Anomaly Detection in Smart Grids with Imbalanced Data Methods

Christian Promper
Information Technology & Systems Management
Salzburg University of Applied Sciences
Puch/Salzburg, Austria
cpromper.its-m2015@fh-salzburg.ac.at

Dominik Engel
Information Technology & Systems Management
Salzburg University of Applied Sciences
Puch/Salzburg, Austria
dominik.engel@fh-salzburg.ac.at

Robert C. Green II
Computer Science Department
Bowling Green State University
Bownling Green, OH, USA
greenr@bgsu.edu

***Abstract*—The research of anomaly-based intrusion detection within smart grids is a current topic and is investigated by many researchers. Thus, little experience is available on how to address the problem of detecting anomalies in smart grids. Another problem emerges when one tries to use common approaches of pattern recognition. As the data in such systems is typically highly imbalanced – there are many more normal instances than attack instances – there is often a high rate of misclassification when considering the attack, or minority class. In order to study this issue, this paper investigates the use of resampling techniques for intrusion detection inside of a hierarchical, three-layer smart grid communication system using a relatively new data set called ADFA-LD (this dataset includes contemporary attacks and is well-known for evaluating the performance of anomaly-based intrusion detection systems). Results compare the performance of typical and resampled techniques, demonstrating that the use of resampling leads to improved detection of attacks with a smart grid communication system.**

Keywords—smart grid, anomaly-based intrusion detection, imbalanced data, cost-sensitive learning, under- and over-sampling

I. Introduction

Over 100 years ago, the world's largest engineered system, the electric grid, was built. The electric grid consists of many different systems, components and owners but was not built for the requirements of the 21st century. Thus, the electric grid struggles with many weaknesses. For instance, as it is difficult to match the energy generation to the demand, energy utilities need to over-generate electricity to ensure a complete supply. Nonetheless, power outages can occur and these outages are usually recognized only after a customer complaint. Additionally, due to the unidirectional architecture of the grid, it is difficult to integrate renewable energy power plants (e.g., wind farms or photovoltaic systems) into the electric grid. To overcome these shortcomings, the so-called "smart grid" has emerged. Within a smart grid, intelligent communication and information systems are used, for instance, to flatten peak demands, to predict demand while balancing power generation or to transmit price information so that intelligent devices can be activated automatically. This intelligent system is comprised of a variety of sensors and communication devices, resulting in data flows between all grid components, utility providers and customers of the grid. This leads to various algorithms for estimation, control and pricing [1].

Unfortunately, through the integration of such systems many vulnerabilities arise [1]. Therefore, it is suggested in [2] to use methods from data analytics to monitor the communication in a smart grid to detect potential anomalies. But since most of the data is in a system such as the smart grid is typically associated with normal behavior and not disturbances or attacks, one must deal with an imbalanced data problem, where algorithms typically fail to classify the minority class with a high accuracy rate. However, methods exist to overcome these shortcomings.

Within this paper, these improved methods for the classification of imbalanced datasets will be investigated comprehensively using the Australian Defense Force Academy Linux Dataset (ADFA-LD). These methodologies are then transferred, using the same dataset, to a hierarchical communication model for the smart grid in order to develop an intrusion detection system that is capable of handling imbalanced data.

The remainder of the paper is organized as follows: In Section II, a thorough literature review will be presented; In Section III, the ADFA-LD will be discussed in detail; Section IV will describe the proposed methodology for the evaluation process of the ADFA-LD and the construction of a prototypical hierarchical smart grid intrusion detection system, in which a test for the ADFA-LD with the best common approaches and the best imbalanced data method will be executed; In Section V, the results for the evaluation process and the smart grid IDS will be presented and compared; Finally, the paper is concluded in section VI.

II. Literature Review

A. Intrusion Detection for Smart Grids

An Automated Metering Infrastructure (AMI) represents a communication network including smart meters, monitoring systems, computer hard- and software, data management systems and lots of sensors. An AMI is used for the bidirectional communication in a smart grid between utilities and the demand side. Since an AMI will consist mainly of wireless (mesh) networks with a lot of nodes, it is on the one hand more vulnerable for network-related attacks and unauthorized physical access and on the other hand it is more difficult to monitor such topologies [2].

978-1-5386-2726-6/17/$31.00 ©2017 IEEE

But it is learned from information technology security that a comprehensive security system needs to include monitoring systems. So, this study considers using an IDS within the smart grid network for monitoring purposes. To deploy an IDS within a smart grid, one must consider the requirements (e.g., encryption and real-time transmissions) and constraints (e.g., topology and bandwidth) of smart grid communication systems. These considerations can help to define impacts and limitations on functionalities and security for the communication architecture and the monitoring system. For example, a constraint for the implementation of an IDS in a smart grid is a high detection rate including zero-day attacks while causing only a low overhead [2].

However, it is a challenge to apply the knowledge of intrusion detection systems to smart grids to cover the vulnerabilities and yet to consider industry strengths. The main limitation of a traditional IDS architecture is to make it scalable for the size of a smart grid network since the processing of the data from millions of nodes on a central system would be too inefficient. To circumvent this problem, it would be for example possible to outsource some of the processing load directly to the sensors whereby the central management station is only responsible for coordinating sensors and collecting high-level alerts. Another requirement is the robustness against failures and attacks. So, the system is supposed to operate even when a subset of sensors or the management station are unavailable or compromised. While sensors can be protected through virtualization or by using a separate hardware, the approach for management stations is to use redundant systems. To detect compromised systems, various methods exist (e.g., a reputation system or a distributed proof system). Finally, it is suggested to use separated communication networks between sensors and management servers [2].

Existing approaches and concepts to implement an intrusion detection system within smart grids include a Model-Based IDS [3], a Behavior-Rule-Based IDS [4], an IDS with Domain Knowledge [5] and the Smart Grid Intrusion Detection System (SGDIDS) [6]. Since this paper considers the anomaly-based approach for the intrusion detection and the SGDIDS is based on an anomaly-based approach, this concept will be used as reference model later. Basically, the SGDIDS works with the three layers Home-Area Network (HAN), Neighborhood-Area Network (NAN) and Wide-Area Network (WAN) as detection architecture with a top-down and vice versa communication and information flow [6].

B. The Imbalanced Data Problem

An imbalanced data problem means that a dataset has an unequal distribution between the classes. The issue thereby is to achieve the same performance as for balanced datasets since common algorithms or classifiers are only optimized for balanced datasets or equal misclassification costs. In case of a two-class problem, an imbalanced data problem means that one class has significantly more instances than the other class. As example, consider the real-world medicine problem of detecting cancer with the two occurring classes healthy (negative) and cancerous (positive). This domain has imbalanced data in its nature since more healthy than cancerous patients exist. For example, the real-world "Mammography Data Set" contains 10,923 negative examples, denoted as the majority class, and only 260 positive examples, denoted as minority class. Usually, one wants to achieve a high classification performance for both classes. But with such a dataset the performance might be very high for the majority class (close to 100% correct classifications) and tends to be bad for the minority class (e.g., between 0% and 10% correct classifications). This implies, that from the 260 cancerous patients 90% to 100% would be classified as healthy. Since within the medical domain it is costlier to classify a cancerous patient as healthy than vice versa, it is important to improve the accuracy of the minority class. This problem can be assigned to other domains such as fraud detection or network intrusion, too [7].

A lot of different solutions to address the imbalanced data problem exist. They can be categorized by data-level solutions, algorithm-level solutions and ensemble solutions [8]. The data-level and algorithm-level solutions will be used within this paper and will be explained subsequently.

1) Data-level solutions

The approach for data-level solutions is to change the distribution of an imbalanced dataset to build a (more) balanced set. A sampled dataset is then used for the learning procedure and then the classifier might achieve better classification results. In a lot of studies, it was proved that some classifiers achieved a better overall performance with a sampled and (more) balanced dataset [7].

In general, one can distinguish between under-sampling and over-sampling. While under-sampling removes data from the majority class in the original imbalanced dataset, the over-sampling algorithm adds data to the minority class. Both sampling techniques have their own pros and cons. While under-sampling might lose some important concepts through removing instances from the majority class but might achieve a better performance in terms of the processing time, over-sampling might have a better detection performance but might lead to overfitting since some instances might be simply duplicated through the randomness. Therefore, some intelligent approaches exist to overcome these shortcomings [7].

In addition to various under- and over-sampling approaches, hybrid-sampling approaches exist, too. Basically, hybrid sampling combines under- and over-sampling in diverse ways to improve the performance [7]. The different approaches for under-sampling, over-sampling and hybrid-sampling will be explained following.

a) Under-Sampling

First, the RandomUnderSampler (RUS) simply chooses and removes majority samples randomly until the classes are balanced. To start with the more intelligent approaches, the CondensedNearestNeighbour (CNN) is based on the nearest neighbor rule. But a shortcoming of this method is that the classifier must store all training instances. So, CNN under-sampling is an improved method of the nearest neighbor rule which needs finally less space for storing. The EditedNearestNeighbours (ENN) under-sampling is based on the k-nearest neighbor rule. Basically, under-sampling performed by ENN creates a more balanced dataset distribution by accepting only instances which were correctly classified by

the k-nearest neighbor rule. RepeatedEditedNearestNeighbor (RENN) works identically as ENN. The only difference is that the process of removing wrongly classified instances is repeated infinite times resp. as long as no more eliminations are possible. However, this method has no proof of performance improvement in comparison to the ENN under-sampling [9]. The next under-sampling approach, All-KNN, iterates from k=1 to n over a given distribution of a dataset. For each round, the k-nearest neighbors are calculated and then each instance within the distribution is classified. If most of the n predictions for an instance are wrong, then this instance will be removed [10]. The InstanceHardnessThreshold (IHT) assumes a value denoted as hardness for each instance within a dataset. This value indicates the probability of misclassification. So, this method comprises an algorithm to measure the hardness to filter the instances based on a given threshold. NearMiss consists of three different versions, but all focus on the relation between minority and majority class. While version 1 selects instances with the lowest average distance between majority instances and three minority instances, version 2 calculates the distance to all minority instances and selects then the instances with the average distance to the three farthest minority examples. Finally, version 3 selects majority instances which are surrounded by minority instances. The under-sampling technique TomekLinks is based on CNN. Since CNN might have some shortcomings (e.g., random selection of instances at the beginning of the algorithm, which might lead to a disregarding of boundary instances), TomekLinks uses two modifications for an increased consideration of boundary instances. The OneSidedSelection (OSS) method creates subsets of all minority instances and only a single majority instance. Then, the original dataset is reclassified by the one-nearest neighbor rule and the misclassifications are added to the generated subset. Finally, TomekLinks under-sampling is used to remove noisy and borderline instances of the majority class. Finally, the NeighbourhoodCleaningRule (NCR) works like OSS but changes the one-nearest neighbor rule since the rule might be too sensitive to noise in the data. So, NCR under-sampling uses ENN under-sampling for the majority class to remove noisy instances. Then, misclassified instances are removed from both the minority and majority class with the 3-nearest neighbor rule [9].

b) Over-Sampling

First, the RandomOverSampler (ROS) is simply the reversed version of the RandomUnderSampler. ROS replicates minority instances randomly until the dataset is balanced [9].

The Synthetic Minority Over sampling TEchnique (SMOTE) uses synthetic instances to achieve a more balanced dataset. The regular version calculates the distance between an instance and the nearest neighbor and then multiplies this distance with a random number between 0 and 1. SMOTE borderline 1 and 2 assume that borderline instances are more likely to get misclassified. Thus, they are more important and so these over-sampling methods try to synthetize only borderline instances. Last, SMOTE SVM focuses on the borders of the minority and majority class. Finally, Adaptive Synthetic (ADASYN) over-sampling is based on SMOTE. The key difference is that ADASYN uses the k-nearest neighbors of an instance from the majority class and decides then, based on a weighting algorithm, how many minority instances the algorithm should synthetize. This is done with the intention to reduce bias through imbalance and to shift boundaries towards harder examples [9].

c) Hybrid-Sampling

The first approach, SMOTETomek, starts with over-sampling the dataset using SMOTE and then uses Tomek to under-sample the dataset. Since both under- and over-sampling have their shortcomings, the idea is to improve the results with a combination of both methods. The other hybrid-sampling approach, SMOTEENN, performs a similar procedure like SMOTETomek except using ENN to remove samples after the SMOTE over-sampling process. Since ENN is used instead of Tomek, this might lead to more removed instances which might further lead to a better performance [9].

2) Algorithm-level solutions

Beside cost-sensitive learning methods, the kernel-based learning framework, one-class learning and active learning approaches exist. However, this paper will focus only on cost-sensitive learning methods [7][8].

Regarding cost-sensitive learning, a misclassification of a data instance might be associated with different costs. These costs are represented as so-called cost-matrix. A cost-matrix contains numerical values with costs/penalties for misclassifying a pattern whereby there are usually no costs assigned for classifying a class correctly. However, if the actual cost values are unavailable, a common way to build a cost matrix for imbalanced data problems is to assign the imbalanced ratios inversely. Then, as soon as the cost matrix is built, the goal for cost-sensitive learning is to minimize the overall costs for the training set [7].

In general, methods for cost-sensitive learning can be distinguished by three distinct categories. These categories are data-space weighting, meta-techniques and classifiers with built-in cost-sensitive functions or features.

a) Data-space weighting

To apply cost-sensitive learning through data-space weighting, the misclassification costs are used to change the training data distribution. This approach is strongly based on the theoretical foundations of the Translation Theorem in [11]. So, the training distribution is changed to minimize the costs and to get the best possible distribution by multiplying each case by its relative cost. This can be performed either as transparent box or black box [12].

The transparent box passes the cost-matrix directly to the classifiers while the black box performs a re-sampling with the same cost-matrix before handing the data over to the classifier. However, this method might lead to overfitting [12].

b) Meta-techniques

The second category is built on theoretical foundations of the MetaCost Framework by [13]. In contrast to data-space weighting, a meta-technique does not sample the data distribution and is also called non-sampling cost-sensitive meta-learning. With cost-sensitive meta-learning it is possible to convert cost-insensitive classifiers into cost-sensitive classifiers

without modifying them. This is done either with pre-processing the training data or post-processing the output. However, this category can be further divided into the subcategories relabeling, weighting and threshold adjusting.

The first subcategory, relabeling, changes the classes of single instances by the minimum expected cost criterium. Relabeling can be either done for the training or test data. The next method, weighting, basically assigns a given weight (based on the cost-matrix resp. misclassification costs) to classes and so classes with higher weights get more consideration. The last method, threshold adjusting or also referred to as Thresholding, investigates the output probabilities and optimizes the threshold to minimize the total misclassification costs based on a given cost-matrix. In general, the output probabilities from the training instances are used to calculate a new optimal threshold. Then, the new calculated threshold is used as decision criterion to classify the output probabilities from the test instances. If the probability of a pattern is above the new threshold, the instance is predicted as positive and if the probability is lower than the new threshold then the instance is labelled as negative. This method avoids overfitting, too [14].

c) Built-in cost-sensitive functions

The last category integrates cost-sensitive learning methods directly into various classifiers. Since the way how functions are integrated or features are changed are very different, no unifying framework is available [7]. Classifiers used for such modifications are for example decision trees [7] [15], neural networks [7], random forests [15], bagging classifier [15], pasting classifier [15] and random patches classifier [15].

C. Performance Metrics

The performance of a two-class classification problem can be generally represented with a so-called confusion matrix including True Positives (TP), False Positives (FP), False Negatives (FN) and True Negatives (TN) and in general, all metrics are based on these values [7]. The most common metrics to evaluate the performance are the accuracy (ACC) and the error rate. While the error rate is just $1 - ACC$, the accuracy is expressed as seen in (1) where equation expresses the correct classification rate over all instances and is calculated by adding up all correct classifications and then dividing them by all instances.

$$ACC = \frac{TP+TN}{TP+FN+TN+FP} \tag{1}$$

Since in this paper, the performance for imbalanced data are evaluated, this metric is not very meaningful. Let us assume that an imbalanced ratio of 1:99 is present, which means that the minority class consists only of 1% of all data instances and the majority class consists of the remaining 99%. If an accuracy of 99% is achieved, that could mean that all majority instances were classified correctly but all minority instances wrongly [12]. To overcome this shortcoming, various other evaluation metrics exist which are more suited for the imbalanced domain [12] [16]. These metrics include Precision, True Positive Rate (TPR)/Recall/Sensitivity, True Negative Rate (TNR)/Specifity, False Positive Rate (FPR) and False Negative Rate (FNR) which are shown in (2)-(6).

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$TPR\ /\ Recall\ /\ Sensitivity = \frac{TP}{TP+FN} \tag{3}$$

$$TNR\ /\ Specifity = \frac{TN}{TN+FP} \tag{4}$$

$$FPR = \frac{FP}{FP+TN} \tag{5}$$

$$FNR = \frac{FN}{FN+TP} \tag{6}$$

Precision measures the exactness which means how many of all predicted positives are classified correctly. On the other hand, TPR or Recall/Sensitivity measures the completeness which means how many instances of all real positives are predicted correctly [16]. Intuitively, TNR computes how many instances of all real negatives are predicted correctly. FNR and FPR have an inverse relationship to TPR and TNR respectively. FNR states how many instances of all real positives are predicted wrongly and FPR calculates how many instances of all real negatives are predicted wrongly.

In addition to these metrics, other measures such as the F-Measure and G-Mean are commonly used and are shown in (7)-(8).

$$F - Measure = \frac{(1+ß^2)*Recall*Precision}{ß^2*Recall+Precision} \tag{7}$$

$$G - mean = \sqrt{sensitivity * specifity} \tag{8}$$

The F-Measure combines Recall and Precision as a weighted ratio to represent the effectiveness of the classifier. The weight is based on the ß parameter. Usually, this parameter is set to 1 and so a balanced weight of Precision and Recall is achieved (so-called "F1-Measure"). Even though this metric gives more insight than the accuracy metric, it is still sensitive to imbalanced data distributions. Finally, the G-mean calculates the ratio of positive accuracy and negative accuracy which represents the degree of inductive bias [16].

Additionally, the Receiver Operating Characteristics (ROC) curve plots the TPR against the FPR. Each point within this graph represents a single classifier at a specific data distribution. This means, that such a graph yields to a visual representation between benefits (TPR) and costs (FPR) for various data distributions. A single point within the plot originates from hard-type classifiers which are only able to produce a single $\{TPR, FPR\}$ pair. On the other hand, a series of ROC points produced by a threshold can generate full-featured ROC curves,

which are provided by soft-type classifiers. To compare the average performance of different classifiers, the area under curve (AUC) can be calculated [7].

III. The ADFA-LD

The ADFA-LD was made for anomaly based intrusion detection systems and created due to missing datasets containing contemporary attack protocols. One example of an outdated dataset is the Knowledge Discovery and Data Mining (KDD) dataset which was generated in 1998. The KDD dataset was historically the most used dataset for IDS research [17] [18].

To generate the ADFA-LD, an Ubuntu Linux Server Version 11.04 was used as operating system. To allow different attacks, Apache Version 2.2.17 with PHP Version 5.3.5 and MySQL in Version 14.14 were installed and started. The File Transfer Protocol (FTP) and the Secure Shell (SSH) services were enabled, too. To add additional vulnerability, TikiWiki Version 8.1 was installed and started. After the full installation of the software and the installation of all available patches, different payloads to attack the operating system were generated. These payloads include password brute forcing, adding new superusers, a Java Based Meterpreter, a Linux Meterpreter Payload and a C100 Webshell. The vectors used for the password brute force attack were FTP by Hydra and SSH by Hydra. A client-side poisoned executable vector was used for both adding new superusers and to transfer the Linux Meterpreter Payload. To get a Java Based Meterpreter session, a TikiWiki vulnerability exploit was sent to the server. Finally, for the C100 Webshell payload a PHP Remote File Inclusion vulnerability was exploited [17] [18].

Altogether, these payloads and attack vectors represent current practices to exploit a system. Considering the preparation of the server, a realistic defense environment was provided, too. Through several tests with different algorithms and the comparison with the KDD dataset, the ADFA-LD was validated as challenging and representative dataset for current cyber-attacks [17] [18].

A. Structure

During normal operations like web browsing or document operations, 833 traces of system calls for normal training data and 4373 traces of system calls for normal validation data were collected. The normal training data traces contain only traces with a file size between 300 Bytes and 6 Kilobytes while the normal validation data traces contain traces with a file size between 300 Bytes and 10 Kilobytes. The separation was done as trade-off between data fidelity and processing time. Since the goal of this paper is to gain the best detection rate, all normal data traces were combined. This results in a total of 5206 normal behavior traces. For the generation of the attack data, ten attacks were executed for each attack vector, which results in totally 746 attack data traces. Consequently, the imbalanced ratio is approximately 1:7 [17].

B. Processing

Each single data instance is an individual file and equals a system call trace whereby the term “trace” refers to a sequence of single system calls for a privileged process. Each different system call has a different unique system call identification (ID). In the dataset, a sequence of system call IDs is saved for each system call trace and is therefore a single data instance. Since system call traces have different lengths, it is not possible to process them directly with a machine learning algorithm. Different solutions to bypass this problem consider for example, trace lengths, the usage of common patterns or to count the frequencies. In [18] the author stated, that the trace length is not an effective way to find anomalies. Whereas common patterns like consecutive system call IDs are effective but highly time-consuming. Thus, a frequency based counting to gain a common sample length for system call traces was used.

C. Previous performance

Regarding [19], the highest percentage achieved for the ADFA-LD measured by the area under the curve for a ROC curve is 95.32%. This value was achieved by a classification with a semantic Extreme Learning Machine [20]. In comparison, 88.93%, 76.22% and 86.87% were achieved with a semantic SVM, a syntactic Hidden Markov Model [21] and the Sequence Time-Delay Embedding 10 method [19] respectively.

IV. Proposed Methodology

For this study, all evaluation tasks were executed with the Python programing language and all performances for the common approaches were achieved with the help of NumPy, SciPy, Matplotlib and especially Scikit-Learn [22]. To execute the cost-sensitive learning methods, the CostsensitiveClassification package [15] includes the stated cost-sensitive classifiers and was also used to perform Thresholding. To add weights to the classifiers, the classifiers within the Scikit-Learn package could be used. Finally, for all sampling methods, the Imbalanced-Learn package [23] was used.

The main goal of this study – implementing improved anomaly detection methods for imbalanced datasets in a smart grid communication hierarchy – was achieved through multiple steps.

First, common approaches and the mentioned imbalanced data methods were evaluated considering the ADFA-LD. The used classifiers included the k-nearest neighbor (k-NN) rule, a Multilayer Perceptron (MLP) classifier, the Quadratic Discriminant Analysis (QDA), Support Vector Machines (SVM), a boosting ensemble classifier with decision trees as base learner (“DTBoost”), a bagging ensemble classifier with decision trees as base learner (“DTBagg”) and a Random Forest classifier (“RForest”) To improve the performances for these classifiers, a grid search within their hyper-parameter space was executed. Additionally, a Plurality Voting classifier (“PlurVt”) and a Weighted Voting classifier (“WeighVt”) were used, too.

While the common methods and all sampling methods can use all stated classifiers, the cost-sensitive learning methods have restricted possibilities. The cost-sensitive weighting method is only executable for selected classifiers such as decision trees and SVMs. Therefore, the weighted classifier set consists of the classifiers SVM, DTBoost, DTBagg, RForest and the two voting classifiers PlurVt and WeighVt. Since thresholding is only possible for classifiers which can produce

probability outputs, the classifier set is restricted to the classifiers k-NN, MLP, QDA, SVM, DTBoost, DTBagg and RForest (voting classifiers cannot produce probability outputs). The cost-sensitive classifier set, with directly built-in cost-sensitivity, consists of a decision tree and Bagging, Pasting, Random Forests and Random Patches classifiers.

To execute the imbalanced methods, some individual changes in the common classification process are necessary. Let us start with the changes for cost-sensitive weighting and cost-sensitive classifiers. The only relevant change is that a cost-matrix is created which is either directly integrated into the classifier (weighting) or passed to train the cost-sensitive classifier. More changes are necessary for cost-sensitive thresholding. Once the original classifier is trained, the classifier is used to predict probabilities for test and training data. Then, a cost-matrix is created and the thresholding classifier is trained by the predicted training data probabilities and the cost-matrix. Finally, the targets are predicted with the thresholding classifier by the predicted test data probabilities. To execute the evaluation process with sampling methods, only a single change is necessary. The training data is sampled with the according sampling method and then the classifier is trained with the sampled training data. Then, each round is executed with a 5-fold cross validation and 20 repetitions to create robust classifiers and performances.

The next task is to build a hierarchical smart grid IDS to simulate a communication flow within a smart grid. To evaluate the performance of this communication system, the ADFA-LD is used to provide normal and attack data. One simulation round will use the unchanged dataset and another simulation round will use the previously chosen best imbalanced data method. Then, the two performances for the smart grid IDS will be compared.

So, a three-layer hierarchy smart grid architecture will be built similar to the hierarchical smart grid IDS system as described in [6]. Since the used architecture and communication flow for this hierarchical smart grid IDS are very complex, a prototypical implementation with a more simplified communication flow will be created. Therefore, the created prototype uses only a single IDS at each layer and is simulated only with if/else decisions.

For a better understanding of the created three-layer smart grid architecture, a single decision process for a single data instance will be described. This process illustrates the communication flow of a single data instance. First, a single data instance is passed to the HAN layer. Since the devices used in HANs (e.g., smart meters) have usually a low-performance, just the two fastest but still well performing classifiers were chosen for the HAN IDS. The used classifiers are the k-NN rule and the SVM classifier. The single data instance is then predicted with both classifiers. If both the k-NN rule and the SVM classifier predict the same class, then this prediction is a final decision. If they disagree in their decision, the data instance is passed to the next layer. Within the NAN layer, the data instance is now predicted from an IDS with four different classifiers, namely k-NN, MLP, SVM and Random Forests. If most of the classifiers decide for one class, then this class is the final decision. On the other hand, if two of the classifiers predict one class and the other two classifiers predict the other class, then the data instance is again forwarded to the next and last layer. In the WAN layer, the data instance is predicted by the plurality voting classifier. Since this is the last layer, no further decision is necessary. Consequently, this prediction is the final decision.

V. Results

For all results, the F1-Measure is denoted as “F1” and the G-mean is denoted as “G”.

A. Performance Evaluation

The results for the performance evaluation task can be found in Table I. Since the comparison of all imbalanced data methods would be beyond the scope of this study, only the best method for each sampling type and the cost-sensitive methods are stated.

The common approaches were compared to the imbalanced data methods. All sampling methods were able to improve the AUC score. But on the other hand, nearly all cost-sensitive learning methods decreased the AUC performance. The RENN under-sampling method can best improve the AUC performance of each single classifier and ensemble learner. An improvement of the AUC between 0.30% and 1.50% was achieved by a more balanced detection rate. This means, that there are less detected majority instances but therefore a lot more detected minority instances. This behavior is exactly as expected and improves the overall detection rate (based on the AUC metric) while a much better detection rate for the minority class was achieved. Although the RENN under-sampling achieved the best detection rates for the minority class, with ADASYN over-sampling the AUC score could be raised again. Comparing the best single classifiers, the common approach achieved an AUC of 93.58% with the plurality voting classifier, the RENN under-sampling achieved an AUC of 94.56% with the weighted voting classifier and through ADASYN over-sampling the plurality voting classifier achieved an AUC score of 95.10%. SMOTETomek hybrid-sampling achieved also for each classifier a better AUC score than the common approach. But, the detection rate for the minority class is worse than the detection rate achieved by under- or over-sampling. Anyway, a slightly better detection rate for the majority class was achieved. Nevertheless, the best classifier from SMOTETomek hybrid-sampling achieved in terms of the AUC metric a 0.10% better performance than the best classifier from RENN and a 0.50% worse performance than ADASYN. The cost-sensitive weighting had only insignificant impact on the performance. While individual classifiers are slightly better, other classifiers are slightly worse and the voting classifiers achieve also a worse performance since there are less single classifiers to consider for the voting. So, the overall performance of cost-sensitive weighting is worse than the normal scenario.

All in all, the cost-sensitive classifiers are very disappointing, since cost-sensitive weighting achieved only a similar performance to the normal scenario and thresholding and the cost-sensitive classifiers even decreased the performance. But the sampling methods compensate this deficient performance.

TABLE I. ADFA-LD PERFORMANCE RESULTS

ADFA-LD		FPR	FNR	AUC	ACC	F1	G
Common approaches	k-NN	0.0186	0.1251	0.9281	0.9680	0.8728	0.9266
	MLP	0.0165	0.1334	0.9250	0.9689	0.8746	0.9232
	QDA	0.1661	**0.0319**	0.9010	0.8507	0.6191	0.8985
	SVM	0.0133	0.1443	0.9212	0.9703	0.8783	0.9189
	DTBoost	0.0135	0.1487	0.9189	0.9695	0.8751	0.9164
	DTBagg	0.0111	0.1444	0.9223	0.9722	0.8852	0.9199
	RForest	**0.0097**	0.1436	0.9233	0.9735	0.8901	0.9209
	PlurVt	0.0107	0.1176	**0.9358**	0.9759	0.9017	**0.9343**
	WeighVt	0.0099	0.1190	0.9355	**0.9764**	**0.9034**	0.9339
RENN Under-Sampling	k-NN	0.0663	0.0603	0.9367	0.9345	0.7824	0.9367
	MLP	0.0583	0.0660	0.9379	0.9407	0.7980	0.9379
	QDA	0.1633	**0.0288**	0.9040	0.8536	0.6245	0.9015
	SVM	0.0542	0.0777	0.9341	0.9428	0.8018	0.9340
	DTBoost	0.0546	0.1018	0.9218	0.9394	0.7881	0.9215
	DTBagg	0.0435	0.0883	0.9341	0.9509	0.8230	0.9338
	RForest	**0.0386**	0.0839	0.9387	**0.9557**	**0.8384**	0.9385
	PlurVt	0.0514	0.0613	0.9437	0.9474	0.8173	0.9437
	WeighVt	0.0523	0.0564	**0.9456**	0.9472	0.8174	**0.9456**
ADASYN Over-Sampling	k-NN	0.0407	0.0735	0.9429	0.9552	0.8383	0.9428
	MLP	0.0347	0.0849	0.9402	0.9590	0.8484	0.9399
	QDA	**0.2209**	**0.0392**	0.8699	0.8019	0.5487	0.8652
	SVM	0.0339	0.0929	0.9366	0.9587	0.8462	0.9361
	DTBoost	0.0229	0.1176	0.9297	0.9652	0.8641	0.9285
	DTBagg	0.0274	0.0887	0.9420	0.9649	0.8670	0.9415
	RForest	0.0237	0.0928	0.9418	0.9677	0.8755	0.9411
	PlurVt	0.0283	0.0697	**0.9510**	0.9665	0.8746	**0.9508**
	WeighVt	0.0254	0.0745	0.9500	**0.9684**	**0.8802**	0.9497
SMOTETomek Hybrid-Sampling	k-NN	0.0385	0.0721	0.9447	0.9573	0.8448	0.9446
	MLP	0.0263	0.1092	0.9322	0.9633	0.8588	0.9313
	QDA	0.1458	**0.0464**	0.9039	0.8667	0.6420	0.9025
	SVM	0.0257	0.1200	0.9272	0.9625	0.8547	0.9260
	DTBoost	**0.0196**	0.1284	0.9260	0.9668	0.8680	0.9244
	DTBagg	0.0212	0.1098	0.9345	0.9677	0.8734	0.9334
	RForest	0.0175	0.1056	0.9385	0.9715	0.8872	0.9375
	PlurVt	0.0212	0.0867	0.9460	0.9706	0.8862	0.9455
	WeighVt	0.0199	0.0874	**0.9464**	**0.9717**	**0.8898**	**0.9458**
Cost-sensitive weighting	SVM	0.0219	0.2355	0.8713	0.9514	0.7976	0.8647
	DTBoost	0.0150	0.1543	0.9154	0.9676	0.8674	0.9127
	DTBagg	0.0118	0.1451	0.9215	0.9715	0.8826	0.9191
	RForest	0.0101	**0.1442**	**0.9228**	0.9731	**0.8885**	**0.9204**
	PlurVt	**0.0083**	0.1638	0.9140	0.9722	0.8830	0.9106
	WeighVt	0.0093	0.1451	**0.9228**	**0.9737**	0.8905	0.9203
Cost-sensitive Thresholding	k-NN	0.0415	**0.0791**	**0.9397**	0.9538	0.8332	**0.9395**
	MLP	**0.0183**	0.1830	0.8993	0.9610	0.8402	0.8955
	QDA	0.0195	0.1218	0.9293	**0.9677**	**0.8719**	0.9279
	SVM	**0.0183**	0.1831	0.8993	0.9610	0.8402	0.8955
	DTBoost	**0.0183**	0.1831	0.8993	0.9610	0.8402	0.8955
	DTBagg	0.0195	0.1221	0.9292	**0.9677**	**0.8719**	0.9278
	RForest	0.0195	0.1221	0.9292	**0.9677**	**0.8719**	0.9278
Cost-sensitive classifiers	DT	**0.0256**	0.4955	0.7395	**0.9155**	0.5996	0.7011
	Bagging	0.0864	0.1014	0.9061	0.9117	0.7185	0.9061
	Pasting	0.0965	0.0879	0.9078	0.9046	0.7056	0.9078
	RForest	0.0879	0.0839	**0.9141**	0.9126	**0.7244**	**0.9141**
	RPatches	0.1215	**0.0753**	0.9016	0.8843	0.6671	0.9013

TABLE II. SMART GRID HIERARCHY IDS PROCESSES INSTANCES

	HAN passed	HAN acc.	NAN passed	NAN acc.	WAN passed	WAN acc.
original	1161.54	97.92%	19.14	83.32%	10.32	74.22%
sampled	1143.56	97.03%	32.04	82.92%	15.40	69.57%

Regarding the best method among them, the decision is not easy to make. Compared only based on the AUC metric, ADASYN over-sampling has the best performance even when RENN under-sampling achieved just a 0.50% lower AUC score while having generally a higher detection rate for the minority class. Nevertheless, ADASYN over-sampling achieved with the plurality voting classifier the best AUC performance (95.10%).

B. Smart Grid Hierarchy IDS

The processed data instances and the achieved accuracy for each hierarchy layer are stated in Table II. In total 1,191 test data instances were processed. Since the smart grid IDS detection process was repeated 100 times, the amount of processed data for each hierarchy layer has decimals. In the simulation process with un-sampled training data, on average 1,161.54 data instances were processed in the HAN layer. This means, that less than 30 instances were passed to next layer. For the simulation round with over-sampled training data, the number of processed instances at the HAN layer is 1,143.56 averaged. Less than 48 data instances were passed to the next layer. But this fact is only remarkable since the total accuracy at this layer is very high. So, the simulation with the original training data achieved an accuracy of 97.92% and the simulation with the over-sampled training data achieved an accuracy of 97.03% within the HAN layer. The first comparison of this accuracy already implies, that the behavior for the over-sampled data might be similar as described in the previous section. However, for the scenario with the original data, the NAN layer processed over 19 data instances from the approximately remaining 30 (approximately 65% of the remaining instances). This was accomplished with a total accuracy of 83.32%. On the other hand, the scenario with the over-sampled training data processed over 32 instances from the averaged 47.44 remaining instances (approximately 67.50% of the remaining instances). The total accuracy at the NAN layer for these instances is 82.92%. The WAN layer processed only 10.32 data instances for the round with original data with an accuracy of 74.22% and only 15.40 data instances for the round with over-sampled data with an accuracy of 69.57%.

The performance metrics for the complete smart grid communication model are stated in Table III. As assumed, the behavior of the smart grid communication system is similar to a common prediction process from the previous section. For the simulation round with unchanged methods, the detection rate for the minority class is 88.04% and for the majority class 98.82%. This leads to an AUC score of 93.46%. Through ADASYN over-sampling, the AUC score was raised to 94.90%. This score was achieved with a minority class detection rate of 93.03% and a majority class detection rate of 96.77%. So, a 1.44% higher AUC score with 5% more detected minority instances was achieved. Finally, the authors compared this performance to the results from the previous section. The smart grid IDS could achieve the same performance as their respective best classifiers.

TABLE III. SMART GRID HIERARCHY IDS PERFORMANCE RESULTS

	FPR	FNR	AUC	ACC	F1	G
original	1.18%	11.96%	93.46%	97.48%	89.68%	93.27%
sampled	3.23%	6.97%	94.90%	96.30%	86.34%	94.88%

VI. CONCLUSION

The goal of this study was to investigate imbalanced data methods and to use these methods for anomaly detection in smart grids. For this purpose, various classifiers were tested for the ADFA-LD with common approaches and imbalanced data methods. While the performance for the cost-sensitive learning methods were disappointing, the sampling methods fulfilled

their expectations. Especially through over-sampling they could improve the detection rate of the minority class while the detection rate for majority class nearly remained. This behavior led to an overall improved AUC score.

After the exploration of all methods, the best method for the ADFA-LD was chosen to build a smart grid IDS. To build the smart grid IDS, a hierarchical three-layer communication system were constructed with if/else conditions. Then, both the best common method and the best imbalanced data method were evaluated by a simulation with the built smart grid IDS. The expectation was, that the imbalanced data method outperforms existing approaches. Considering the AUC score and the detection rate for the minority class, this goal was definitely achieved (at the expense of a little lower detection rate for the majority class). But the hierarchical smart grid IDS itself was also able to improve the overall performance. So, the performance for both methods match their respective best performing classifier, namely the plurality voting classifier. Consequently, a higher performance was achieved only through the use of the hierarchical three-layer smart grid IDS, too.

To extend this work, one might experiment with various combinations of classifiers and structures for the hierarchical smart grid IDS to improve the performance. Another possibility would be to add some ensemble solution classifiers to the classifier set or to add some classifiers from the algorithm-level solutions (e.g., kernel-based learning framework, one-class learning approach or active learning approach). Finally, one could change the prototypical implementation with if/else conditions to a real smart grid communication system as created in [6].

REFERENCES

[1] Y. Mo, T. H.-J. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig und B. Sinopoli, „Cyber–Physical Security of a Smart Grid Infrastructure," Proceedings of the IEEE, Bd. 100, Nr. 1, pp. 195 - 209, January 2012.

[2] R. Berthier, W. H. Sanders und H. Khurana, „Intrusion Detection for Advanced Metering Infrastructures: Requirements and Architectural Directions," in IEEE International Conference on Smart Grid Communications, Gaithersburg, MD, USA, 2010.

[3] F. M. Tabrizi und K. Pattabiraman, „A Model-Based Intrusion Detection System for Smart Meters," in International Symposium on High-Assurance Systems Engineering, Miami Beach, FL, USA, 2014.

[4] R. Mitchell und I.-R. Chen, „Behavior-Rule Based Intrusion Detection Systems for Safety Critical Smart Grid Applications," IEEE Transactions on Smart Grid, pp. 1254-1263, 29 April 2013.

[5] O. Linda, M. Manic und T. Vollmer, „Improving cyber-security of smart grid systems via anomaly detection and linguistic domain knowledge," in International Symposium on Resilient Control Systems, Salt Lake City, UT, USA, 2012.

[6] Y. Zhang, L. Wang, W. Sun, R. C. Green II und M. Alam, „Distributed Intrusion Detection System in a Multi-Layer Network Architecture of Smart Grids," IEEE Transactions on Smart Grid, pp. 796-808, 29 July 2011.

[7] H. He und E. A. Garcia, „Learning from Imbalanced Data," IEEE Transactions on Knowledge and Data Engineering, Bd. 21, Nr. 9, pp. 1263-1284, September 2009.

[8] B. Zhu, B. Baesens und S. K. vanden Broucke, „An empirical comparison of techniques for the class imbalance problem in churn prediction," Information Sciences, Bd. 408, pp. 84-99, October 2017.

[9] S. Shekarforoush, R. Green und R. Dyer, „Classifying Commit Messages: A Case Study in Resampling Techniques," in International Joint Conference on Neural Networks, Anchorage, Alaska, 2017.

[10] I. Tomek, „An Experiment with the Edited Nearest-Neighbor Rule," IEEE Transactions on Systems, Man, and Cybernetics, Bde. %1 von %2SMC-6, Nr. 6, pp. 448-452, June 1976.

[11] B. Zadrozny, J. Langford und N. Abe, „Cost-sensitive learning by cost-proportionate example weighting," in Third IEEE International Conference on Data Mining, Melbourne, FL, USA, 2003.

[12] P. Branco, L. Torgo und R. P. Ribeiro, „A Survey of Predictive Modeling on Imbalanced Domains," ACM Computing Surveys, Bd. 49, Nr. 2, p. Article No. 31, November 2016.

[13] P. Domingos, „MetaCost: a general method for making classifiers cost-sensitive," KDD '99 Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 155-164, August 1999.

[14] V. S. Sheng und C. X. Ling, „Thresholding for Making Classifiers Cost-sensitive," Proceedings of the 21st national conference on Artificial intelligence, Bd. 1, pp. 476-481, July 2006.

[15] A. Correa Bahnsen, „Example-Dependent Cost-Sensitive Classification with Applications in Financial Risk Modeling and Marketing Analytics," University of Luxembourg, 2015.

[16] A. K. Jain, R. P. Duin und J. Mao, „Statistical Pattern Recognition: A Review," IEEE Transactions on Pattern Analysis and Machine Intelligence, Bd. 22, Nr. 1, pp. 4-37, January 2000.

[17] G. Creech und J. Hu, „Generation of a new IDS test dataset: Time to retire the KDD collection," in Wireless Communications and Networking Conference (WCNC), Shangai, China, 2013.

[18] M. Xie und J. Hu, „Evaluating Host-Based Anomaly Detection Systems: A Preliminary Analysis of ADFA-LD," in International Congress on Image and Signal Processing (CISP), Hangzhou, China, 2013.

[19] G. Creech, „Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks," The University of New South Wales, 2013.

[20] G.-B. Huang, Q.-Y. Zhu und C.-K. Siew, „Extreme learning machine: a new learning scheme of feedforward neural networks," in IEEE International Joint Conference on Neural Networks, Budapest, Hungary, 2004.

[21] L. Rabiner und B. Juang, „An introduction to hidden Markov models," IEEE ASSP Magazine, Bd. 3, Nr. 1, pp. 4-16, 1 January 1986.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot und E. Duchesnay, „Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, pp. 2825-2830, October 2011.

[23] G. Lemaitre, F. Nogueira und C. K. Aridas, „Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," Journal of Machine Learning Research, Bd. 18, Nr. 17, pp. 1-5, 2017.