{0.8854 \pm 0.0307}$	$\mathbf{0.8905 \pm 0.0168}$
abalone	0.8819 ± 0.0179	0.8760 ± 0.0338	0.8783 ± 0.0243	0.5000 ± 0.0000	0.8812 ± 0.0210	0.8725 ± 0.0089
car	0.9930 ± 0.0035	0.9543 ± 0.0129	0.9724 ± 0.0112	0.5143 ± 0.0196	$\mathbf{0.9932 \pm 0.0035}$	$\mathbf{0.9941 \pm 0.0043}$
nursery	0.9999 ± 0.0003	0.9875 ± 0.0073	0.9949 ± 0.0016	0.6652 ± 0.0173	$\mathbf{0.9994 \pm 0.0002}$	$\mathbf{0.9990 \pm 0.0004}$
letter	0.9872 ± 0.0051	0.9818 ± 0.0074	0.9808 ± 0.0077	0.8613 ± 0.0110	0.9860 ± 0.0062	$\mathbf{0.9887 \pm 0.0018}$
connect-4	0.7744 ± 0.0426	0.7732 ± 0.0185	0.7844 ± 0.0235	0.5183 ± 0.0054	$\mathbf{0.8262 \pm 0.0241}$	0.7824 ± 0.0349

TABLE VII. PERFORMANCE COMPARISON OF ALL APPROACHES USING LOGISTIC REGRESSION

Sensitivity

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.0867 ± 0.0447	0.5667 ± 0.0850	0.2933 ± 0.1657	0.1533 ± 0.0380	$\mathbf{0.4667 \pm 0.1179}$	$\mathbf{0.4867 \pm 0.1043}$
abalone	0.0000 ± 0.0000	0.9154 ± 0.0258	0.8962 ± 0.0322	0.0000 ± 0.0000	0.8923 ± 0.0421	0.9308 ± 0.0586
car	0.3429 ± 0.1174	0.9571 ± 0.0639	0.8714 ± 0.0782	0.3286 ± 0.1481	1.0000 ± 0.0	1.0000 ± 0.0000
nursery	0.7394 ± 0.0127	0.9424 ± 0.0628	0.9515 ± 0.0392	0.7576 ± 0.0557	1.0000 ± 0.0000	1.0000 ± 0.000
letter	0.6171 ± 0.0482	0.9000 ± 0.0471	0.9197 ± 0.0086	0.6566 ± 0.0205	0.9184 ± 0.0231	$\mathbf{0.9518 \pm 0.0249}$
connect-4	0.2920 ± 0.0785	0.6300 ± 0.0361	0.5600 ± 0.0543	0.3040 ± 0.0439	$\mathbf{0.8120 \pm 0.0327}$	$\mathbf{1.0000 \pm 0.0000}$

G-mean

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.2845 ± 0.0782	0.7186 ± 0.0503	0.5141 ± 0.1537	0.3861 ± 0.0497	$\mathbf{0.6596 \pm 0.0784}$	$\mathbf{0.6722 \pm 0.0778}$
abalone	0.0000 ± 0.0000	0.8441 ± 0.0184	0.8264 ± 0.0245	0.0000 ± 0.0000	0.8290 ± 0.0197	0.7971 ± 0.0496
car	0.5746 ± 0.0979	0.9543 ± 0.0322	0.9092 ± 0.0410	0.5556 ± 0.1362	$\mathbf{0.9747 \pm 0.0050}$	$\mathbf{0.9716 \pm 0.0040}$
nursery	0.8573 ± 0.0066	0.9600 ± 0.0309	0.9655 ± 0.0189	0.8679 ± 0.0319	$\mathbf{0.9860 \pm 0.0017}$	$\mathbf{0.9887 \pm 0.0013}$
letter	0.7826 ± 0.0305	0.9243 ± 0.0252	0.9329 ± 0.0076	0.8082 ± 0.0122	0.9270 ± 0.0114	0.9286 ± 0.0100
connect-4	0.5293 ± 0.0778	0.7383 ± 0.0234	0.7051 ± 0.0317	0.5438 ± 0.0409	$\mathbf{0.8046 \pm 0.0164}$	$\mathbf{0.8142 \pm 0.0451}$

AUC-ROC

Dataset	Baseline	SMOTE	SMOTEBoost	RUSBoost	RACOG	wRACOG
prompting	0.8714 ± 0.0173	0.8643 ± 0.0371	0.8404 ± 0.0246	0.5691 ± 0.0192	0.8269 ± 0.0482	0.8222 ± 0.0124
abalone	0.8914 ± 0.0082	0.8900 ± 0.0167	0.8756 ± 0.0191	0.4999 ± 0.2852	0.8914 ± 0.0149	0.8832 ± 0.0304
car	0.9778 ± 0.0031	0.9758 ± 0.0039	0.9652 ± 0.0054	0.6562 ± 0.0754	$\mathbf{0.9731 \pm 0.0080}$	$\mathbf{0.9741 \pm 0.0065}$
nursery	0.9954 ± 0.0007	0.9948 ± 0.0010	0.9923 ± 0.0009	0.8764 ± 0.0279	$\mathbf{0.9956 \pm 0.0007}$	$\mathbf{0.9948 \pm 0.0010}$
letter	0.9817 ± 0.0025	0.9782 ± 0.0055	0.9756 ± 0.0066	0.8258 ± 0.0096	0.9810 ± 0.0048	0.9787 ± 0.0065
connect-4	0.8769 ± 0.0086	0.8538 ± 0.0088	0.8320 ± 0.0157	0.6403 ± 0.0238	$\mathbf{0.8742 \pm 0.0276}$	$\mathbf{0.8840 \pm 0.0066}$

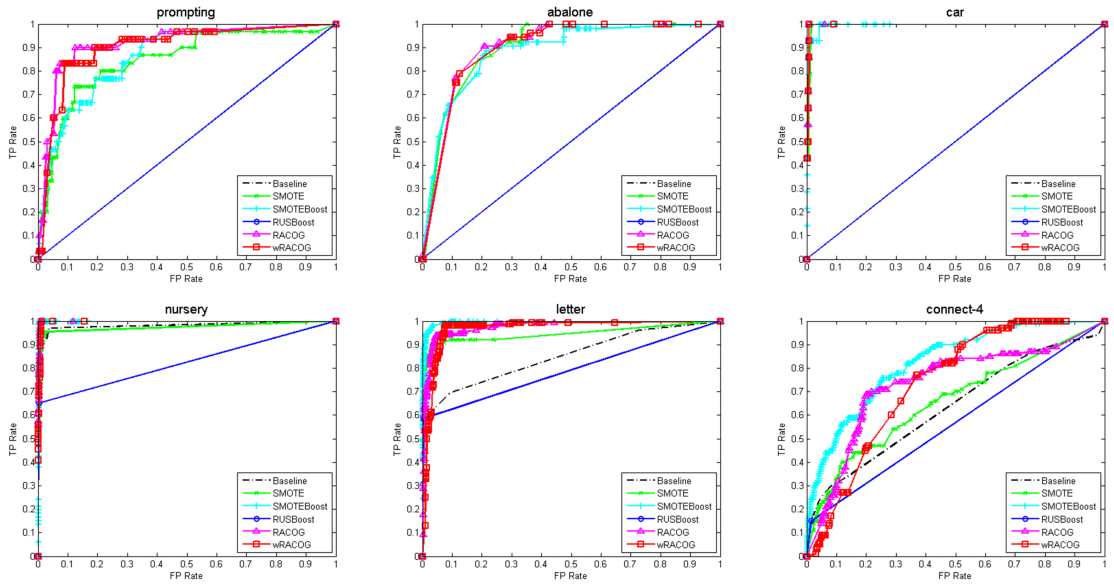

Fig. 9. ROC curves produced by C4.5 Decision Tree

to calculate *i-stat* (given in Equation 6) which measures the increase in the number of iterations due to dependence in the sequence (Markov chain). Raftery and Lewis indicate that *i-stat* is indicative of a convergence of the sampler if the value does not exceed 5.

$$i - stat = \frac{M + N}{N_{min}} \tag{6}$$

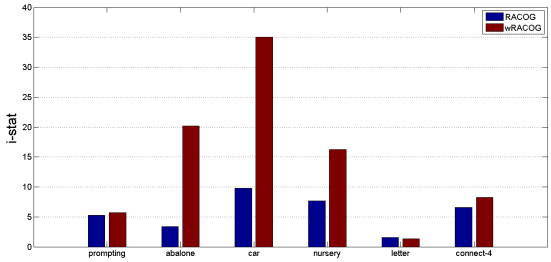

Fig. 10. *i-stat* values for RACOG and wRACOG

We perform the Raftery-Lewis test for the methods under consideration. From Figure 10 it is evident that convergence is almost achieved by both RACOG and wRACOG for the *prompting* and *letter* datasets. However, the wRACOG *i-stat* value is greater than the RACOG value for the remainder of the datasets. Although this indicates that wRACOG could not converge to the target minority class distribution for the *abalone*, *nursery* and *car* datasets, we have seen that wRACOG's sensitivity and G-mean are better or at par with RACOG. Therefore, one explanation is that convergence (not necessarily probabilistic) here is subjective as it is tightly coupled with the application.

We further analyze the convergence of RACOG and wRACOG in terms of the number of iterations the Gibbs sampler undergoes to achieve the *i-stat* value. From Figure 11 we notice that wRACOG generates much fewer (except

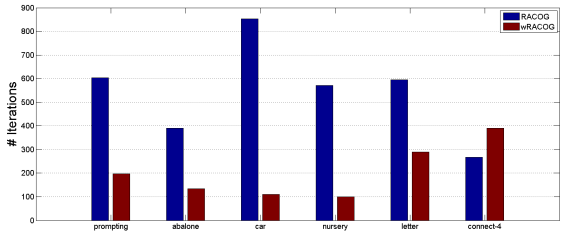

Fig. 11. Comparison of total number of iterations required by different methods to achieve given *i-stat*

for *connect-4* dataset) number of iterations ($\sim$ 63%) than RACOG. This explains the poor performance of wRACOG in terms of probabilistic convergence on the Raftery-Lewis test. However, the sensitivity and G-mean scores are not affected by the reduction in the number of iterations.

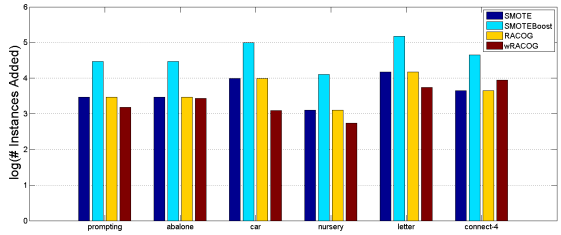

Fig. 12. Comparison of *log*(number of instances added) by different methods

The number of samples added to the baseline datasets by the different oversampling algorithms for achieving the reported performance is also an important parameter to analyze. Ideally, we would want to obtain a high performance by adding as less number of samples as possible. Figure 12 illustrates the number of samples added by the different approaches. As the number of samples added by SMOTEBoost is far higher than other methods, we present log_{10} values of the

number of added samples for better representation. SMOTE and RACOG try to achieve a 50:50 class distribution and thus add samples accordingly. SMOTEBoost requires ten boosting iterations to produce ten hypotheses on which weighted voting is performed while prediction. The hypothesis learned at each iteration $h_t : X \times Y \longrightarrow [0, 1]$, stores the samples generated by SMOTE at every iteration. Hence, SMOTEBoost adds ten times the number of samples added by SMOTE. Whereas, wRACOG requires a fraction ($\sim 56\%$) of the number of samples generated by SMOTE and RACOG, and ($\sim 5.6\%$) of the number of samples generated by SMOTEBoost, to obtain superior performance. We attribute this behavior to wRACOG's sample selection methodology which ensures the diversity in the samples that are added.

VII. Conclusion

In this paper, we propose a Gibbs sampling-based algorithm for generating new minority class samples for class imbalanced datasets. A Bayesian tree-based approach is used to impose dependencies among data attributes. wRACOG iteratively selects samples from the Markov chain generated by the Gibbs sampler that have the highest probability of being misclassified by a classifier. Experiments with wRACOG on a wide variety of datasets and classifiers indicate that the algorithm attains higher sensitivity than other methods, while maintaining higher G-mean. This supports our hypotheses that generating new samples by considering the distribution of minority class samples is a good approach to address class imbalance. However, the datasets used in the experiments do not have very high dimensions nor are the training samples in millions. In our future work, we will consider very large (in terms of dimension and cardinality) datasets. We will also take into account extreme class imbalance situations, as it occurs when the minority class is $< 1\%$ of the total number of samples.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant 1064628 and by the National Institutes of Health under application R01EB009675.

References

[1] D. A. Cieslak, N. V. Chawla, and A. Striegel, "Combating imbalance in network intrusion datasets," in *Proceedings of 2006 IEEE international conference on granular computing*, 2006, pp. 732–737.

[2] C. Phua, D. Alahakoon, and V. Lee, "Minority report in fraud detection: classification of skewed data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 50–59, 2004.

[3] M. Kubat, R. Holte, and S. Matwin, "Machine learning for the detection of oil spills in satellite radar images," *Machine learning*, vol. 30, no. 2, pp. 195–215, 1998.

[4] P. Turney *et al.*, "Learning algorithms for keyphrase extraction," *Information Retrieval*, vol. 2, no. 4, pp. 303–336, 2000.

[5] A. Liu, J. Ghosh, and C. Martin, "Generative oversampling for mining imbalanced datasets," in *Proceedings of the 2007 International Conference on Data Mining, DMIN*, 2007, pp. 25–28.

[6] B. Zadrozny, J. Langford, and N. Abe, "Cost-sensitive learning by cost-proportionate example weighting," in *Third IEEE International Conference on Data Mining, 2003. ICDM 2003*. IEEE, 2003, pp. 435–442.

[7] C. Elkan, "The foundations of cost-sensitive learning," in *International Joint Conference on Artificial Intelligence*, vol. 17, no. 1. Lawrence Erlbaum Associates Ltd., 2001, pp. 973–978.

[8] K. McCarthy, B. Zabar, and G. Weiss, "Does cost-sensitive learning beat sampling for classifying rare classes?" in *Proceedings of the 1st international workshop on Utility-based data mining*. ACM, 2005, pp. 69–77.

[9] G. Weiss, "Mining with rarity: a unifying framework," *SigKDD Explorations*, vol. 6, no. 1, pp. 7–19, 2004.

[10] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, June 2002.

[11] N. Chawla, A. Lazarevic, L. Hall, and K. Bowyer, "SMOTEBoost: Improving prediction of the minority class in boosting," *Knowledge Discovery in Databases: PKDD 2003*, pp. 107–119, 2003.

[12] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 40, no. 1, pp. 185–197, 2010.

[13] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[14] N. Chawla, "Data mining for imbalanced datasets: An overview," *Data Mining and Knowledge Discovery Handbook*, pp. 875–886, 2010.

[15] X. Liu and Z. Zhou, "The influence of class imbalance on cost-sensitive learning: an empirical study," in *Data Mining, 2006. ICDM'06. Sixth International Conference on*. IEEE, 2006, pp. 970–974.

[16] G. Wu and E. Chang, "Kba: Kernel boundary alignment considering imbalanced data distribution," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 786–795, 2005.

[17] G. Fung and O. Mangasarian, "Multicategory proximal support vector machine classifiers," *Machine Learning*, vol. 59, no. 1, pp. 77–97, 2005.

[18] F. Provost, T. Fawcett, and R. Kohavi, "The case against accuracy estimation for comparing induction algorithms," in *Proceedings of the Fifteenth International Conference on Machine Learning*, vol. 445, 1998.

[19] H. Han, W. Wang, and B. Mao, "Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning," *Advances in Intelligent Computing*, pp. 878–887, 2005.

[20] H. He, Y. Bai, E. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *IEEE International Joint Conference on Neural Networks, 2008 (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 1322–1328.

[21] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," in *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*. IEEE, 2009, pp. 324–331.

[22] N. Meteopolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[23] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, p. 1087, 1953.

[24] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2, pp. 131–163, 1997.

[25] C. Geyer, "Practical markov chain monte carlo," *Statistical Science*, vol. 7, no. 4, pp. 473–483, 1992.

[26] V. García, R. Mollineda, and J. Sánchez, "On the k-nn performance in a challenging scenario of imbalance and overlapping," *Pattern Analysis & Applications*, vol. 11, no. 3, pp. 269–280, 2008.

[27] G. Weiss and F. Provost, "Learning when training data are costly: The effect of class distribution on tree induction," *Journal of Artificial Intelligence Research (JAIR)*, vol. 19, pp. 315–354, 2003.

[28] T. Khoshgoftaar, C. Seiffert, J. Van Hulse, A. Napolitano, and A. Folleco, "Learning with limited minority class data," in *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*. IEEE, 2007, pp. 348–353.

[29] A. Raftery and S. Lewis, "How many iterations in the gibbs sampler," *Bayesian statistics*, vol. 4, no. 2, pp. 763–773, 1992.

[30] J. LeSage and R. Pace, *Introduction to spatial econometrics*. Chapman & Hall/CRC, 2009, vol. 196.