The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations

Stjepan Picek[1], Annelie Heuser[2], Alan Jovic[3], Shivam Bhasin[4], and Francesco Regazzoni[5]

[1] Delft University of Technology, Delft, The Netherlands
[2] CNRS/IRISA, Rennes, France
[3] University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
[4] Physical Analysis and Cryptographic Engineering, Temasek Laboratories at Nanyang Technological University, Singapore
[5] University of Lugano, Switzerland

Abstract. We concentrate on machine learning techniques used for profiled side-channel analysis when having imbalanced data. Such scenarios are realistic and often occurring, for instance in the Hamming weight or Hamming distance leakage models. In order to deal with the imbalanced data, we use various balancing techniques where we show that most of them help in mounting successful attack when the data is highly imbalanced. Especially the results with the SMOTE technique are encouraging since we observe scenarios where it reduces the number of necessary measurements for more than 8 times. Next, we provide extensive results on comparison of machine learning and side-channel metrics where we show that machine learning metrics (and especially accuracy as the most often used one) can be extremely deceptive. This opens a need to revisit the previous works and their findings in order to properly assess the performance of machine learning in side-channel analysis.

Keywords: Profiled side-channel attacks, Imbalanced datasets, Synthetic examples, SMOTE, Metrics

1 Introduction

SCA is a serious threat, which exploits weakness in physical implementation of cryptographic algorithms rather than the algorithms themselves [1]. The weakness stems from basic device physics of underlying computing elements i.e., CMOS cells, which makes it hard to eliminate such threats. It exploits any unintentional leakage observed in physical channels like timing, power dissipation, electromagnetic (EM) radiation, etc. For instance, a data transition from $0 \rightarrow 1$ or $1 \rightarrow 0$ in a CMOS cell causes current flow leading to power consumption. This can be easily distinguished from the case when no transition occurs

($0 \rightarrow 0$ or $1 \rightarrow 1$). When connected with sensitive data, these differences can be exploited by an adversary using statistical means. To date, the SCA which received the largest amount of attention is power analysis. As a side-channel, it uses the power consumed by the device. Despite being presented almost two decades ago [2], it is still a very relevant topic of research. Template attack is recognized as the most powerful side-channel attack, at least from an information theoretic point of view [3]. There, the attacker firstly profiles the behavior of a device similar to the targeted one and then uses this information to finalize the attack. In practice, there are many scenarios, for instance when the profiling set is small, where machine learning (ML) techniques are outperforming template attack. For this reason, researchers explored the use of machine learning (and more recently, deep learning) in the context of side-channel attacks [4–11].

In order to run side-channel analysis, one may select a leakage model where common examples are intermediate value, the Hamming weight, and the Hamming distance models. As an example, let us consider an 8-bit circuit returning random numbers between 0 and 255. If we take output values as class labels, we have uniformly distributed data. A simpler models would be the Hamming weight (HW) and the Hamming distance (HD), as commonly done in power analysis. Unfortunately, with such models, we obtain severely imbalanced data. There, some classes appear in 1/256 cases (when the HW/HD equals 0 and 8) while one class appears in 70/256 cases (when the HW equals 4). This problem, in reality, is much more complex due to the presence of noise. In this case, previous works demonstrate that often machine learning techniques classify all measurements as the majority class (Hamming weight 4), see e.g., [12]. Then, accuracy will reach around 27% on average but such classifier will not provide any relevant information in the context of SCA to recover the secret key. Such issues with imbalanced data are well-known in data science community and there exists no definitive general solution for this problem. The solutions that are available are purely empirical so it is not possible to give proper theoretical results on the best approaches to deal with imbalanced data.

To further elaborate this problem, we give a small experiment. In a simulated setting, we investigate the impact of imbalanced dataset on correlation (a commonly used distinguisher in SCA). We compute Pearson's correlation coefficient between a random generated dataset and its observation in form of HW. Two independent datasets are generated. While the first set is imbalanced as per binomial distributions of the HW model, the other set is balanced by oversampling. Oversampling is done by adding samples to underrepresented classes to make them at par with highly represented class (HW 4). When only one class is available, let us say HW 4, correlation remains zero as no distinguishing information can be recovered. As we add more classes, the correlation between the dataset and its observation increases. In the experiment, classes are gradually added from the most populated to the least populated. Figure 1 shows how the imbalanced dataset perform compared to the balanced one, each time as we increase the number of classes from 1 to 9. While adding noise (shown for standard deviation σ 0.5 and 1) to the dataset has limited impact, a balanced

dataset surely improves the correlation. Consequently, the advantage of balanced dataset is evident.

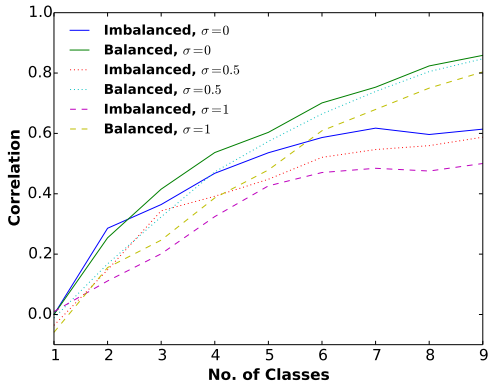


Fig. 1: Correlation vs the number of classes for imbalanced and balanced dataset

Since imbalanced data can introduce severe problems in the classification process, the question is how to assess the performance of a classifier or even how to compare the performance of several classifiers. While ML uses metrics like accuracy, precision, or recall as indicators of performance, SCA domain has specific metrics like guessing entropy and success rate that are applied over a set of experiments [13]. As we show in this paper, in some scenarios, the metrics from those two domains are sufficiently similar. Then, it is possible to estimate the success capabilities of an SCA already on the basis of ML metrics. In other scenarios, ML metrics do not provide relevant information to side-channel attackers.

In this paper, we concentrate on the problem of imbalanced datasets and how such data could be still used in a successful SCA. We examine the influence of the imbalanced data over several datasets and then we balance them by using either class sensitive learners or data sampling techniques. To the best of our knowledge, the performance of various oversampling techniques has not yet been studied in the SCA context. To assess the performance of such methods, we use both standard ML and SCA metrics. Our results show that data sampling techniques are a very powerful option to fight against imbalanced data and that such techniques, especially SMOTE, enables us to conduct successful SCAs and to significantly reduce the number of measurements needed. We emphasize that although we discuss machine learning, the same issues with imbalanced data and metrics remain for deep learning. For instance, Cagli et al. report problems coming from imbalanced data when using convolutional neural networks [11]. They use accuracy as the performance metric and recognize some limitations of

it but do not investigate it in more depth.

Our main contributions are:

1. We show the benefits of data sampling techniques to fight against imbalanced data.
2. We provide a detailed analysis of various machine learning metrics for assessing the performance of classifiers and we show that ML metrics should not be used to properly assess SCA performance.
3. The data balancing techniques we use, especially SMOTE, enables us to reach excellent results where we reduce the number of traces needed for a successful attack for up to 8 times.
4. We investigate the use of different machine learning metrics already in the training process in order to mitigate the effects of imbalanced data.
5. We present a detailed discussion on accuracy and SCA metrics to recognize the limitations of one metric for assessing the performance with another metric. As far as we are aware, such analysis has not been done.

The rest of the paper is organized as follows. Section 2 summarizes the background on the target algorithm and on side-channel analysis and introduces the particularities of our dataset. Section 3 discusses the phenomenon of imbalanced data and introduces techniques to reduce the influence of imbalanced data. Section 4 describe the experiments we carried out and report the achieved results. Finally, Section 5 discusses in details accuracy as a machine learning metric and guessing entropy/success rate as side-channel analysis metrics.

2 Background

In this section, we discuss profiling SCA and the Hamming weight and distance models. Next, we present the datasets we use, machine learning techniques, and performance metrics.

2.1 Profiling SCA

Profiling SCA performs the worst case security analysis since it assumes a strong adversary which has access to a clone device. The adversary obtains side-channel measurements from a clone device with known inputs, including the secret key. From this data set, also known as the profiling set, the adversary completely characterizes the relevant leakages. Characterized leakages are typically obtained for secret key dependent intermediate values, that are processed on the device and result in physical leakages. A leakage model or profile maps the target intermediate values to the leakage measurements. These models can then be used in the attacking phase on the target device to predict which intermediate values are processed and therefore have conclusions about the secret key.

Formally, a small part of secret key k^* is processed with t (i.e., a part of) input plaintext or output ciphertext of the cryptographic algorithm. In the case of AES, k^* and t are bytes to limit the attack complexity. The mapping y maps

the plaintext or the ciphertext $t \in \mathcal{T}$ and the key $k^* \in \mathcal{K}$ to a value that is assumed to relate to the deterministic part of the measured leakage x. For example,

$$y(t, k^*) = HW(\texttt{Sbox}[t \oplus k^*]), \tag{1}$$

where $\texttt{Sbox}[\cdot]$ is SubBytes and HW the Hamming weight. We denote $y(t, k^*)$ as the label which is coherent with the terminology used in the machine learning community.

In the rest of the paper, we are particularly interested in multivariate leakage $\boldsymbol{x} = x_1, \ldots, x_D$, where D is the number of time samples, i.e., features (or attributes). The adversary first profiles the clone device with known keys and uses obtained profiles for the attack. In particular, the attack functions in two phases:

- *profiling phase:* N traces $\boldsymbol{x}_{p_1}, \ldots, \boldsymbol{x}_{p_N}$, plaintext/ciphertext $t_{p_1}, \ldots, t_{p_N}$ and the secret key k_p^*, such that the attacker can calculate the labels $y(t_{p_1}, k_p^*), \ldots, y(t_{p_N}, k_p^*)$.
- *attacking phase:* Q traces $\boldsymbol{x}_{a_1}, \ldots, \boldsymbol{x}_{a_Q}$ (independent from the profiling traces), plaintext/ciphertext $t_{a_1}, \ldots, t_{a_Q}$.

In the attack phase, the goal is to make predictions about the occurring labels

$$y(t_{a_1}, k_a^*), \ldots, y(t_{a_N}, k_a^*),$$

where k_a^* is the secret unknown key on the attacking device.

One of the first and most commonly used profiling SCA method is template attack (TA) [3]. The attack uses Bayes theorem, dealing with multivariate probability distributions as the leakage over consecutive time samples is not independent.

2.2 The Hamming Weight and Distance Models

The preference for HW/HD model is related to the underlying device. As stated earlier, observing power consumption allows distinguishing a transition from no transition. Thus, when a new data is written into memory (or flip-flop), the total power consumption is directly proportional to the number of bit transitions. For example, this happens when a new data is written over old data (HD model) in flip-flops on embedded devices, or on a precharged data bus (HW model) in a microcontroller. Although the power consumption occurs both in logic and memory elements, the power consumption of memory is synchronized with the clock and is stronger than in logic. This makes exploitation easier due to high SNR. While weighted HW/HD model was shown to be better [14], it requires strict profiling, which varies from device to device. Contrary, HD/HW model works on a range of devices, providing a good starting point for evaluations.

In Eq. (1) $y(t, k^*)$ for i.i.d. values for t and k^*, follows a binomial distribution $B(n, p)$ with $p = 0.5$ and $n = 8$ in case of AES. Accordingly, the HW class value are imbalanced. Table 1 gives their occurrences.

Obviously, observing a HW value of 4 is more likely than any other value. This also has an influence on the amount of information each observed HW class

Table 1: Class taxonomy

HW value	0	1	2	3	4	5	6	7	8
Occurrences	1	8	28	56	70	56	28	8	1

value gives to an attacker to recover the secret key k_a^*. For example, knowing t and observing a HW of 4, it gives an attacker 70 possible secret keys, whereas observing a HW of 0 or 8 leads to only one possible secret key. Accordingly, the occurrence of HW classes close to 4 are more likely, but brings less information about the secret key.

To avoid such imbalance, working with intermediate values rather than its HW is an alternative. However, the computational complexity increases when dealing with huge number of intermediate classes (256 vs 9). With only 9 classes, HW is more resistant to noise as compared to 256 classes, which means lesser misclassification. The disadvantages of HW model, apart from imbalance, are less information on secret key as multiple intermediate value classes map to same HW class. HW model can sometimes be also misleading when dealing with countermeasures like dual-rail logic [15].

2.3 Attack Datasets

We use three different datasets for our experiments. The underlying cryptographic algorithm remains AES. As we are dealing with the classification problem with different machine learning algorithms, we are more interested in the first order leakage rather than higher order variants [16]. Consequently, countermeasures like masking remain out of scope. To test across various setting, we target 1) low-SNR unprotected implementation on FPGA, 2) high-SNR unprotected implementation on a smartcard, and 3) low-SNR implementation on a smartcard protected with the randomized delay countermeasure.

Unprotected AES-128 on FPGA (AES_HD) We first target an unprotected implementation of AES-128. AES-128 core was written in VHDL in a round based architecture, which takes 11 clock cycles for each encryption. The AES-128 core is wrapped around by a UART module to enable external communication. It is designed to allow accelerated measurements to avoid any DC shift due to environmental variation over prolonged measurements. The total area footprint of the design contains 1 850 LUT and 742 flip-flops.

The design was implemented on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board. Side-channel traces were measured using a high sensitivity near-field EM probe, placed over a decoupling capacitor on the power line. Measurements were sampled on the Teledyne LeCroy Waverunner 610zi oscilloscope. A suitable and commonly used (HD) leakage model when attacking the last round of an unprotected hardware implementation is the register writing in the

last round [17], i.e.,

$$Y(k^*) = HW(\underbrace{\texttt{Sbox}^{-1}[C_{b_1} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_{b_2}}_{\text{ciphertext byte}}), \tag{2}$$

where C_{b_1} and C_{b_2} are two ciphertext bytes, and the relation between b_1 and b_2 is given through the inverse ShiftRows operation of AES. We choose $b_1 = 12$ resulting in $b_2 = 8$ as it is one of the easiest bytes to attack. These measurements are relatively noisy and the resulting model-based SNR (signal-to-noise ratio), i.e., $\frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x-y(t,k^*))}$, with a maximum value of 0.0096. In total, 500 000 traces were captured corresponding to 500 000 randomly generated plaintexts, each trace with 1 250 features. As this implementation leaks in HD model, we denote this implementation as AES_HD.

DPAcontest v4 DPAcontest v4 provides measurements of a masked AES software implementation [17]. As we are interested in unmasked implementation, we consider the mask to be known and thus can easily turn it into an unprotected scenario. It is a software implementation with most leaking operation not being the register writing but the processing of the S-box operation and we attack the first round. Accordingly, the leakage model changes to

$$Y(k^*) = HW(\texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}}), \tag{3}$$

where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. Compared to the measurements from AES_HD, the SNR is much higher with a maximum value of 5.8577. The measurements consist of 4 000 features around the S-box part of the algorithm execution.

Random Delay Countermeasure Dataset As our last use case, we use a protected (i.e., with a countermeasure) software implementation of AES. The target smartcard is an 8-bit Atmel AVR microcontroller. The protection uses random delay countermeasure as described by Coron and Kizhvatov [18]. Adding random delays to the normal operation of a cryptographic algorithm has as an effect on the misalignment of important features, which in turns makes the attack more difficult to conduct. As a result, the overall SNR is reduced. We mounted our attacks in the Hamming weight power consumption model against the first AES key byte, targeting the first S-box operation. The dataset consists of 50 000 traces of 3 500 features each. For this dataset, the SNR has a maximum value of 0.0556. Recently, this countermeasure were shown to be prone to deep learning based side-channel [11]. However, since its quite often used countermeasure in commercial product, while not modifying the leakage order (like masking), we use it as a target case study. In the rest of the paper, we denote this dataset as the Random delay dataset.

2.4 Performance Metrics

As machine learning performance metrics, we consider total classification accuracy (ACC), Matthew's correlation coefficient (MCC), Cohen's kappa score (κ), precision, recall, F1 metric, and G-mean. To evaluate a side-channel attack, we use two common SCA metrics: success rate (SR) and guessing entropy (GE) [13].

Machine Learning Metrics MCC was first introduced in biochemistry to assess the performance of protein secondary structure prediction [19]. It can be seen as a discretization of the Pearson correlation for binary variables. Cohen's kappa is a coefficient developed to measure agreement among observers [20]. It shows the observed agreement normalized to the agreement by chance. Precision (also positive predictive value) is considered to be a measure of classifier's exactness, as it quantifies true positive instances among the all deemed positive instances. Recall (also sensitivity) is considered to be a measure of classifier's completeness, as it quantifies true positive instances that are found among positive instances. F1 is a harmonic mean value of precision and recall, while G-mean is geometric mean of recall (also called sensitivity) and negative accuracy (also called specificity). MCC, κ, precision, recall, F1, and G-mean are all well established in measuring classification performance on imbalanced datasets and are great improvements over accuracy on such datasets [21–23]. The equations used to obtain the evaluation metrics are given here:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4}$$

$$PRE = \frac{TP}{TP + FP}, \quad REC = \frac{TP}{TP + FN}. \tag{5}$$

$$F1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC} = \frac{2TP}{2TP + FP + FN}. \tag{6}$$

$$G_{mean} = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}}. \tag{7}$$

$$\kappa = \frac{P_{Obs} - P_{Chance}}{1 - P_{Chance}}. \tag{8}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}}. \tag{9}$$

TP refers to true positive, TN to true negative, FP to false positive, and FN to false negative classified instances. P_{Obs} is the percentage of observed agreement among observers, and P_{Chance} is the agreement expected by pure chance. Note

that due to distribution skewness in the analyzed datasets, ACC may not be the best choice for evaluation. We still include it for comparison purposes and for optimizing classifiers in the tuning phase. To efficiently visualize the performance of an algorithm, we can use the confusion matrix where in each row we represent the instances in an actual class, while each column represents the instances of a predicted class.

Success Rate and Guessing Entropy A side-channel adversary $A_{E_K,L}$ conducts experiment $\mathsf{Exp}_{A_{E_K,L}}$, with time-complexity τ, memory complexity m, and making Q queries to the target implementation of the cryptographic algorithm. The attack outputs a guessing vector g of length o, and is considered success if g contains correct key k^*. o is also known as the order of the success rate. The o^{th} order $\mathsf{success\ rate}$ of the side channel attack $\mathsf{A}_{\mathsf{E_K,L}}$ is defined as:

$$\mathrm{SR}^{o}_{A_{E_K,L}}(\tau, m, k^*) = \mathsf{Pr}[\mathrm{Exp}_{A_{E_K,L}} = 1]$$

The Guessing entropy measures the average number of key candidates to test after the attack. The Guessing entropy of the adversary $A_{E_k,L}$ against a key class variable S is defined as:

$$GE_{A_{E_K,L}}(\tau, m, k^*) = \mathsf{E}[\mathrm{Exp}_{A_{E_K,L}}]$$

As SCA metrics, we report the number of traces needed to reach a first-order success rate $\mathrm{SR}^{1}_{A_{E_K,L}}(\tau, m, k^*)$ (in short SR) of 90% as well as a guessing entropy $GE_{A_{E_K,L}}(\tau, m, k^*)$ (in short GE) of 10. We use '–' in case these thresholds are not reached within the test set.

2.5 Classifiers

In all our experiments, we use two classifiers: radial kernel support vector machines (SVM) and random forest (RF). These two well-known classifiers were used since they represent the usual classifiers of choice if highly accurate classification is sought. It is expected that they will perform among the best classifiers on the variety of datasets [24]. Although they may perform reasonably well even for moderately imbalanced data sets, it was already shown that performance of the classifiers on highly imbalanced data is expected to be reduced [25, 26].

Radial Kernel Support Vector Machines Radial Kernel Support Vector Machines (denoted SVM in the rest of this paper) is a kernel based machine learning family of methods that are used to accurately classify both linearly separable and linearly inseparable data. The idea for linearly inseparable data is to transform them to a higher dimensional space using a kernel function, wherein the data can usually be classified with higher accuracy. Radial kernel based SVM that is used here has two significant tuning parameters: cost of the margin C and the kernel parameter γ. The scikit-learn implementation we use

considers libsvm's C-SVC classifier that implements SMO-type algorithm based on [27]. The multiclass support is handled according to a one-vs-one scheme. The time complexity for SVM with radial kernel is $\mathrm{O}\big(D \cdot N^3\big)$, where D is the number of features and N is the number of instances. We experiment with $C = [0.001, 0.01, 0.1, 1]$ and $\gamma = [0.001, 0.01, 0.1, 1]$ in the tuning phase.

Random Forest Random Forest (RF) is a well-known ensemble decision tree learner [28]. Decision trees choose their splitting attributes from a random subset of k attributes at each internal node. The best split is taken among these randomly chosen attributes and the trees are built without pruning, RF is a parametric algorithm with respect to the number of trees in the forest. RF is a stochastic algorithm because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting. Learning time complexity for RF is approximately $\mathrm{O}\big(I \cdot k \cdot NlogN\big)$. We use $I = [10, 50, 100, 200, 500, 1000]$ trees in the tuning phase, with no limit to the tree size.

3 Imbalanced Data and How to Handle It

Imbalanced data are a phenomenon often occurring in real-world application where the distribution of classes is not balanced, i.e., some classes appear much more frequently than the other ones. In such situations, machine learning classification algorithms (e.g. decision trees and decision forests, neural networks, classification rules, support vector machines, etc.) have difficulties since they will be biased towards the majority class. The reason is that canonical machine learning algorithms assume the number of measurements for each class to be approximately the same. Usually, within imbalanced setting, we consider cases where the ratio between the majority and minority classes goes between $1:4$ and $1:100$. When the imbalancedness is even more pronounced, we talk about extreme imbalance data [29]. By referring to Table 1, we see that our HW scenario belongs to imbalanced scenarios, but approaching extreme imbalanced scenarios.

When the data is imbalanced, algorithms can decide to simply set all the measurements to belong to the majority class. As an example, let us consider the Hamming weight scenario with a significant amount of noise (so to make the classification process even more difficult) and two hypothetical classifiers. The first classifier can set all measurements to the class HW 4, which will result in accuracy of $70/256 \approx 27\%$. The second classifier will not classify everything as HW 4 but will have problems in correct classification (e.g., its accuracy will be somewhat better than a random guess of $1/9 \approx 11\%$, let us say 20%). The standard machine learning metrics will favor the first classifier since it reaches the better result, although such a result is not giving any useful information. Unfortunately, all machine learning classifiers (some more than others) are susceptible to such a behavior, but there are numerous techniques designed to alleviate the imbalanced data problem.

3.1 Handling Imbalanced Data

In order to improve the classification results in imbalanced data setting, there are essentially two main approaches:

1. Data-level methods that modify the measurements by balancing distributions.
2. Algorithm-level methods that modify classifiers to remove (or reduce) the bias towards majority classes.

Both of the approaches are performed in the data preprocessing phase, independently of the classifier that is used later for building the model. We consider typically used methods in machine learning community for both approaches. Aside from the methods that we consider here, there are also other approaches to help with imbalanced datasets, including those based on loss function maximization in cost-sensitive learning, classifiers adaptations (e.g., boosting SVMs) [30], or active learning [31]. For the purpose of introducing efficient imbalance solving methods in SCA, we focus on the well-known and successful methods for handling imbalanced data, which are described in the following paragraphs.

3.2 Cost-Sensitive Learning by Class Weight Balancing

The importance of a class is equal to its weight, which may be determined as the combined weight of all the instances belonging to that class. Balancing the classes prior to classification can be made by assigning different weights to instances of different classes (so called dataspace weighting) [23], so that the classes have the same total weight. The total sum of weights across all instances in the dataset is usually maintained, which means that the new instances are not introduced and that the weights of the existing instances are rebalanced so that it counteracts the effect of numbers of instances in each class in the original dataset. Thus, for example, a class A, having 2 times the number of instances as class B, would have all its instances' weights divided by 2, while class B would have all its instances multiplied by 2. To calculate the class weights, we use expression:

$$class\_weight_i = \frac{\#samples}{\#classes * \#samples_i}, \tag{10}$$

where $\#samples$ denotes the number of measurements in a dataset, $\#classes$ the number of classes, and $\#samples_i$ denotes the number of measurements belonging to the class i.

3.3 Data Resampling Techniques

Data resampling techniques usually belong in two major categories: undersampling and oversampling. In undersampling, the number of instances for a majority class is reduced, so that it becomes the same or similar to the minority class. In oversampling, the number of instances in the minority class is increased in order to become equal or similar to the majority class. In imbalanced multiclass setting, undersampling reduces the number of instances in all classes except

the one with the smallest number of instances, and oversampling increases the number of instances of all classes except the one with the highest number of instances. Oversampling may lead to overfitting when samples from the minority class are repeated and thus synthetic samples (synthetic oversampling) may be used to prevent it [32]. Here, overfitting means that the machine learning algorithm adapts to the training set too well and thus loses the ability to generalize to another datasets (e.g., test set). A simple way to identify overfitting is to compare the results on the training and testing sets: if the training set accuracy is much higher than the test set accuracy, then the algorithm overfitted.

As random undersampling may lead to loss of information in the majority class, informed undersampling needs to be used, which is based on classifier ensembles or k-NN approach [23]. Here, we consider two oversampling techniques (one random and one synthetic) and one combined synthetic oversampling with informed undersampling technique.

We do not use random undersampling techniques because of two reasons:

- Since we need to undersample all the classes except the least populated one (HW 0 or HW 8), we must significantly reduce the number of measurements in other classes. For instance, on average we need to reduce the measurements belonging to HW 4 for 70 times, or measurements belonging to classes HW 3 and HW 5 for 56 times. Although a common assumption is that the profiling phase is unbounded, the ratio of acquired measurements vs the number of actually used measurements is extremely unfavorable from the attacker's perspective.
- The second reason is even more difficult to mitigate. Since we need to remove measurements, we are in danger of removing extremely important information (measurements), which would make the loss of information even more significant than suggested by purely considering the number of removed measurements. Since we do not classify before undersampling (if we did, it would render undersampling not needed anymore), we cannot know whether we remove measurements that are the most informative.

Random Oversampling with Replacement Random oversampling with replacement oversamples the minority class by generating instances randomly selected from the initial set of minority class instances, with replacement. Hence, an instance from a minority class is usually selected multiple times in the final prepared dataset, although there is a possibility that some instances may not be selected at all. All minority classes are oversampled in order to reach the number of instances equal to the highest majority class. Interestingly, this simple technique has previously been found comparable to some more sophisticated resampling techniques [33].

Synthetic Minority Oversampling Technique The second method is SMOTE, a well-known resampling method that oversamples by generating synthetic minority class instances [32]. This is done by taking each minority class instance and introducing synthetic instances along the line segments joining any/all of the

k minority class' nearest neighbors (using Euclidean distance). It is reported that the k parameter works best for $k = 5$ [32]. The user may specify the amount of oversampling for each class, or else, the oversampling is performed in such a way that all minority classes reach the number of instances in the (highest) majority class.

Synthetic Minority Oversampling Technique with Edited Nearest Neighbor SMOTE + ENN [33] combines oversampling used by SMOTE and data cleaning by Edited Nearest Neighbor (ENN) method, originally proposed by Wilson [34]. ENN cleaning method works by removing from the dataset any instance whose class differs from the classes of at least two of its three nearest neighbors. In this way, many noisy instances are removed from both the majority and minority classes. By first applying SMOTE oversampling on all but the most numerous class, thus leveling the number of instances per class, and then applying ENN, noisy instances from all the classes are removed so that the dataset tends to have more defined class clusters of instances. Note that this type of cleaning may again lead to some class imbalance, depending on the data.

4 Experimental Validation and Discussion

As the first step, we randomly select a number of measurements from each dataset. From DPAv4 and AES_HD datasets, we select 75 000 measurements, while for the Random delay dataset, we take all 50 000 measurements that are available. Next, before running the classification process, we select the most important 50 features for each dataset. To do that, we use Pearson correlation coefficient. Pearson correlation coefficient measures linear dependence between two variables, x and y, in the range $[-1, 1]$, where 1 is total positive linear correlation, 0 is no linear correlation, and -1 is total negative linear correlation [35].

We divide the traces into training and testing sets, where each test set has 25 000 measurements. We experiment with three training set sizes where the measurements are selected randomly from the full training set: 1 000, 10 000, and 50 000 measurements (25 000 for Random delay). We use 3 datasets with significantly different sizes to demonstrate that imbalanced data problem persists over different problem sizes and that simply adding/removing measurements cannot help. On the training set, we conduct a 5-fold cross-validation for 10 000 and 50 000 (25 000 for Random delay) measurements. We run 3-fold cross-validation for 1 000 measurements due to the least represented class having only 3 measurements on average. We use the averaged results of individual folds to select the best classifier parameters. Before running the experiments, we normalize all the data into $[0, 1]$ range. Due to the lack of space, we report results from the testing phase only. All the experiments are done with the scikit-learn library [36] from Python.

4.1 Results

The classification results for the original (imbalanced), class weight balanced, random oversampling, SMOTE, and SMOTE+ENN datasets are available in Tables 2 to 6. Note that we do not give MCC, kappa, and G-mean results, since we found those metrics not providing relevant information except in the easiest cases (where also the presented metrics work). Additionally, we observe that even when SCA metrics show significant differences between scenarios, MCC, kappa, and G-mean often do not differ significantly (or at all).

Table 2 gives results for DPAcontest v4 scenario. Since this dataset has the highest SNR of all considered datasets (and is consequently the easiest), we see that machine learning algorithms do not have problems with dealing with imbalanced data. When the number of measurements is sufficiently high, we easily get accuracies of around 70%. At the same time, both SR and GE indicate it is possible to attack the target without issues. What is interesting, the difference in GE between SVM with 10 000 measurements and RF with 50 000 measurements is more than double, while the accuracies are within 1%. This is a clear indication that we cannot use accuracy as a good estimate of a susceptibility of an attack, even for a simple dataset. When applying class weight balancing, we observe small changes in both accuracies and GE/SR (no apparent correlation in change). For RF with 50 000 measurements, the accuracy even decreases when comparing to the imbalanced case, but both SR and GE reduce significantly. Random oversampling does not seem to be a good technique for handling imbalanced data in SCA, since, although accuracy does not decrease significantly, GE/SR for certain cases indicate much larger number of traces needed when compared to the imbalanced case. Finally, SMOTE and SMOTE+ENN techniques shows that, although accuracy can be even improved over imbalanced case, there seems to be no apparent advantage in using such techniques when considering SCA metrics. To conclude, in this low noise scenario, we see that using techniques to fight imbalanced data are not always bringing high improvements, especially when considering SCA metrics. As a natural question, one could ask how to decide do we need to use techniques to balance the data. One option would be to consider the confusion matrix. We give one example of it in Table 3. As it can be seen, machine learning classifier is able to correctly classify examples of all but one class, which is a good indication that we do not need to use additional techniques (although it could be beneficial).

When considering the dataset with the random delay countermeasure, we see the problem to be much more difficult. In fact for imbalanced dataset, only in few cases we are able to reach the threshold for SR/GE, but the number of traces needed is quite high. Interestingly, here we do not see almost any improvement when using class weight balancing (more precisely, we require around 500 traces less to reach the threshold for GE). Random oversampling is able to bring improvements, since now we are able to reach the thresholds on two more cases when considering GE and in 4 cases when considering SR. SMOTE, although strictly speaking successful in one less occasion, brings even more significant improvements since we now need much less traces to successfully reach

Table 2: DPAv4 dataset. Values are given as percentages (ML metrics) and number of traces (for GE/SR) on test set.

Method	Tr. size	Tuned	ACC	PRE	REC	F1	GE	SR
Imbalanced classification results								
SVM	1 000	C=1,γ=1	52.5	44	52	47	3	8
SVM	10 000	C=1,γ=1	72.4	71	72	71	3	7
SVM	50 000	C=1,γ=1	70.9	71	71	70	3	7
RF	1 000	I=1 000	69.1	69	69	68	5	11
RF	10 000	I=1 000	74.8	75	75	74	5	11
RF	50 000	I=500	73.2	73	73	72	7	16
Class weight balancing								
SVM	1 000	C=1,γ=1	42.4	37	42	36	4	10
SVM	10 000	C=1, γ=1	73.1	74	73	73	3	6
SVM	50 000	C=1, γ=1	71.5	72	72	71	3	7
RF	10 00	I=1 000	67.5	67	67	66	4	12
RF	10 000	I=1 000	74.1	74	74	73	4	10
RF	50 000	I=1 000	71.6	70	72	70	5	13
Random oversampling								
SVM	1 000	C=1, γ=1	49.4	43	49	44	4	9
SVM	10 000	C=1, γ=1	73.5	74	74	73	3	7
SVM	50 000	C=1, γ=1	70.7	71	71	70	3	8
RF	1 000	I=50	63.4	63	63	61	17	48
RF	10 000	I=1 000	72.6	72	73	71	5	11
RF	50 000	I=1 000	72.4	71	72	71	6	17
SMOTE								
SVM	1 000	C=1, γ=1	46.4	39	46	41	4	10
SVM	10 000	C=1, γ=1	72.6	73	73	72	3	7
SVM	50 000	C=1, γ=1	70.5	71	71	70	3	8
RF	1 000	I=200	67.0	66	67	65	9	13
RF	10 000	I=500	72.2	72	72	72	6	23
RF	50 000	I=500	72.2	72	72	72	9	23
SMOTE+ENN								
SVM	1 000	C=1, γ=1	36.3	28	36	27	20	62
SVM	10 000	C=1, γ=1	71.7	72	72	71	3	9
SVM	50 000	C=1, γ=1	68.4	69	68	68	3	9
RF	1 000	I=50	59.6	64	60	56	18	54
RF	10 000	I=500	73.2	74	73	73	6	17
RF	50 000	I=500	72.6	72	73	72	7	21

Table 3: Confusion matrix for DPAcontest v4 imbalanced dataset, SVM with $C = 1, \gamma = 1$, 10 000 measurements in the training phase.

Predicted									Actual
0	1	2	3	4	5	6	7	8	
0	65	43	0	0	0	0	0	0	0
0	39	710	6	0	0	0	0	0	1
0	0	2 118	670	4	0	0	0	0	2
0	0	326	4 148	894	3	0	0	0	3
0	0	6	744	5 410	719	2	0	0	4
0	0	0	5	951	4 121	421	0	0	5
0	0	0	0	9	628	2 069	37	0	6
0	0	0	0	0	2	584	176	0	7
0	0	0	0	0	0	9	73	8	8

the thresholds. Consider imbalanced case, RF with 50 000 measurements, where we need 13 500 measurements and the same classifier with SMOTE where we need only 1 600 measurements, which represents an improvement of more than 8 times. Moreover, with SMOTE we are able to reach a SR of 90% with only approx 5 500 measurements, where for all imbalanced data sets this threshold cannot be reached. SMOTE+ENN is again less successful than SMOTE and somewhere similar as the class weight balancing technique. Generally speaking, we observe that RF is more successful than SVM, which we attribute to the RF capability to deal with noisy measurements. Finally, this dataset is a good example to depict the problem of assigning all measurements to the majority class as it can be seen in Table 5. Regardless of the number of measurements, with such imbalancedness, we would never be able to break this target despite relatively good accuracy of 27.3%.

Finally, in Table 6, we give results for the AES_HD dataset. The results could be considered somewhere between the previous two cases: the dataset characteristics and imbalancedness represents bigger problem than for DPAcontest v4, but not as significant as in the Random delay dataset. We observe that, for this scenario, class weight balancing is actually deteriorating the behavior of classifiers as in less cases we are able to actually reach the threshold. Contrary, random oversampling helps and we have only three instances where GE or SR do not reach the threshold. Additionally, we see that, due to oversampling, several scenarios require less measurements to reach the threshold values. SMOTE, as in the previous scenarios, proves to be the most powerful method. There is only one instance where we are not able to reach the threshold and we observe significant reduction in the number of traces needed. SMOTE+ENN reaches all thresholds for the SVM algorithm but none for the RF algorithm. This further demonstrates how accuracy is not suitable measure since RF algorithm reaches higher accuracy values. Finally, other considered ML metrics and confusion matrices also do not reveal further insights, which shows how misleading ML metrics can be. We compare two confusion matrices for imbalanced scenario, RF with 10 000

Table 4: Random delay dataset. Values are given as percentages (ML metrics) and number of traces (for GE/SR) on test set.

Method	Tr. size	Tuned	ACC	PRE	REC	F1	GE	SR
Imbalanced classification results								
SVM	1 000	C=.001,γ=.001	27.3	7	27	12	–	–
SVM	10 000	C=.001,γ=.001	27.3	7	27	12	–	–
SVM	25 000	C=.001,γ=.001	27.3	7	27	12	–	–
RF	1 000	I=1 000	25.1	21	25	19	–	–
RF	10 000	I=1 000	26.5	23	26	18	18 800	–
RF	25 000	I=1 000	26.6	29	27	17	13 490	–
Class weight balancing								
SVM	1 000	C=1, γ=1	18.9	19	19	19	–	–
SVM	10 000	C=.01, γ=.01	11.1	1	11	2	–	–
SVM	25 000	C=.01, γ=.001	21.7	5	22	8	–	–
RF	1 000	I=100	24.9	19	25	19	–	–
RF	10 000	I=1 000	27.1	24	27	16	18 660	–
RF	25 000	I=1 000	27.0	24	27	15	12 980	–
Random oversampling								
SVM	1 000	C=1, γ=1	21.1	19	21	20	–	–
SVM	10 000	C=1, γ=1	20.7	20	21	20	19 290	–
SVM	50 000	C=1, γ=1	19.8	21	20	20	7 177	19 210
RF	1 000	I=200	24.4	20	24	20	–	–
RF	10 000	I=1 000	26.2	21	26	19	17 360	–
RF	50 000	I=1 000	26.3	25	26	19	7 173	20 650
SMOTE								
SVM	1 000	C=1, γ=1	22.0	20	22	21	–	–
SVM	10 000	C=1, γ=1	21.5	20	21	21	–	–
SVM	50 000	C=1, γ=1	21.3	21	21	21	10 320	–
RF	1 000	I=1 000	20.2	20	20	20	–	–
RF	10 000	I=1 000	23.2	21	23	21	4 305	13 710
RF	50 000	I=1 000	24.1	22	24	22	1 619	5 593
SMOTE+ENN								
SVM	1 000	C=1, γ=1	7.7	7	8	4	–	–
SVM	10 000	C=1, γ=1	9.3	14	9	5	–	–
SVM	50 000	C=1, γ=1	8.9	12	9	5	11 780	–
RF	1 000	I=500	7.3	5	7	4	–	–
RF	10 000	I=1 000	8.2	7	8	4	15 770	–
RF	50 000	I=1 000	8.9	19	9	5	20 400	–

Table 5: Confusion matrix for random delay imbalanced dataset, SVM with $C = 1, \gamma = 1$, 10 000 measurements in the training phase.

Predicted									Actual
0	1	2	3	4	5	6	7	8	
0	0	0	0	99	0	0	0	0	0
0	0	0	0	727	0	0	0	0	1
0	0	0	0	2 767	0	0	0	0	2
0	0	0	0	5 481	0	0	0	0	3
0	0	0	0	6 815	0	0	0	0	4
0	0	0	0	5 422	0	0	0	0	5
0	0	0	0	2 777	0	0	0	0	6
0	0	0	0	809	0	0	0	0	7
0	0	0	0	103	0	0	0	0	8

measurements and for SMOTE, RF with 10 000 measurements in Tables 7 and 8. Differing from Table 5, we observe that here, even for the imbalanced scenario, our classifier is able to correctly classify measurements into several classes (more precisely, 5 classes but where for one of them, we have only a single successful measurement). After applying SMOTE we observe that for 7 classes we have correct predictions.

In Figures 2a until 2d, we depict guessing entropy and success rate results for all 3 datasets when using either imbalanced datasets (straight lines) or those after applying SMOTE (dashed lines). We depict the results for both SVM and RF classifiers illustrating the significant improvements for the Random delay and AES_HD datasets.

4.2 Discussion

Our results clearly demonstrate that if the classification problem is sufficiently hard (e.g., for a dataset with a high level of noise) and there is imbalance, data sampling techniques may increase SR and GE significantly. As it can be seen from Table 2, the DPAcontest v4 dataset constitutes an easy classification problem. In this case, adding more samples to try to balance the dataset either by adjusting class weights or by oversampling is not needed or even beneficial. For the two more difficult datasets (Random delay and AES_HD), we see that balancing the classes may bring significant improvements. Comparing the techniques we investigated, the SMOTE technique performs the best, followed by Random Oversampling, class weight balancing, and finally, SMOTE+ENN.

On a more general level, our experiments indicate that none of the ML metrics we tested can be used as a reliable indicator of SCA performance when dealing with imbalanced data. In the best case, machine learning metrics can serve as indicator of performance where high value means the attack should be possible, while low value could indicate attack would be difficult or even impossible. But as it can be seen from our results, those metrics are not reliable.

Table 6: AES_HD dataset. Values are given as percentages (ML metrics) and number of traces (for GE/SR) on test set.

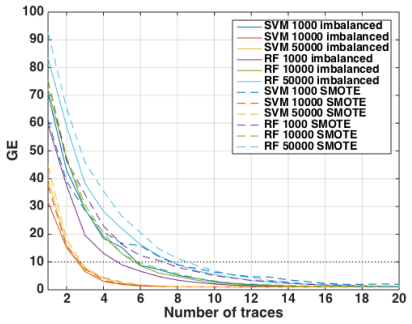
Method	Tr. size	Tuned	ACC	PRE	REC	F1	GE	SR
Imbalanced classification results								
SVM	1 000	C=.001,γ=.001	27.0	7	27	11	–	–
SVM	10 000	C=.001,γ=.001	27.0	7	27	11	13 330	24 700
SVM	50 000	C=.001,γ=.001	27.0	7	27	11	17 680	–
RF	1 000	I=500	24.7	21	25	20	–	–
RF	10 000	I=1 000	26.0	19	26	18	16 620	–
RF	50 000	I=1 000	26.0	23	26	18	13 560	24 380
Class weight balancing								
SVM	1 000	C=.001, γ=1	0.4	0	0	0	–	–
SVM	10 000	C=.01, γ=.001	11.0	1	11	2	–	–
SVM	50 000	C=.01, γ=.001	0.3	0	0	0	–	–
RF	1 000	I=200	25.1	21	25	19	–	–
RF	10 000	I=1 000	26.4	26	26	17	16 120	24 990
RF	50 000	I=1 000	26.7	17	27	15	16 650	–
Random oversampling								
SVM	1 000	C=1, γ=1	11.6	20	12	12	6 653	20 160
SVM	10 000	C=1, γ=1	17.5	21	18	18	10 320	14 520
SVM	50 000	C=1, γ=1	9.9	19	10	12	9 986	21 820
RF	1 000	I=500	24.3	20	24	20	–	–
RF	10 000	I=1 000	25.4	21	25	20	12 530	24 960
RF	50 000	I=1 000	25.9	21	26	19	16 190	–
SMOTE								
SVM	1 000	C=1, γ=1	18.0	20	18	17	11 700	21 850
SVM	10 000	C=1, γ=1	23.0	21	23	19	9 170	20 450
SVM	50 000	C=1, γ=1	23.7	20	24	19	17 320	–
RF	1 000	I=500	17.6	20	18	18	8 328	19 700
RF	10 000	I=1 000	16.7	20	17	17	2 877	7 943
RF	50 000	I=1 000	14.0	20	14	14	4 771	12 030
SMOTE+ENN								
SVM	1 000	C=1, γ=1	7.4	8	7	4	10 700	21 800
SVM	10 000	C=1, γ=1	6.2	3	6	4	10 390	22 410
SVM	50 000	C=1, γ=1	3.9	3	4	2	11 770	23 270
RF	1 000	I=500	8.9	3	9	4	–	–
RF	10 00	I=1 000	7.8	14	8	4	–	–
RF	50 000	I=1 000	7.8	3	8	4	–	–

Table 7: Confusion matrix for the AES_HD imbalanced dataset, RF with 1 000 iterations, 10 000 measurements in the training phase.

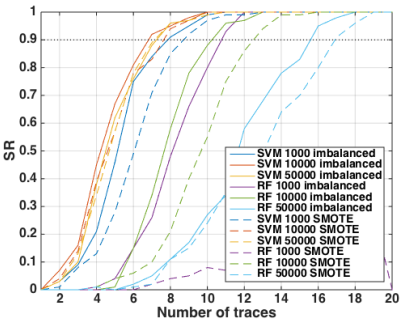
Predicted									Actual
0	1	2	3	4	5	6	7	8	
0	0	1	14	72	15	0	0	0	0
0	0	5	80	582	92	0	0	0	1
0	0	14	274	2 047	386	0	0	0	2
0	0	29	565	4 174	744	2	0	0	3
0	0	16	596	5 114	1 028	0	0	0	4
0	0	27	514	4 175	807	0	0	0	5
0	0	8	229	2 081	436	1	0	0	6
0	0	3	68	583	125	0	0	0	7
0	0	1	6	70	16	0	0	0	8

Table 8: Confusion matrix for the AES_HD imbalanced dataset after SMOTE, RF with 1 000 iterations, 10 000 measurements in the training phase.

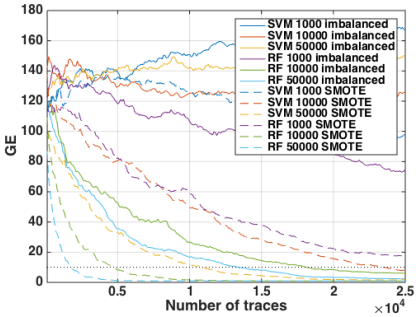
Predicted									Actual
0	1	2	3	4	5	6	7	8	
0	4	23	20	21	21	11	1	1	0
0	19	158	124	197	146	76	34	5	1
2	43	533	446	732	523	308	113	21	2
7	87	1 091	864	1 440	1 111	617	276	21	3
4	104	1 192	995	1 767	1 478	813	373	28	4
8	76	948	842	1 458	1 159	651	351	30	5
2	43	433	414	723	625	330	174	11	6
1	7	123	115	215	159	106	49	4	7
0	2	11	11	32	27	8	2	0	8



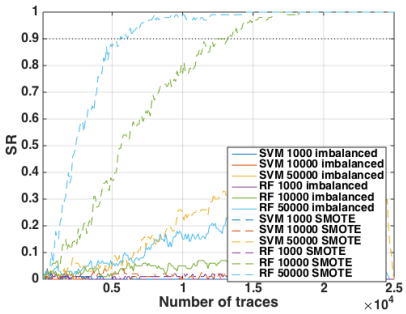
(a) Guessing entropy (GE) for the imbalanced and SMOTE dataset on DPAcontest v4.



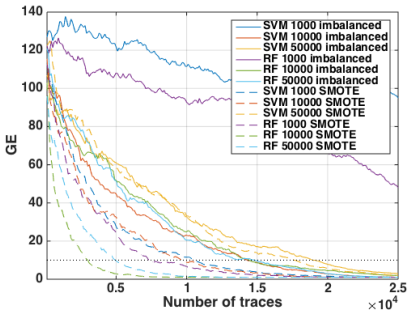
(b) Success rate (SR) for the imbalanced and SMOTE dataset on DPAcontest v4.



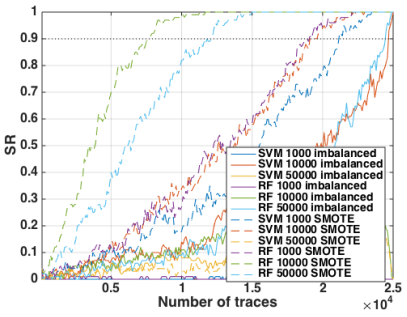
(c) Guessing entropy (GE) for the imbalanced and SMOTE dataset on Random delay dataset.



(d) Success rate (SR) for the imbalanced and SMOTE dataset on Random delay dataset.



(e) Guessing entropy (GE) for the imbalanced and SMOTE dataset on AES_HD.



(f) Success rate (SR) for the imbalanced and SMOTE dataset on AES_HD.

Fig. 2: Guessing entropy and success rate for imbalanced and SMOTE on all three datasets

Sometimes a small difference in machine learning metric means large difference in SCA metrics, but that is not a rule. We also see situations where machine learning metrics indicate significant difference in performance and yet, SCA metrics show absolutely no difference. Finally, as the most intriguing case, we also see that even lower values of machine learning metrics can actually be better when considering SCA metrics. To conclude, we experimentally prove that there is no clear connection between machine learning and side-channel metrics. Still, there are general answers (or intuitions) we can give.

Q Can we use accuracy as a good estimate of behavior of SCA metrics?

A The answer is no, since our experiments clearly show that sometimes accuracy can be used to infer about SCA success but is often misleading. This is also very important from the perspective where SCA community questions whether a small difference in accuracy (or other machine learning metrics) means anything for SCA. Unfortunately, our experiments show there is no definitive answer to that question. What is more, we see that we also cannot use accuracy to compare the performance of two or more algorithms. We give a detailed discussion about the differences between accuracy and SR/GE in the following section.

Q If accuracy is not appropriate machine learning metric for SCA, can we use some other ML metric?

A The answer seems to be again no. We experimented with 7 different machine learning metrics and none gave good indication of SCA behavior over different scenarios.

Q If we concluded that accuracy is not appropriate measure, what sense does it make to evaluate other ML metrics on test set, since still accuracy is used in training/tuning phase?

A We modified our classifiers to use different machine learning metrics (as given in Section 2.4) already in the training phase. The results are either comparable or even worse than for accuracy. Naturally, we did not test exhaustively all possible combinations, but the current answer seems to be that other ML metrics in the training phase do not solve the problem.

Q Can we design a new ML metric that would better fit SCA needs?

A Currently, the answer seems to be no. Simply put, using all the information relevant for SCA would mean that we need to use SCA metrics in classifiers. Anything else would mean that we need to extrapolate the behavior on the basis of only partial information.

Q Since we said that using all relevant information for SCA means using SCA metrics in ML classifiers, what are the obstacles there?

A Although there does not seem to be any design obstacles for this scenario, there are many from the implementation perspective. SCA metrics are computationally expensive on their own. Using them within machine learning classifiers means that we need to do tuning and training with metrics that are complex and slow to evaluate. Next, many machine learning algorithms

are actually much slower when required to output probabilities (e.g., SVM). Consequently, this would mean that the computational complexity would additionally increase. Finally, not all machine learning algorithms are even capable of outputting probabilities. This can be circumvented by simply not using such algorithms, but then we already impose some constraints on our framework.

Finally, our conclusions are not connected with any specific dataset or machine learning algorithm. The same problems should be expected in different settings, which implies that the same solutions could be used. For instance, Cagli et al. use deep learning (convolutional neural networks) and encounter problems with imbalanced data. They propose data augmentation techniques to remedy it but their techniques use random changes in the measurements [11]. We believe our setting is more powerful since we do not rely solely on random changes. More precisely, although such changes can improve generalization, they also introduce more erroneous measurements that limit the performance of classifiers.

5 Accuracy vs SR/GE

In this section, we discuss two differences between accuracy and SR/GE, where the first difference is present regardless of the imbalanced data problem and applies in general. We start by detailing the empirical computations of accuracy, SR, and GE in practice.

5.1 Empirical Computation of Accuracy and SR/GE

Let us denote the class labels in the attacking phase as

$$y_{a_1}, \ldots, y_{a_Q} = y(t_{a_1}, k_a^*), \ldots, y(t_{a_Q}, k_a^*), \tag{11}$$

with $y \in \{c_1, \ldots, c_C\}$ with C being the number of classes. For example, when considering the HW/HD over a byte we have $C = 9$ with $\{c_1, \ldots, c_9\} = \{0, \ldots, 8\}$. We denote the vector of output probabilities of a classifier for the i^{th} measurement sample as

$$\mathbf{p}_i = p_{i,c_1}, \ldots, p_{i,c_C}, \tag{12}$$

where $i = 1, \ldots, Q$. For each sample i in the test set, classifier predicts a class label $\tilde{y}_{a_i}$ corresponding to the maximal output probability in $\mathbf{p}_i$, i.e.,

$$\tilde{y}_{a_i} = \underset{\{c_1,\ldots,c_C\}}{\arg\max}\ \mathbf{p}_i. \tag{13}$$

The accuracy is then computed as

$$\frac{1}{Q} \sum_i \mathbb{1}_{\tilde{y}_{a_i} = y_{a_i}} \tag{14}$$

with $\mathbb{1}$ being the indicator function. Accordingly, accuracy only takes into account the most likely label predictions without their exact values of probabilities (see Eq. (13)) and predictions over i are considered independently (see Eq. (14)).

Contrary, GE and SR are computed regarding the secret key k_a^* and output probability values are accumulated as relying on a maximum-likelihood approach (like template attack). In particular, for a given plaintext t_{a_i} let us denote the set of keys corresponding to the class c_i through $y(t_{a_i}, k) = c_i$ as

$$K_{c_i;t_{a_i}} = \{k_1, \ldots, k_{\delta_{c_i}}\}, \tag{15}$$

where δ_{c_i} is the amount of keys corresponding to one class c_i. For example, for $y(t_{a_i}, k) = HW(\texttt{Sbox}[t_{a_i} \oplus k])$ we have $\delta_{c_i} = \binom{c_i}{8}$. Now, for each class c_i the probability $p_{i,k}$ of each key k in $K_{c_i;t_{a_i}}$ is set to p_{i,c_i}. Given Q amount of samples in the test set and uniformly chosen plaintexts t, the likelihood for each k is calculated as

$$p_k^Q = \sum_{i=1}^{Q} p_{i,k}. \tag{16}$$

A classifier now decides for the key $\tilde{k}^Q$ with the maximal likelihood, i.e.,

$$\tilde{k}^Q = \arg\max_k p_k^Q. \tag{17}$$

The SR is then computed over an amount of E experiments as

$$\frac{1}{E} \sum_{e=1}^{E} \mathbb{1}_{\tilde{k}^Q = k_a^*}. \tag{18}$$

Note that, normally for each experiment e an independent and uniformly distributed set of plaintexts and a new secret key k_a is chosen. Taking p_k^Q in Eq. (16) and sorting it in descending order of likelihood, the GE over E experiments is the average position of k_a in the sorted vector.

5.2 Label Prediction vs Fixed Secret Key Prediction

The first difference between accuracy and SR/GE is that for accuracy each label prediction in the test set is considered independently, whereas SR/GE is computed regarding a fixed secret key. More precisely, comparing Eq. (14) and Eq. (18) one can see that accuracy is measured regarding class labels y averaged over Q amount of samples, whereas SR (and GE) is measured in respect to the secret key k_a accumulated over Q amount of samples and averaged over E experiments. Moreover, SR/GE are taking into account the exact value of the output probability of each class (see Eq. (16)), whereas accuracy only considers which class corresponds to the maximal output probability (see Eq. (13)).

Based on these differences, we can derive that a low accuracy may not indicate that the SR is reaching the threshold value of 90% using a higher amount of traces

(or similarly the GE). Let us consider a toy example with 3 classes c_1, c_2, c_3 with $k_a = 2$ for $Q = 3$ and

$$p_1 = \{0.4, 0.5, 0.1\},\ p_2 = \{0.3, 0.4, 0.3\},\ p_3 = \{0.1, 0.4, 0.5\}, \quad \text{and} \tag{19}$$

$$y_1 = c_1,\ y_2 = c_3,\ y_3 = c_2. \tag{20}$$

Moreover, we consider the simplified case that each class label corresponds to only one key, i.e., $K_{c_i} = k_i$, and $E = 1$. According to Eq. (19) and (20), the accuracy is 0%, but the SR will reach 100% for ≥ 1 sample(s), as

$$p^1_{k=1} = 0.4,\ p^2_{k=1} = 0.7,\ p^3_{k=1} = 0.8, \tag{21}$$

$$p^1_{k=2} = 0.5,\ p^2_{k=2} = 0.9,\ p^3_{k=2} = 1.3, \tag{22}$$

$$p^1_{k=3} = 0.1,\ p^2_{k=3} = 0.4,\ p^3_{k=3} = 0.9. \tag{23}$$

However, the opposite conclusion might hold: A high accuracy may indicate that the SR/GE is reaching the threshold value of 90% using a lower amount of traces. Note that, the differences between accuracy and SR/GE derived in this subsection are not based on the imbalancedness of the class labels, as also our toy example shows.

5.3 Global Accuracy vs Class Accuracies

When considering the case of imbalanced classes, as e.g., $y(t_{a_i}, k_a) = HW(\texttt{Sbox}[t_{a_i} \oplus k_a])$, the amount of information in respect to the secret key k_a is varying depending on the observed class $y(t_{a_i}, k)$ (see δ_{c_i} in Eq. (15) or the explanation in Subsection 2.2). Accordingly, accurately predicting the classes corresponding to a smaller δ_c may improve SR/GE more than accurately predicting classes with a higher δ. Therefore, the class accuracies with smaller δ may be more relevant than the class accuracies for higher δ or the global accuracy (i.e. averaged over all classes).

Note that this observation may bring new direction for future work on how to derive (or tune) classification techniques which are more accurate for classes contributing more information to the secret key.

Remark 1. Even though our previous experiments demonstrated the beneficial impact of balancing techniques like SMOTE, a straightforward approach to compensate the effect of global vs class accuracies may be to not consider the Hamming weight and directly use the intermediate value e.g., $\texttt{Sbox}[t_{a_i} \oplus k_a]$. However, this approach has its own merits and demerits (see also Subsection 2.2). Using the intermediate value directly increases the number of classes, for which a larger training set is required. As larger number of classes are present within the same margins, the classification becomes more prone to noise. The aforementioned problems may be partly solved if a large enough set of profiling traces are provided. That is not always possible due to several practical shortcomings. To name a few, countermeasures can restrict the number of available traces for a given key. Similarly, time bounded certification process also does not give the

luxury to collect a large number of traces. Accordingly, to cope up with these issues in absence of infinite number of traces, considering HW/HD classes with proposed data balancing techniques can prove as a practical solution.

6 Conclusions and Future Work

In this paper, we focused on the problem of highly imbalanced datasets and classification. Side-channel analysis offers realistic scenarios where we encounter datasets that have large amounts of noise and are highly imbalanced (where some classes are on average 70 times more represented than some other classes). Additionally, the SCA domain uses specific metrics to assess the performance of classifiers where the end goal is to estimate the number of measurements needed for a successful attack.

We conducted a detailed analysis on techniques that can help in imbalanced data scenarios and we show that SMOTE is especially useful in some difficult (noisy) scenarios. Interestingly, we observe significant discrepancy between ML metrics and SCA metrics, which indicates that estimating the success of a potential side-channel attack is a difficult task if we rely solely on ML metrics. In such scenarios, accuracy is not a reliable metric to predict the ability of key recovery in SCA.

In future work, we plan to continue working on the last two questions from Section 4.2. Designing a new ML metric that reflects the SCA behavior better seems to be very difficult (or even impossible), but using SCA metrics in ML process is possible. The main question is whether such an approach would be acceptable from the computational complexity perspective. Finally, we considered some well-known methods for handling imbalanced data which were highly efficient on our datasets but naturally, the choice of the methods was not exhaustive.

References

1. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (December 2006) ISBN 0-387-30857-1, `http://www.dpabook.org/`.
2. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of CRYPTO'99. Volume 1666 of LNCS., Springer-Verlag (1999) 388–397
3. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. Volume 2523 of LNCS., Springer (August 2002) 13–28 San Francisco Bay (Redwood City), USA.
4. Heuser, A., Zohner, M.: Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Schindler, W., Huss, S.A., eds.: COSADE. Volume 7275 of LNCS., Springer (2012) 249–264
5. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering **1** (2011) 293–302 10.1007/s13389-011-0023-x.

6. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.: Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). In: COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers. (2015) 20–33
7. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES - Reaching the limit of side-channel attacks with a learning model. J. Cryptographic Engineering **5**(2) (2015) 123–139
8. Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A Machine Learning Approach Against a Masked AES. In: CARDIS. Lecture Notes in Computer Science, Springer (November 2013) Berlin, Germany.
9. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
10. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). (May 2015) 106–111
11. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 45–68
12. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: Side-channel analysis and machine learning: A practical perspective. In: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017. (2017) 4095–4102
13. Standaert, F.X., Malkin, T., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: EUROCRYPT. Volume 5479 of LNCS., Springer (April 26-30 2009) 443–461 Cologne, Germany.
14. Doget, J., Prouff, E., Rivain, M., Standaert, F.X.: Univariate side channel attacks and leakage modeling. J. Cryptographic Engineering **1**(2) (2011) 123–144
15. Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: Dual-rail transition logic: A logic style for counteracting power analysis attacks. Computers & Electrical Engineering **35**(2) (2009) 359 – 369 Circuits and Systems for Real-Time Security and Copyright Protection of Multimedia.
16. Standaert, F.X., Peeters, E., Quisquater, J.J.: On the masking countermeasure and higher-order power analysis attacks. In: International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II. Volume 1. (April 2005) 562–567 Vol. 1
17. TELECOM ParisTech SEN research group: DPA Contest (4^{th} edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
18. Coron, J., Kizhvatov, I.: An Efficient Method for Random Delay Generation in Embedded Software. In: Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. (2009) 156–170
19. Matthews, B.: Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA) - Protein Structure **405**(2) (1975) 442 – 451
20. Cohen, J.: A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement **20**(1) (1960) 37–46
21. Boughorbel, S., Jarray, F., El-Anbari, M.: Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. PLOS ONE **12**(6) (06 2017) 1–17

22. Jeni, L.A., Cohn, J.F., De La Torre, F.: Facing Imbalanced Data–Recommendations for the Use of Performance Metrics. In: Proceedings of the 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction. ACII '13, Washington, DC, USA, IEEE Computer Society (2013) 245–251
23. He, H., Garcia, E.A.: Learning from Imbalanced Data. IEEE Trans. on Knowl. and Data Eng. **21**(9) (September 2009) 1263–1284
24. Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D.: Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? Journal of Machine Learning Research **15** (2014) 3133–3181
25. Akbani, R., Kwek, S., Japkowicz, N.: Applying Support Vector Machines to Imbalanced Datasets. In Boulicaut, J.F., Esposito, F., Giannotti, F., Pedreschi, D., eds.: Machine Learning: ECML 2004, Berlin, Heidelberg, Springer Berlin Heidelberg (2004) 39–50
26. Dittman, D.J., Khoshgoftaar, T.M., Napolitano, A.: The Effect of Data Sampling When Using Random Forest on Imbalanced Bioinformatics Data. In: 2015 IEEE International Conference on Information Reuse and Integration. (Aug 2015) 457–463
27. Fan, R.E., Chen, P.H., Lin, C.J.: Working Set Selection Using Second Order Information for Training Support Vector Machines. J. Mach. Learn. Res. **6** (December 2005) 1889–1918
28. Breiman, L.: Random Forests. Machine Learning **45**(1) (2001) 5–32
29. Krawczyk, B.: Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence **5**(4) (Nov 2016) 221–232
30. Longadge, R., Dongre, S.: Class Imbalance Problem in Data Mining Review. CoRR **abs/1305.1707** (2013)
31. Ertekin, S., Huang, J., Giles, C.L.: Active Learning for Class Imbalance Problem. In: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '07, New York, NY, USA, ACM (2007) 823–824
32. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. J. Artif. Int. Res. **16**(1) (June 2002) 321–357
33. Batista, G.E.A.P.A., Prati, R.C., Monard, M.C.: A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. SIGKDD Explor. Newsl. **6**(1) (June 2004) 20–29
34. Wilson, D.L.: Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. IEEE Transactions on Systems, Man, and Cybernetics **SMC-2**(3) (July 1972) 408–421
35. James, G., Witten, D., Hastie, T., Tibsihrani, R.: An Introduction to Statistical Learning. Springer Texts in Statistics. Springer New York Heidelbert Dordrecht London (2001)
36. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12** (2011) 2825–2830