New Cluster Undersampling Technique for Class Imbalance Learning

Robert A. Sowah, Moses A. Agebure, Godfrey A. Mills, Koudjo M. Koumadi, and Seth Y. Fiawoo

***Abstract*—Adequately learning from datasets that are highly imbalance has become one of the most challenging tasks in Data Mining and Machine Learning disciplines. Most datasets from high risk application areas are often adversely affected by the class imbalance problem due to the limited occurrence of positive examples. This paper presents a new undersampling technique, called Cluster Undersampling Technique (CUST) that has the capability of further improving the performance of classification algorithms when learning from imbalance datasets. The performance of CUST is evaluated by using it to undersample 16 real world class imbalance datasets prior to building classification models using C4.5 decision tree and OneR algorithms. The performance of the models are compared to the performance of random undersampling and oversampling, synthetic minority oversampling, one-sided selection, and cluster-based undersampling. The experimental results using area under receiver-operating characteristic curve and geometric mean showed that CUST resulted in higher performance and is statistically better compared to well-known techniques.**

***Index Terms*—Class imbalance, classification, clustering, oversampling, undersampling.**

I. Introduction

Classification is a well-studied technique in the machine learning and data mining communities [1]-[3]. Classification involves learning from existing data with known classes to predict or classify incoming instances or instances with unknown classes as belonging to a particular class or sub-concept. Classification has been widely used in real world applications due to its predictive or forecasting nature. For instance, a medical prediction model or system can be built using medical datasets that are based on the symptoms of patients who suffered from a particular ailment and used to predict if a new patient arriving at the hospital is suffering from that ailment or not, given the symptoms exhibited by the new patient.

Classification is data dependent and therefore, the nature or quality of the training data plays a key role on the prediction accuracy of classification models. That is, the performance of classification models are often influenced negatively by several data quality issues when present in the training data in substantial amounts of which noisy instances [4], [5], outliers [6], and class imbalance [7]-[9] are examples.

Noise and outliers in most cases are erroneously created. Class imbalance on the other hand is a major data intrinsic problem that plaques so many high risk application domains including software defect prediction or quality estimation [10]-[12], medical datasets [13], fraud detection [14], intrusion detection [15], and risk management [16].

Class imbalance unlike noise and outliers is not a problem that arises potentially from mechanical errors or deficiencies in the data generation processes but occurs due to the nature of most application domains and therefore cannot be avoided. For example, standard software under development is generally expected to have minimal faults or defects. For instance, data generated from this software during testing based on its modules will likely yield results from the modules with defects labelled as positive instances and those without defects as negative instances. Based on this, it is fairly obvious that, the positive instances would constitute a smaller percentage of the data generated as compared to the negative instances. In some cases the difference between the number of positive and negative instances could be very high such that the positive instances constitute less than 1% of the dataset [11]. This phenomenon referred to as class imbalance or unbalanced datasets is reported to undermine the learning ability of most classification algorithms and as a result they tend to predict that new datasets have only the negative class instances and thereby incorrectly classifying the positive class instances, which are of interest [9]. Class imbalance is however, predominant in high risk application domains as mentioned above, and in these areas wrongly classifying a positive instance comes with a higher cost as compared to a negative instance [17].

As a result of the above stated challenges, researchers have explored the use of several class imbalance learning techniques with the goal of improving the performance of classification models when learning from imbalance datasets. Common among these is the data level or data sampling approach. Data sampling involves either increasing the size of the minority class instances, oversampling, or reducing the size of the majority class instances, undersampling. Some undersampling techniques contrary to the process adapted by the random undersampling technique targets the removal of potential problematic majority class instances such as inconsistent/noisy and redundant instances as in One-Sided Selection (OSS) [18] and majority class instances in overlap regions as in Cluster-Based Undersampling technique (CBU) [19], with the main goal of keeping only relevant majority instances for training.

Manuscript received March 11, 2015; revised May 27, 2016. This work was supported in part by the Carnegie Corporation of New York through the University of Ghana, under the UG-Carnegie "Next Generation of Academics in Africa" Project.

R. A. Sowah is with the Computer Engineering Department, University of Ghana, Ghana (e-mail: rasowah@ug.edu.gh, bobsowah@gmail.com).

M. A. Agebure is with the Computer Science Department, University for Development Studies, Ghana (e-mail: magebure@gmail.com).

However, to the best of our knowledge at the time of this research, none of the existing undersampling techniques explicitly targets the removal of outliers as part of the sampling process. It is demonstrated by Acuña and Rodríguez [6] that the presence of outliers in training data increases misclassification error rates of classifiers. In this work a new undersampling technique called Cluster Undersampling Technique (CUST) is presented. CUST targets the removal of repeated and noisy instances as well as outliers or instances that behave like outliers from the majority class as a form of undersampling. This technique has been demonstrated to improve the performance of commonly used classification algorithms such as the C.5 decision tree and OneR. It is better than most well-known sampling techniques using imbalance datasets from the NASA MDP repository [20] and UCI repository [21].

The remaining part of the paper is organised as follows, Section II covers a review of related work, Section III details the proposed undersampling technique, the experimental setup is outlined in Section IV, Section V provides the experimental results and discussions, and conclusion and relevant recommendations are presented in Section VI.

II. Related Work

Due to the challenges of adequately learning from imbalance datasets and the critical nature of areas where it is prevalent, several imbalance learning approaches have been proposed in literature and targeted at improving the correct classification of minority class instances while minimising false alarm rates. These approaches according to [22] can be grouped into; data sampling, kernel-based learning, cost-sensitive learning, active learning, and one-class learning. A comprehensive review of these techniques is presented in [23]

Among these imbalance learning approaches, the data sampling approach is one of the most preferred approach because instead of tuning parameters of classification algorithms to maximise the detection of minority class instances, the training data is rather manipulated. This serves as leverage to researchers who do not have the technical expertise to tune the classification algorithms in the learning process. Data sampling techniques are categorised primarily into oversampling and undersampling techniques. Oversampling increases the number of minority instances in the training data either randomly or synthetically, whiles undersampling discards some majority instances from the training dataset with the core aim of minimising the difference in numbers between the majority and minority instances.

Random Sampling Techniques: The simplest and most common among the sampling techniques are random oversampling (ROS) and random undersampling (RUS). ROS randomly replicates the minority class instances thereby increasing the minority class population in the training data; RUS on the other hand randomly discards some majority class instances hence, reducing the size of the majority class in the training data [22]. RUS and ROS have their respective pros and cons. Though these methods are the easiest to use and much faster than other techniques, a major problem associated with RUS is the loss of information that occurs due to the random removal of instances from the training data and this problem aggravates when the training datasets are small, this however is not a problem with oversampling, but rather, the increase in size of the resulting training dataset after oversampling can lead to longer training time and also, the fact that instances are randomly replicated, which adds no new information may result to overfitting [11].

Synthetic Minority oversampling Technique (SMOTE) [17] oversamples a dataset by creating synthetic minority instances and not by duplicating already existing minority class instances as done in random oversampling. SMOTE was introduced to minimise the problems associated with random oversampling. It finds the k nearest neighbours of each instance in the minority class and then generates synthetic instances in the direction of some or all of the nearest neighbours of an instance depending on the percentage of oversampling required. In the original work [17], $k = 5$ was used. To create a synthetic instance, the difference between the feature vector of the minority instance under consideration and its nearest neighbour is calculated and multiplied by a random number between 0 and 1. The product is then added to the feature vector of the minority instance under consideration to form the new instance. Though SMOTE has been used extensively in various fields of application with good results, its mode of creating the synthetic instances is sometimes problematic particularly in very skewed dataset as it blindly generalizes the regions of the minority class without regard to the majority class instances, this depending on the sparseness of the minority instances in the dataset may led to an increase in class overlapping [24].

The Cluster-Based Undersampling (CBU) technique proposed by Das *et al.* [19] is aimed at solving the class imbalance problem by discarding majority instances in overlap regions of the training data. This is achieved by clustering the training dataset into some k clusters and discarding all majority class instances from clusters which satisfy $0 < r < 1$ and $r \geq \tau$.
where

$$r = \frac{c^i}{|C|}, \quad 0 < i \leq k \tag{1}$$

c^i is the number of minority instances in cluster i, $|C|$ is the total number of instances in cluster i, and τ is an empirically determined value. They argued that doing this will minimise the class overlap problem whilst reducing the number of majority class instances, hence minimising the imbalance in classes. CBU, though is strategically designed to minimise the class overlapping problem as oppose to SMOTE, also has a limitation when sampling from much skewed datasets as it fails to adequately reduce the number of majority class instances. For example, the PC2 dataset considered in this study have approximately 14 minority and 1412 majority instances in the training set after stratification. Clustering this data may result in a lot of clusters having only majority instances, this implies that the majority instances in these clusters would be retained as per the rational of the technique and the few clusters that contain minority instances would be discarded depending

on the τ value considered. In this case only a small percentage of the majority class instances would be discarded.

Sampling techniques such as RUS do not take other data quality issues such as the presence of inconsistent/noisy instances, redundant instances, and outliers, which have the potential of affecting the accurate estimation of the generalisation of classification models, into consideration when sampling. These problems are inherent in most real world datasets as they may be created erroneously or arise due to the nature of the application domain [5], [6]. It may be argued that these problematic instances can easily be removed during data cleansing processes, but depending on the nature of the data at hand, blindly removing instances from the dataset may further worsen the class imbalance problem since the minority class instances may also be affected. OSS undersampling technique as proposed by Kubat and Matwin [18], targets the removal of some of these problematic (noisy/borderline and redundant) instances from the majority class as a form of undersampling. This, they suggest would reduce the number of majority class instances and thereby minimize the difference in class distribution. OSS undersamples a dataset S by first moving all minority instances and one randomly selected majority instances into a dataset C. C is used to train a 1-NN rule, which is used to classify the instances in S and all misclassified instances are moved from S to C. C is now consistent with S but smaller in size. All majority class instances in C that participate in Tomek links are further removed, this process removes noisy and borderline instances. The remaining instances in C are then considered as the final training set. OSS just like the other undersampling techniques however, does not explicitly avoid the inclusion of outliers in the final training set. Also, the performance of OSS in most empirical studies, particularly in software defect prediction using the NASA MDP datasets where the presence of noise and repeated instances is evident, is not encouraging as it has been consistently outperformed by other techniques such as RUS and SMOTE [4], [25], and [26].

III. The Proposed Technique

Cluster Undersampling Technique (CUST) seeks to improve the performance of classification algorithms when learning from imbalance datasets by undersampling the majority class instances in the training set taking into account the presence of inconsistent instances, repeated instances, and outliers in the majority class. The hypothesis therefore is that, eliminating inconsistent, repeated instances and outliers in the course of undersampling will yield a subsample of the majority class instances that is void of problematic instances that have the potential of negatively affecting a classifier's performance.

The CUST undersampling process is in two stages, the first stage is the removal of inconsistent instances from the majority class, and the second is the stage where outliers and repeated instances are removed along with other instances.

The first part of the process uses a technique derived from Tomek links [27] used in OSS to remove noise and borderline instances. Tomek links are based on the closeness of two instances belonging to different classes. Given two instances I_j and I_k belonging to different classes, and the distance between them $dist(I_j, I_k)$, if there is no other instances I_l such that $dist(I_j, I_l) < dist(I_j, I_k)$ then the pair (I_j, I_k) is called a Tomek link. If two instances form a Tomek link, one of them is either noisy or both are on or near the class boundary.

This implies that for the instances to form a Tomek link, $dist(I_j, I_k)$ must not necessarily be equal to zero, however, the criterion used in the proposed technique, contrary to that used in OSS, only allows a majority class instance to be discarded if and only if $dist(I_j, I_k) = 0$. That is, the instances must have exactly the same attributes but different class labels. Thus, only inconsistent or noisy majority class instances are removed. The minority instances are not discarded because doing so will further worsen the class imbalance problem.

The call of the function that performs the removal of inconsistent instances is optional, in that, if the experimenter is certain that the dataset under consideration do not contain inconsistent instances the option can be set to false and the process begins from the second stage, this is to avoid unnecessary wastage of computational resources and time.

After the removal of inconsistent instances from the majority class or the call of the function being set to false, the second phase of undersampling process starts with the remaining data passed through from the first stage. The second phase is based on the idea of using clustering to identify outliers in datasets [6], [28], and [29]. The main assumption of this technique is that, clustering the majority class instances in the training set into k clusters, instances that portray similar characteristics would fall naturally into the same clusters with the outliers either forming smaller clusters or single instance clusters away from the main clusters.

Clustering of instances is done using the simple k-means clustering algorithm. The default number of clusters in CUST is set to five (5) but the experimenter has the option to change it to any value, k, that is desired. The number of instances to sample from each cluster is determined using (2) below:

$$Maj^i = r \times \frac{MI}{MA} \times MC^i \;; 1 \le i \le k, ?MA \ne \qquad (2)$$

where Maj^i is the number of instances to sample from cluster i calculated to the nearest whole number, MI is the total number of minority class instances in the training set, MA is the total number of majority class instance after inconsistent instances are removed in the first stage, MC^i is the number of instances in cluster i, and r is a parameter that specifies the ratio of majority to minority instances in the final training set. If $r = 0.75$ the sampled majority instances would be $\approx 25\%$ less than the minority instances, if $r = 1$, the ratio is $\approx 1:1$ meaning they are approximately equal in number, and if $r = 2$, the ratio would be $\approx 2:1$, meaning the majority instances sampled are twice as much as the minority instances. The default value of r is set to 1 in the algorithm but, since the best ratio of

majority/minority instances that yields best classifier performance is not always 1:1 [11], the experimenter has the option to vary the value of r until the performance of the classifier is maximized. The optimum value of r depends on the number of clusters, dataset, and classification algorithm at hand. Equation (2) by its nature discriminates against undersampling from clusters that have smaller number of instances, thereby limiting the number of outliers, which potentially resides in these smaller clusters from been included in the sampled data.

After determining the number of instances to sample from a cluster using (2), the instances in the cluster are randomized using a random seed that is provided by the experimenter and the instance that comes first is selected. All duplicates of this instance in the cluster are discarded and the remaining instances randomized. The instance that comes first is selected again and all its duplicates discarded. This process is repeated until the required number of instances to be sampled from the cluster is realized or there are no instances left in the cluster. Discarding all the occurrence of an instance in a cluster after it has been selected is aimed at avoiding duplicates of the instance from been selected, thereby eliminating the occurrence of repeated majority class instances in the final training set. This process is carried out in all the clusters and the selected instances put together to form the set of majority class instances in the final training set. The process used to sample instances from each cluster is much like that of random undersampling but it performs an additional task of discarding all duplicates of an instance after it has been selected. The algorithm of CUST is shown in Table I below.

TABLE I: ALGORITHM OF CUST

1.	Using Tomek links remove all inconsistent majority class instances if any from the training set.
2.	Cluster the remaining majority class instances into k clusters using k-means clustering algorithm.
3.	Determine the number of instances to sample from each cluster using equation 2 above.
4.	For each cluster i in k, if $Maj^i \geq 1$, randomise the instances and select the first instance. Discard all duplicates of the selected instance from the cluster.
5.	Repeat step four (4) for all clusters until the number of instances required from each cluster is realized or there are no instances left in the cluster.
6.	Add all the instances selected from the clusters to the minority class instances to form the new training set.

1. Separate the training data into minority and majority groups;
Min containing minority instances, and
Maj containing majority instances
let MI=number of instances in Min, and
MA= number of instances in Maj

2. removeInconsistent (True/false) (function to remove inconsistent instances*)*
If removeInconsistent=True
for i←1 to MI
for j←1 to MA
if EuclideanDistance(〖Min〗_i, 〖Maj〗_j)==0 then remove 〖Maj〗_j from Maj
endif
endfor
endfor
Maj=remaining Maj; MA=number of instances in Maj
endif
*return (*end of removeInconsistent*)*

3. Cluster Majority instances in Maj into k clusters using k-means algorithm
4 for i←1 to k (extract data in cluster i *)*
〖MC〗^i=number of instances in cluster i
*Compute: 〖Maj〗^i=r×MI/MA× 〖MC〗^i to the nearest whole number (*number of instances to sample from cluster i*)*
While 〖Maj〗^i>0
If 〖MC〗^i=0
Break
endif
rand: randomize instances in cluster i
push 〖Instance〗_1 (Add first instance to sampled data*)*
*for j←1 to 〖MC〗^i-1 (*Remove all repetition of selected instance from cluster*)*
if EuclideanDistance(〖Instance〗_1, 〖Instance〗_(j+1))==0
remove 〖Instance〗_(j+1) from cluster
endif
endfor
〖MC〗^i=number of instances remaining in cluster i
〖Maj〗^i--
endwhile
endfor
*return(*end of doCluster*)*
End of algorithm

TABLE II: SUMMARY OF DATASETS

Dataset	Num. of Attrib.	Num. of instances	Pos. instances		Neg. instances	
			Num.	%	Num.	%
CM1	38	344	42	12.21	302	87.79
KC1	22	2096	325	15.51	1771	84.49
KC3	40	200	36	18.00	164	82.00
MC1	39	9277	68	0.73	9209	99.27
MC2	40	127	44	34.65	83	65.35
MW1	38	264	27	10.23	237	89.77
PC1	38	759	61	8.04	698	91.96
PC2	37	1585	16	1.01	1569	98.99
PC3	38	1125	140	12.44	985	87.56
PC4	38	1399	178	12.72	1221	87.28
Abalone9v18	9	731	42	5.75	689	94.25
Abalone19	9	4177	32	0.76	4145	99.24
Ecoli4	8	336	20	5.95	316	94.05
Glass2	10	214	17	7.94	197	92.06
Yeast2v8	9	264	20	7.58	244	92.42
unbalanced	33	856	12	1.40	844	98.60

IV. EXPERIMENTAL SETUP

A. Datasets

The sixteen imbalance datasets used in this research are sourced from two publicly available data repositories; ten (10) software defect datasets from National Aeronautics and Space Administration (NASA) Metric Data Program (MDP) [20], and five (5) other datasets from University of California, Irvine repository [21]. The last dataset is from WEKA sample datasets that is distributed with it [30].

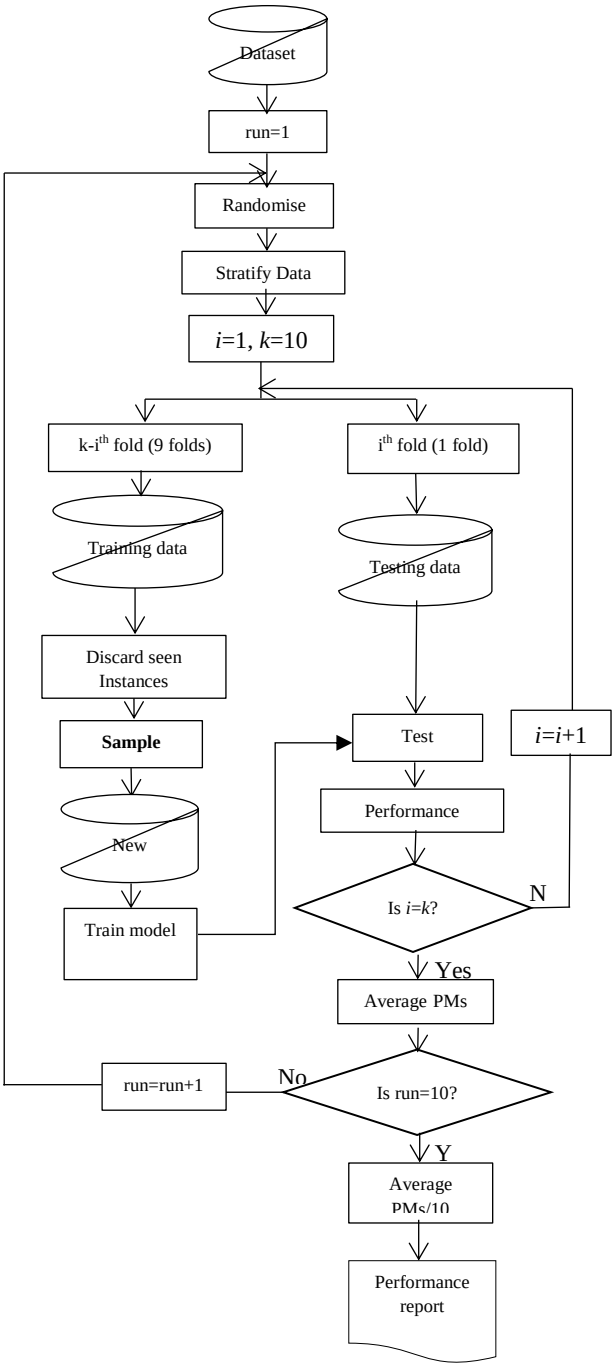

Fig. 1. Framework of experimental process.

The above framework was followed in the implementation of the CUST algorithm for class imbalance learning. Several datasets suited for the class imbalance learning were utilized for the experimental setup.

The NASA datasets were obtained from various software projects carried out at NASA through their Metric Data Program (MDP). This research adapts the cleansed version of the datasets labelled D' by Shepperd *et al.* [31] in order to allow easy verification of the results and reproducibility of the experiments since the original version of the datasets contain some problems such as implausible and missing values.

The datasets sourced from the UCI repository were originally multiple class datasets but are modified into binary class problems for the purpose of imbalance learning. The class combinations used in this study are used by other researchers for various research works involving imbalance learning [32], [33]. Table II shows a summary of the datasets indicating the number and percent of majority and minority class instances.

B. Sampling Techniques

Besides the proposed undersampling technique five (5) other data sampling techniques are considered in this study. These techniques are used as the basis to assess the efficiency of the proposed technique. They include, CBU, RUS, ROS, SMOTE and OSS. For the purpose of this study CBU, RUS, ROS, and OSS are implemented within the framework of the WEKA machine learning tool using NetBeans java IDE. Also, the java archive (jar) file of SMOTE created by Chawla *et al.* [17] and made available online for academic use is used.

Since the right proportion of majority instances to undersample or the proportion of minority class instance to oversample is not known a prior, several sampling parameters are used and the best chosen. Six undersampling parameters are considered for RUS in this study, 5, 10, 25, 50, 75, and 90. This implies that, if RUS is done with parameter 25, then only 25% of the majority class instances are retained for training. For ROS and SMOTE, seven parameters are considered, 50, 100, 200, 300, 500, 750, and 1000. When oversampling is done with a parameter 100, it means that the number of minority class instances is increased by 100%. These parameters were used by some previous studies including [11], [25].

In addition, five ($k = 5$) nearest neighbours is considered for SMOTE. Twelve clustering parameters $k = (5, 10, 15, 30, 50, 55, 70, 75, 100, 110, 120, and\ 130)$ are considered for CBU and the proposed technique, CUST. The wide range of clustering parameters is considered because of the variation in size of the datasets. Also, ten different threshold values:

$\tau = (0.1, 0.09, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01)$ are used for CBU, these values were used in the original work by Das *et al.* [19] and ten (10) r values:

$r = (1.0, 1.25, 1.50, 1.75, 2.0, 2.5, 3.0, 10.0, 25.0, 50.0)$ are considered for CUST. A sampling parameter for each sampling technique that yields the best results per each dataset and classification algorithm is considered and the performance of the given classification algorithm recorded for further analysis.

C. Classification Algorithms

C4.5 Decision tree [34] is a tree based learner that improves upon the ID3 [35] learner by adding support for handling missing values, numeric attributes, and tree pruning (a measure adopted to avoid overfitting). It creates classification models using a statistical property called information gain that measures the effectiveness of an attribute to separate the training instances according to their target classification. Information gain is based on entropy, a measure used in information theory to characterise the purity or impurity of an arbitrary collection of examples [36]. J48 is the WEKA implementation of the C4.5 Decision tree. The default values of the J48 learner specified in the WEKA machine learning tool are used for all experiment namely a confidence factor of 0.25 and a minimum of 2 instances per leaf node.

OneR (1R) [37] is a rule based learner that ranks all attribute with respect to error rates on the training data as opposed to the use of entropy based measures in C4.5. It creates one rule for each attribute value. A rule states that for a given attribute *A* and value *V* the majority class is *C*. The rules that have the highest accuracy on the training data are applied to a hypothesis and those with accuracy below just choosing the majority class instances are pruned from the hypothesis. The final rules are then sorted in order of accuracy in the training data. It treats all attributes with numeric values as continuous and employs a rather straightforward method to divide the range of values into several disjoint intervals. It also treats missing values as legitimate values. The default values (bucket size of 6 instances per rule) of the OneR learner specified in the Weka tool are used for all experiments.

D. Performance Metrics

The determination of how well a classifier will generalises when deployed in real-world setting is often done using a testing dataset that is independent of the training data. The performance of the classifier based on this testing data serves as an approximation of its general performance. For a binary classification problem, there is always one of four possible outcomes each instance predicted in a classification experiment will belong [38]: True Positives (TP), positive examples that are correctly predicted; True Negatives (TN), negative examples that are correctly predicted; False Positives (FP), negative examples that are incorrectly predicted as positive examples; and False Negatives (FN), positive examples that are incorrectly predicted as negative examples.

These measures are often used to form a confusion metrics that describes the basic class specific predictions of the classifier [39]. Using this confusion metrics almost all existing classification performance metrics can be derived. Accuracy and error-rates are the commonly used performance metrics in machine learning, but it is demonstrated in prior studies [11], [38] that these measures are not suitable when learning from class imbalance datasets and are therefore ideal for datasets with even class distribution. Area Under receiver-operating characteristic Curve (AUC) and Geometric Mean (G-Mean) are the main performance metrics used in this research.

AUC provides a single measure from a Receiver-Operating Characteristic (ROC) curve. The AUC measure is independent of a selected decision threshold and thus gives a classifier's performance without considering prior probabilities or misclassification cost. These attributes of AUC makes it a good metric for classifier performance under unbalanced datasets [40]. The AUC has a value between 0 and 1, with 0 indicating the worst performance and 1 the highest performance of a classifier and is given as:

$$AUC = \frac{1 + TPR - FPR}{2} \quad (3)$$

where TPR is True Positive Rate and FPR is False Positive Rate.

Kubat and Matwin [18] also suggested the G-Mean as a good metric for classification problems involving class imbalance. The G-Mean like the AUC provides a single numeric measure of a classifier's performance using its TPR and True Negative Rate (TNR). It gives a value between 0 and 1, with 0 indicating worse performance and 1 highest performance. It is calculated as:

$$G - Mean = \sqrt{TPR \times TNR} \quad (4)$$

E. Experimental Method

Besides sampling the training data prior to training the classification models, models are also trained without sampling the training data and this is referred to as NONE.

The models are built and evaluated using the stratified tenfold cross validation process. That is, each dataset is divided into ten equal folds. For each of the ten experiments, nine of the folds are used as training set and one fold is kept as the test set. Each fold is used once as a test set in the cross validation process. In each case the sampling techniques are not applied to the test set (i.e., the test set is not modified in anyway).

The tenfold cross validation process is also repeated ten times and at each run the dataset is randomized. This repetition eliminates any biasing that might be introduced by the sampling or the stratification process. The average performance of the 10×10 cross validation process (100 experiments) is calculated and considered the performance of the respective classification model given a sampling technique and dataset.

V. Results and Discussion

This section presents the results of the experiments carried out. The performance based on AUC and G-Mean values are presented, and that of other performance metrics intermittently referenced are detailed in [41]. First the performance of the classification models when CUST is used to undersample the training data is compared to the performance of the models when the training data is not sampled prior to training labelled as NONE. Secondly, the performance the models when CUST is used to undersample the training data is compared to the performance of the models when the other five sampling techniques are used. The last section presents an ANOVA analysis of the results to establish if there is significant difference in the performance of the classification models when the various sampling techniques are used.

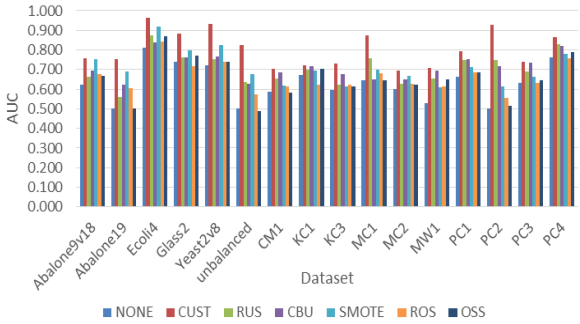

Fig. 2. Plot of CUST vs other Techniques for C4.5, AUC.

Fig. 2 and 3 illustrates the performance of the C4.5 classification models in terms of AUC and G-Mean metrics, respectively, across the sixteen datasets given the various sampling techniques. Figs. 4 and 5 also illustrate the AUC and G-Mean values, respectively of the OneR classification

models, given the various sampling techniques and sixteen datasets.

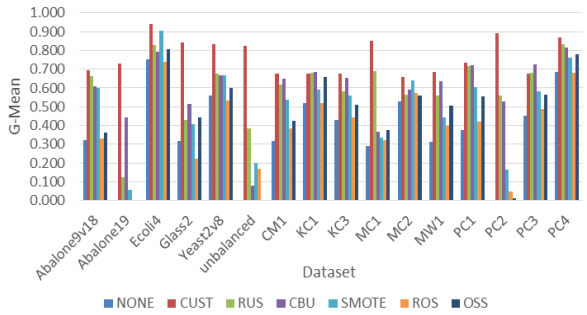

Fig. 3. Plot of CUST vs other techniques for C4.5, G-Mean.

Considering the AUC and G-Mean metric values from Fig. 2 and 3 respectively, for CUST and NONE, it is evident that CUST significantly improved the classification performance of the C4.5 decision tree across all the sixteen datasets and the improvement is however, higher in the datasets that are most skewed (PC2, MC1, abalone19, and unbalanced). The classifier recorded recall values of 0.00 in PC2, abalone19, and unbalanced datasets [41], when the data is not sampled prior to training. This implies that, the classifier failed to correctly classify a single minority class instance in these datasets when the training data is not sampled.

The performance of the OneR learner shown in Figs. 4 and 5 follow similar trends as the C4.5 decision tree. It is clear from these figures that CUST improved the performance of the learner in all the datasets for both AUC and G-Mean metrics with the exception of yeast2v8 dataset in which a lower G-Mean value is recorded by the OneR classification model when the training data is sampled using CUST. That is, the performance of the classification model dropped from 0.670 for NONE to 0.655 for CUST representation a decline of 2.22% in performance.

Comparing the performance of CUST to the other sampling techniques, from Fig. 2, it is clear that CUST outperformed all the other sampling techniques across all the datasets, though the difference between CUST and some of the other sampling technique(s) for some of the datasets is marginal. The performance of the C4.5 algorithm using G-Mean measure as illustrated in Fig. 3 indicates that CBU and RUS marginally outperformed CUST in only two (2) (KC1 and PC3) datasets out of the sixteen datasets considered. Also, the performance of the C4.5 algorithm in the MC1, PC2, abalone19, and unbalanced datasets, suggest that CUST is much robust when handling much skewed datasets as compared to the other sampling techniques as their performance dropped considerably in these datasets using both AUC and G-Mean performance metrics.

The performance of the OneR learner in terms of AUC for the various sampling techniques across the sixteen datasets illustrated in Fig. 4, also points out that CUST outperformed all the other sampling techniques in all datasets with the exception of CBU that performed relatively better in glass2, CM1 and KC1 datasets. The performance of the OneR learner using G-Mean measure shown in Fig. 5 indicates that CUST performed better than the other sampling techniques in fourteen out of the sixteen datasets. CBU outperformed CUST in the KC1 dataset by a difference of 0.024 and OSS outperformed it in the yeast2v8 dataset. Comparatively, the performance of the learner is relatively poor in the abalone19, glass2, unbalanced, MC1 and PC2 datasets when the other sampling techniques are used, particularly ROS, SMOTE, and OSS. Noticeable are the results of OSS, ROS, and SMOTE, where they recorded approximately 0.00 G-Mean values in these datasets. Also, CBU witness its lowest performance in the abalone19 dataset followed by the MC1 dataset. This to a greater extent confirms the accession made above that, CUST is more robust in improving the performance of learners when learning from much skewed datasets than the other sampling techniques.

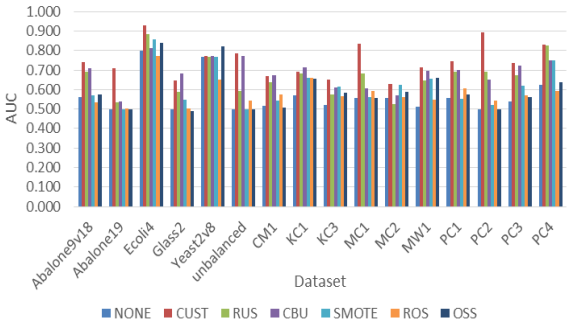

Fig. 4. Plot of CUST vs other techniques for OneR, AUC.

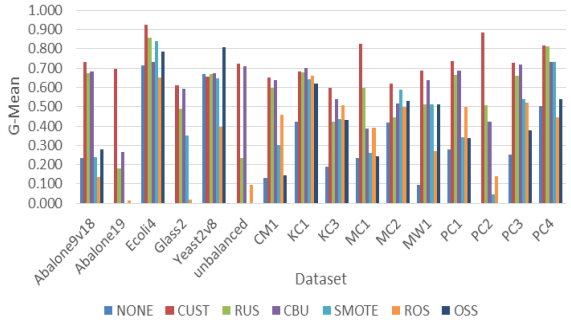

Fig. 5. Plot of CUST vs other techniques for OneR, G-Mean.

The statistical significance of the results presented above is examined using the ANOVA model specified in [41]. ANOVA models of the performances of the C4.5 decision tree using AUC and G-Mean measures are shown in row two (2) and three (3) respectively of Table III. The models of the OneR learner are shown in row four (4) and five (5) for AUC and G-Mean measures, respectively.

The results of the ANOVA models shown in Table III all have p-values equal to 0.000 (p=0.000).

TABLE III: ANOVA MODELS OF C4.5 AND ONER ALGORITHMS (AUC AND G-MEAN)

Experiment	DF	SS	MS	F Value	P
C4.5 AUC	6	0.315612	0.052602018	6.97	0.00
C4.5 G-Mean	6	1.884934	0.31415577	7.83	0.00
OneR AUC	6	0.408899	0.068149977	8.19	0.00
OneR G-Mean	6	2.664982	0.444163737	10.22	0.00

This implies that in all four test cases, at 5% confidence level (α=0.05) the choice of a sampling technique has significant impact on the performance (AUC and G-Mean measures) of the C4.5 decision tree and OneR classification algorithms. This means that the performance of at least one of the sampling techniques is significantly different from

the rest in each of the test cases. Since the ANOVA models suggests that there is significant difference in the performance of the sampling techniques, the Tukey's Honestly Significance Difference (HSD) test specified in [41] is used to carry out a post-hoc pairwise comparison to determine which of the technique(s) resulted in significantly different performance.

The HSD test results of the C4.5 decision tree learner using AUC and G-Mean are shown in Table IV (a) and (b) respectively, and the results of the OneR learner shown in Table V (a) and (b). The first column in each table contains the sampling techniques, the second and third columns show the mean performance and rank of the respective sampling techniques. In these tables, if two techniques have the same letter in an HSD column it implies that their performances are not statistically different.

TABLE IV: HSD TEST RESULTS FOR C4.5

Tech	AUC	
	Mean	HSD
CUST	0.805	A
CBU	0.714	BA
SMOTE	0.709	B
RUS	0.705	B
ROS	0.660	B
OSS	0.656	B
NONE	0.630	B

Tech	G-Mean	
	Mean	HSD
CUST	0.766	A
RUS	0.600	BA
CBU	0.593	BA
SMOTE	0.505	BC
OSS	0.447	BC
ROS	0.392	BC
NONE	0.366	C

TABLE V: HSD TEST RESULTS FOR ONER

Tech	AUC	
	Mean	HSD
CUST	0.7478	A
CBU	0.6854	BA
RUS	0.6683	BAC
SMOTE	0.6152	BDC
OSS	0.5968	BDC
ROS	0.5821	DC
NONE	0.5673	D

Tech	G-Mean	
	Mean	HSD
CUST	0.724	A
CBU	0.602	BA
RUS	0.563	BAC
SMOTE	0.405	BDC
ROS	0.357	DC
OSS	0.351	DC
NONE	0.259	D

In a summary, CUST statistically outperformed SMOTE, ROS, OSS, and NONE in all cases considered. CUST also outperformed RUS in the C4.5 classification algorithm when AUC measure is used as shown in Table IV (a). It performed statistically the same though with higher average performances in the remaining three test cases as RUS, which is reported in literature [4], [25], to be a very competitive sampling technique. RUS, though performed statistically the same as CUST, failed in all cases to outperform SMOTE, ROS and OSS, which CUST statistically outperformed and also, failed to outperform NONE as in Table IV (a). CBU, a lesser known undersampling technique which has never been studied using these datasets particularly the software defect datasets, also performed statistically the same as CUST and RUS. CBU, failed to outperform SMOTE in all cases, ROS and OSS in both cases of the C4.5 decision tree learner. Three sampling techniques, SMOTE, ROS, and OSS have in all cases performed statistically the same as NONE.

VI. CONCLUSION

Data classification or class prediction using machine learning algorithms is widely studied and the aim is often to build prediction models that can identify class of instances in arriving data streams as much as possible in order to allow optimum allocation of limited resources or to aid timely decision making. Class imbalance, which is inherent in most real world datasets militate against the classifiers ability to adequately learn to correctly identify positive class instances since they always constitute the minority in the datasets

Besides the class imbalance problem, there are other well-known data quality issues such as inconsistent/noisy instances, repeated/redundant instances, and outliers, which has the potential of negatively affecting the performance of classification algorithms when present in training data in substantial quantities. This study therefore designed, implemented and examined a Cluster Undersampling Technique that solves the class imbalance problem by eliminating majority class instances from the training data that fall into any of the above categories, thereby producing a final training set that is virtually void of irrelevant instances.

The proposed technique was empirically examined using two machine learning algorithms; C4.5 decision tree and OneR algorithm, ten real-world software defect datasets from the NASA MDP and six other datasets from the UCI repository. The performance of the classification algorithms when CUST is used to sample the training data prior to training was compared to the performance without sampling the training data. The results using AUC and G-Mean performance measures showed that using CUST improved the performance of the algorithms in all the datasets considered as compared to training the models without sampling.

The performance of CUST was compared to five existing sampling techniques which include RUS, CBU, SMOTE, OSS, and ROS. The AUC and G-Mean measures showed that CUST yielded better results in most of the datasets using both C4.5 and OneR learners. RUS and CBU produced competitive results particularly in datasets with fewer repeated instances and higher percentages of minority class instances. A Tukey's HSD test on the mean performance of the classification algorithms at α=0.05 showed that CUST statistically performed better that SMOTE, ROS, OSS and NONE. RUS and CBU yielded competitive results though they did not perform statistically different from SMOTE, ROS and OSS in most cases.

The performance of the proposed technique across the various datasets suggests that it increased the performance of the learners much better in datasets with lesser

percentages of minority class instances. As this study focused much on addressing the class imbalance problem in software defect datasets and datasets from other few areas, assessing the applicability and efficiency of CUST in other application areas where the class imbalance problem is equally predominant such as fraud detection, intrusion detection, and cancer detection is highly recommended.

It is however, worth noting that CUST like CBU becomes computational expensive when used on very large datasets with a larger number of clusters. This is as a result of the use of k-means algorithm whose computation complexity depends on the value of k, number of instances, the dimension of the dataset and the number of iterations performed during clustering. Considering the computational complexity of CUST, it is of interest to research into how the time requirements of CUST can be minimized.

REFERENCES

[1] X. Li, W. Ying, J. Tuo, B. Li, and W. Liu, "Applications of classification trees to consumer credit scoring methods in commercial banks," in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 2004, vol. 5, pp. 4112–4117.

[2] V. U. B. Challagulla, F. B. Bastani, I-L. Yen, and R. A. Paul, "Empirical assessment of machine learning based software defect prediction techniques," in *Proc. the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, 2005

[3] R. Malhotra and A. Jain, "Fault prediction using statistical and machine learning methods for improving software quality," *Journal of Information Processing Systems*, vol. 8, no. 2, June 2012.

[4] C. Seiffert, J. V. Hulse, T. M. Khoshgoftaar, and A. Felleco, "An empirical study of the classification performance of learners on imbalance and noisy software quality data," *Information Science*, 2011

[5] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Reflections on the NASA MDP data sets," *Special Issue on Evaluation and Assessment in Software Eng., IET*, 2011.

[6] E. Acuña and C. Rodríguez, "An empirical study of the effect of outliers on the misclassification error rate," *Transactions on Knowledge and Data Engineering*, 2005.

[7] J. Zhang, and I. Mani, "kNN approach to unbalanced data distributions: A case study involving information extraction," in *Proc. the ICML'2003 Workshop on Learning from Imbalanced Datasets*, 2003.

[8] M. Maloof, "Learning when data sets are imbalanced and when costs are unequal and unknown," in *Proc. the ICML'03 Workshop on Learning from Imbalanced Data Sets*, 2003.

[9] N. V. Chawla, "C4.5 and imbalanced datasets: Investigating the effect of sampling method, probabilistic estimate, and decision tree structure." in *Proc. the ICML'03 workshop on Class Imbalances*, August 2003.

[10] T. M Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Accuracy of software quality models over multiple releases," *Annals of Software Engineering*, vol. 9, no. 1–4, pp. 103–116.

[11] C. Seiffert, T. M. Khoshgoftaar, and J. V. Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 6, November 2009.

[12] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: an empirical case study," *Empirical Software Engineering*, vol. 9, pp. 229-257, 2004.

[13] H. Parvin, B. Minaei-Bidgoli, and H. Alizadeh, "Iranian cancer patient detection using a new method for learning at imbalanced datasets" in *Proc. Lecture Notes in Computer Science*, 2011, vol. 6936, pp. 299-306

[14] M. Di Martino, F. Decia *et al*., "Improving electric fraud detection using class imbalance strategies," *ICPRAM*, 2012

[15] N. Qazi and K. Raza, "Effect of feature selection, SMOTE and under sampling on class imbalance classification," in *Proc. 2012 UKSim 14th International Conference on Computer Modelling and Simulation*, 2012, pp. 145-150.

[16] V. García, A. I. Marqués, and J. S. Sánchez, "Improving risk predictions by pre-processing imbalanced credit data," in *Proc. 19th International Conference on Lecture Notes in Computer Science*, Doha, Qatar, November 12-15, 2012, vol. 7664, pp. 68-75.

[17] N. V. Chawla, L. O. Hall, K W. Bowyer, and W. P. Kegelmeyer, "SMOTE: Synthetic minority oversampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

[18] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One sided selection," in *Proc. 14th Int. Conf. Mach. Learn.*, 1997, pp. 179–186.

[19] B. Das, N. C. Krishnan, and D. J. Cook, "Handling imbalanced and overlapping classes in smart environments prompting dataset," *Springer Book on Data Mining for Services in Studies in Computational Intelligence*, 2012.

[20] NASA. [Online]. Available: http://mdp.ivv.nasa.gov/

[21] A. Asuncion and D. J. Newman, *UCI Machine Learning Repository*, Irvine, CA: University of California, Department of Information and Computer Science, 2007.

[22] H. He, "Introduction," *Imbalanced Learning: Foundations, Algorithms, and Applications*, 1st ed., New Jersey: John Wiley & Sons, Inc., 2013, pp. 1-8.

[23] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, September 2009.

[24] T. Maciejewski and J Stefanowski, "Local neighbourhood extension of SMOTE for mining imbalanced data," *IEEE*, 2011.

[25] D. J. Drown, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary sampling and software quality modeling of high-assurance systems," *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, vol. 39, no. 5, September 2009

[26] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: problems in software defect prediction," in *proc. 22nd International Conference on Tools with Artificial Intelligence*, 2010.

[27] I. Tomek, "Two modifications of CNN," *IEEE Transactions on System, Man, and Cybernetics*, vol. 6, pp. 769-772, 1976.

[28] D. Rocke and D. Woodruff, "Identification of outliers in multivariate data," *Journal of the American Statistical Association*, vol. 91, no. 435, pp. 1047-1061, 1996.

[29] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, New York, 1990.

[30] I. H. Witten, F. Eibe, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann Series in Data Management Systems, 3rd ed., Burlington, USA, Morgan Kaufmann Publishers Inc., 2011.

[31] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect data sets," November, 2011.

[32] R. Barandela, J. S. Sánchez, V. García, and E. Rangel, "Strategies for learning in class imbalance problems," *Pattern Recognition*, vol. 36, no. 3, pp. 849–851, 2003.

[33] V. García, J. S. Sánchez, and R. A. Mollineda, "On the effectiveness of preprocessing methods when dealing with different levels of class imbalance," *Knowledge-Based Systems*, vol. 25, pp. 13–21, 2012.

[34] J. R. Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[35] J. Aczel and J. Daroczy, *On Measures of Information and Their Characterizations*, New York: Academic, 1975.

[36] S. Marsland, "Learning with trees," *Machine Learning: An Algorithmic Perspective*, Machine Learning & Pattern Recognition Series, Chapman & Hall/CRC, Taylor & Francis group, 6000 Broken Sound Parkway NW, pp. 133-151, 2009.

[37] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning*, pp. 63–91, 1993.

[38] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 2-13, Jan. 2007.

[39] G. M. Weiss, "Mining with rarity: A unifying framework," *SIGKDD Explorer*, vol. 6, no. 1, pp. 7–19, 2004.

[40] T. R. Hoens and N. V. Chawla, "Imbalanced datasets: From sampling to classifiers," *Imbalanced Learning: Foundations, Algorithms, and Applications*, 1st ed., New Jersey: John Wiley & Sons, Inc., 2013, pp. 43-59.

[41] M. A. Agebure, "Improving software defect prediction using cluster undersampling," MPhil thesis, Dept. of Comp. Eng., Univ. of Ghana, Legon, Ghana, July 2014.

Robert A. Sowah graduated from the Kwame Nkrumah University of Science and Technology, Kumasi Ghana in 2000 with the bachelor of science degree in electrical & electronics engineering. Dr. Sowah received his master of engineering and PhD in electrical and computer engineering from Howard University, Washington, DC, USA in 2005 and 2009 respectively, and has over ten years of teaching and

research experience at the tertiary level. He has authored and co-authored papers published in refereed journals. His research interests include computational intelligence applications to health insurance claims, research and development of intelligent applications for countering energy thefts, and fault detection and remedial control using signal processing algorithms and wavelet neural networks, as well as modelling and simulation of processes.

Moses A. Agebure received his BSc degree in computer science from the University for Development Studies, Tamale, Ghana in 2008 and master of philosophy degree in computer engineering from the University of Ghana, Accra, Ghana in 2014. After his first degree he worked as a senior research assistant at the Department of Computer Science of the University for Development Studies and is currently an assistant lecturer in the same department of the university for development studies. He has co-authored papers published in journals. His research interests include, data mining, machine learning, software engineering and mobile computing systems.

Godfrey A. Mills received the bachelor degree in electrical & electronics engineering from the University of Science & Technology, Kumasi, Ghana in 1991. He obtained the MSc and PhD degrees in electronics and computing engineering from Gunma University, Japan, in 2002 and 2005 respectively. Prior to the graduate studies, he worked with the Electricity Company of Ghana (ECG) as power systems planning engineer for nearly a decade, where he carried out a number of networks designs and planning activities for the electric power distribution sector of Ghana. Godfrey has since March 2006, been working with the University of Ghana as a lecturer in the Department of Computer Engineering. His general research interests include signal processing, optical information processing and application, cryptographic techniques, embedded system design and computing. He is a corporate member of the Ghana Institution of Engineers (GHIE), Member of IEEE, and Optical Society of America.

Koudjo Koumadi is a 2009 winner of the British Computer Society's Wilkes Best Paper Award (Computer Journal). He holds a Ph.D and an MSc in information and communications engineering from the Advanced Institute of Science and Technology (KAIST), Daejeon, S. Korea, as well as a BSc in telecommunication engineering from Beijing University of Posts and Telecommunications (BUPT), Beijing, China. His research interests include radio resource management, machine-to-machine communications, and network infrastructure sharing. He is currently a senior researcher with Korea Electric Power Corporation, (KEPCO), S. Korea, where he focuses on advanced metering infrastructure (AMI) and machine-to-machine communications. Before joining KEPCO, Dr. Koumadi was with the University of Ghana, Department of Computer Engineering.

Seth Y. Fiawoo graduated with a BSc degree in computer engineering from the University of Ghana in 2012. Upon completing his bachelor's degree, Seth worked as a teaching and research assistant in the Computer Engineering Department at the University of Ghana from 2012 till 2014. He is currently pursuing an MSc in Computer Science from the University of Manitoba and is a member of the Autonomous Agents Lab. Seth's current research interests include multi robot systems, humanoid robotics, and machine learning.