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

Stjepan Picek[1,2], Annelie Heuser[3], Alan Jovic[4],
Shivam Bhasin[5], and Francesco Regazzoni[6]

[1] Delft University of Technology, Delft, The Netherlands
[2] LAGA, Department of Mathematics, University of Paris 8 (and Paris 13 and CNRS), France
[3] Univ Rennes, Inria, CNRS, IRISA, France
[4] University of Zagreb Faculty of Electrical Engineering and Computing, Croatia
[5] Physical Analysis and Cryptographic Engineering, Temasek Laboratories at Nanyang Technological University, Singapore
[6] University of Lugano, Switzerland
S.Picek@tudelft.nl,annelie.heuser@irisa.fr, Alan.Jovic@fer.hr, sbhasin@ntu.edu.sg, regazzoni@alari.ch

Abstract. We concentrate on machine learning techniques used for profiled side-channel analysis in the presence of 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 and we show that most of them help in mounting successful attacks when the data is highly imbalanced. Especially, the results with the SMOTE technique are encouraging, since we observe some scenarios where it reduces the number of necessary measurements 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 finding opens a need to revisit the previous works and their results 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

Side-channel Attacks (SCA) is a serious threat, which exploits weaknesses in the physical implementation of cryptographic algorithms [1]. The weakness stems from basic device physics of underlying computing elements i.e., CMOS cells, which makes it hard to eliminate such threats. SCA 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.

Template attack is recognized as the most powerful SCA, at least from an information theoretic point of view [2]. There, the attacker first profiles the behavior of a device similar to the targeted one, followed by exploitation of the profiled information to finalize the attack. In practice, there are many scenarios where machine learning (ML) techniques are outperforming template attack (for instance, when the profiling set is small). Thus, several works explored the use of machine learning (and more recently, deep learning) in the context of SCA [3,4,5,6,7,8,9,10,11].

In order to run SCA, one may select a leakage model where common examples are the intermediate value, the Hamming weight, and the Hamming distance models. As an example, let us consider a generator returning random numbers between 0 and 255. If we take output values as class labels, we have uniformly distributed data. Simpler models would be the Hamming weight (HW) (and the Hamming distance (HD)), which are commonly used 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 machine learning techniques often classify all measurements as the majority class (Hamming weight 4), see e.g., [12]. Then, accuracy will reach around 27% on average, but such a 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 the data science community and there exists no definitive solution to 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.

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 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 ML and SCA 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 domain-relevant 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 the imbalanced data and metrics also remain in 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 in the context of SCA.
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, enable us to reach excellent results, where we reduce the number of traces needed for a successful attack 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 of, such an analysis has not been done before.
6. We extend the present study to include a deep learning method, like CNN, to show that deep learning equally suffers from the curse of imbalanced data.

2 Background

2.1 Profiled 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 the 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, thus revealing information 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 substitution look-up table 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 methods is template attack (TA) [2]. 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. Contrarily, HD/HW model works on a range of devices, when not many details of the underlying implementations are known to provide a good starting point for evaluations. The leakage model can then be improved faster after a few hints on the implementations are derived.

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

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

Obviously, observing an 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 value gives to an attacker to recover the secret key k_a^*. For example, knowing t and observing an HW of 4 gives an attacker 70 possible secret keys, whereas observing an HW of 0 or 8 leads to only one possible secret key. Accordingly, the occurrence of HW classes close to 4 is 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 a 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 less misclassification. The impact is even higher when dealing with 16-bit (65,536 vs 17), 32-bit (4,294,967,296 vs 33) or wider intermediate values. The disadvantages of the HW model, apart from imbalance, are less information on the secret key as multiple intermediate value classes map to the 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 settings, we target 1) high-SNR unprotected implementation on a smartcard, 2) low-SNR implementation on a smartcard protected with the randomized delay countermeasure, and 3) low-SNR unprotected implementation on FPGA.

DPAcontest v4 DPAcontest v4 provides measurements of a masked AES software implementation [17]. As we are interested in an 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 the 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

$$y(t_{b_1}, k^*) = HW(\texttt{Sbox}[t_{b_1} \oplus k^*] \oplus \underbrace{m}_{\text{known mask}}), \tag{2}$$

where t_{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 (AES_RD) Next, 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[7] [18]. Adding random delays to the normal operation of a cryptographic algorithm has an effect on the misalignment of important features, which in turns makes the attack more difficult. 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 was shown to be prone to deep learning based side-channel [11]. However, since it is a quite often used countermeasure in commercial products, while not modifying the leakage order (like masking), we use it as a target case study. We additionally keep the misalignment countering features of deep learning out of scope in order to study the impact of imbalanced classes only. In the rest of the paper, we denote this dataset as the AES_RD.

Unprotected AES-128 on FPGA (AES_HD) Finally, we target an unprotected implementation of AES-128, which was written in VHDL in a round based architecture that 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[8]. 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.,

$$y(t_{b_1}, t_{b_2}, k^*) = HW(\underbrace{\texttt{Sbox}^{-1}[t_{b_1} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{t_{b_2}}_{\text{ciphertext byte}}), \tag{3}$$

where t_{b_1} and t_{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

[7] Trace set publicly available at https://github.com/ikizhvatov/randomdelays-traces

[8] Trace set publicly available at https://github.com/AESHD/AES_HD_Dataset. Note we provide a full dataset consisting of 1 250 features but here we use only the 50 most important features that are selected with Pearson correlation.

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.

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 called 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 [21,22,23]. 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 (correctly classified positive), TN to true negative (correctly classified negative), FP to false positive (falsely classified positive), and FN to false negative (falsely classified negative) instances. TP, TN, FP and FN are well-defined for hypothesis testing and binary classification problems. In the multiclass classification, they are defined in one class–vs–all other classes manner, and are calculated from the confusion matrix. The calculation of TP, TN, FP, and FN instances for an actual class 0 in a three-class classification problem confusion matrix example is shown in Table 2. Note that the evaluation metrics in Eqs. (4)- (9) consider mean values of TP, TN, FP, and FN taken from all the individual classes included in the multiclass problem, unless otherwise stated.

Table 2: Calculation of TP, TN, FP, and FN instances for actual class 0.

Predicted (%)			Actual
0	1	2	
12.1	8.12	4.06	0
1.62	16.2	9.75	1
2.43	3.25	42.27	2

P_{Obs} is the percentage of observed agreement among observers, and P_{Chance} is the agreement expected by pure chance. 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 Most of the time, in side-channel analysis an adversary in not only interested to predict the labels $y(\cdot, k_a^*)$ in the attacking phase, but aims at revealing the secret key k_a^*. Common measures are the success rate (SR) and the guessing entropy (GE) of a side-channel attack. In particular, let us assume, given Q amount of samples in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability with $|\mathcal{K}|$ being the size of the keyspace. So, g_1 is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate.

The success rate is defined as the average empirical probability that g_1 is equal to the secret key k_a^*. The guessing entropy is the average position of k_a^* in $\mathbf{g}$. As SCA metrics, besides plotting GE and SR, we report the number of traces needed to reach a success rate SR of 0.9 as well as a guessing entropy GE of 10. We use '–' in case these thresholds are not reached within the test set.

Both SR and GE can be applied to a variety of SCA distinguishers to evaluate an attack. These distinguishers may include Pearson's correlation [24], mutual

information [25], maximum likelihood (for templates or linear regression), etc. Evaluation labs often resort to these distinguishers and metrics for common criteria evaluations of security critical products. There is much space for adopting ML-based distinguishers in such evaluations.

FIPS standard is based on a different methodology called conformance-style testing. The idea here is to detect the presence of leakage in SCA measurement rather than exploiting it for key recovery. Test vector leakage assessment [26] is a popular choice for conformance based testing, which uses t-test to detect the presence of side-channel leakage. Few other works look into alternate statistical tools for leakage assessment [27]. The use of ML in leakage assessment is still an open question.

2.5 Classifiers

We use radial kernel support vector machines (SVM) and Random Forest (RF). These 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 [28]. 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 [29,30].

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: the 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 [31]. The multiclass support is handled according to a one-vs-one scheme. The time complexity for SVM with the radial kernel is $\mathrm{O}\left(D \cdot N^3\right)$, 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 [32]. 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 $\mathrm{O}\left(I \cdot k \cdot NlogN\right)$. 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 the 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, 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 and are derived optimizing accuracy. Usually, within an 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 extremely imbalanced data [33]. By referring to Table 1, we see that our HW scenario belongs to imbalanced scenarios, but approaching extremely imbalanced scenarios.

3.1 Handling Imbalanced Data

There are essentially two main approaches to improve the classification results by avoiding model overfitting to majority class in imbalanced data setting:

1. Data-level methods that modify the measurements by balancing distributions (which falls under the term data augmentation).
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) [34], or active learning [35]. 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 the 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 [36]. Here, overfitting means that the machine learning algorithm adapts to the training set too well and thus loses the ability to generalize to another dataset (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.

Random Undersampling Random undersampling undersamples all classes except the least populated one (here, HW 0 or HW 8). This is a very simple technique to balance the data but one that can suffer from two important drawbacks. The first one is that 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 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 that 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.

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 [37].

Synthetic Minority Oversampling Technique The second method is SMOTE, a well-known resampling method that oversamples by generating synthetic minority class instances [36]. 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 $k = 5$ [36]. 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.

Cagli et al. proposed a custom data augmentation (DA) technique to fight overfitting, where augmented traces are generated from the original traces by applying a uniform (random) shift. The augmented traces are added to all the classes. In this setting, SMOTE can be considered as a general case of the DA proposed in [11]. SMOTE not only adds synthetic examples with random shift but it also applies other transformations along with the goal of balancing the classes.

Synthetic Minority Oversampling Technique with Edited Nearest Neighbor SMOTE + ENN [37] combines oversampling used by SMOTE and data cleaning by Edited Nearest Neighbor (ENN) method, originally proposed by Wilson [38]. 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 ML metrics vs SCA metrics

Most previous works on machine learning techniques for SCA used ML evaluation metrics to tune (and even compare) their performances. However, it has been noted already (e.g. [3]) that ML metrics may not coincide with SCA metrics. In order to better understand their diversities, we formally discuss and highlight two differences between accuracy and SR/GE. 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.

4.1 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, the 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)).

Contrarily, GE and SR are computed regarding the secret key k_a^* and output probability values are computed over a set of i measurements. 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 log-likelihood for each k is calculated as

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

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

$$\tilde{k}^Q = \underset{k}{\arg\max}\ \log(p_k^Q). \tag{17}$$

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 the descending order of likelihood, the GE over E experiments is the average position of k_a in the sorted vector.

4.2 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 with 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

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

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

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

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

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

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

Still, 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, but are general in nature.

4.3 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_i} may improve SR/GE more than accurately predicting classes with a higher δ_{c_i}. Therefore, the class accuracies corresponding to smaller δ_{c_i} may be more relevant than the class accuracies for higher δ_{c_i} or the global accuracy (i.e., averaged over all classes). Note that this observation may bring a 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. Note that the same arguments given for accuracy apply also for recall. Even though recall is computed class-wise, it does not consider the imbalancedness, and the arguments given in Subsect. 4.2 and Subsect. 4.3 follow similarly.

5 Experimental Validation and Discussion

First, we randomly select a number of measurements from each dataset. From DPAv4 and AES_HD datasets, we select 75 000 measurements, while for the AES_RD 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 the Pearson correlation coefficient. Pearson correlation coefficient measures linear dependence between two variables, x and y, in the range $[-1, 1]$, where 1 is a total positive linear correlation, 0 is no linear correlation, and -1 is the total negative linear correlation [39].

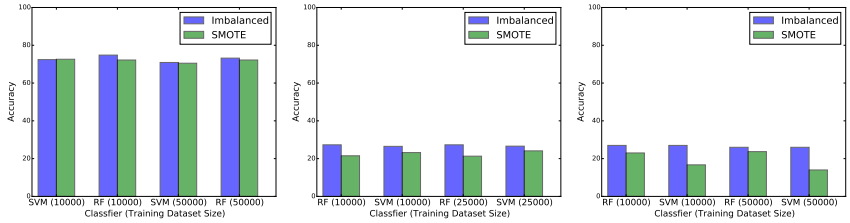
We divide the traces into training and testing sets, where each test set has 25 000 measurements. We experiment with three training set sizes, where the measurements are selected randomly from the full training set: 1 000, 10 000, and 50 000 measurements (25 000 for AES_RD). We use 3 datasets with significantly 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 AES_RD) 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. We report results from the testing phase only, as these are more relevant than the training set results in assessing the actual classification strength of the constructed models. All the experiments are done with the scikit-learn library [40] from Python.

5.1 Results

We tested all the classifiers with all the datasets with varying training set sizes and computed the relevant metrics. Interested readers can refer to Tables 7 to 9 in Appendix A. These tables provide the classification results for the original (imbalanced), class weight balanced, random oversampling, SMOTE, and SMOTE+ENN datasets. 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).

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 an imbalance within the dataset, data sampling techniques may increase SR and GE significantly. Comparing techniques we investigated, the SMOTE technique performs the best, followed by Random Oversampling, class weight balancing, and finally, SMOTE+ENN. Here, we focus on three main metrics: accuracy (Fig. 1), success rate, and guessing entropy (Fig. 2). We compare the results for the imbalanced case (i.e., original) and after applying the SMOTE method (i.e., the method with the best results). We also provide insights on how other tested balancing methods compare against SMOTE.



(a) DPA v4 dataset. (b) AES_RD dataset. (c) AES_HD dataset.

Fig. 1: Accuracy for imbalanced and SMOTE on all three datasets.

DPAcontest v4 dataset has the highest SNR of all the considered datasets (and is consequently the easiest one). Here, we see that machine learning algorithms do not have problems in dealing with imbalanced data – Figure 1a and Table 7. 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 a much larger number of traces needed when compared to the imbalanced case. Finally, SMOTE and SMOTE+ENN techniques show that, although accuracy could be even improved over the 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 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).

Table 3: Confusion matrix for DPAcontest v4 imbalanced dataset, SVM with $C = 1, \gamma = 1$, 10 000 measurements in the training phase and 25 000 measurements in the testing phase. Results are given in percentages.

Predicted (%)									Actual
0	1	2	3	4	5	6	7	8	
0	0.26	0.17	0	0	0	0	0	0	0
0	0.15	2.84	0.02	0	0	0	0	0	1
0	0	8.47	2.68	0.01	0	0	0	0	2
0	0	1.30	16.59	3.57	0.01	0	0	0	3
0	0	0.02	2.97	21.64	2.87	0.01	0	0	4
0	0	0	0.02	3.80	16.48	1.68	0	0	5
0	0	0	0	0.03	2.51	8.27	0.14	0	6
0	0	0	0	0	0.01	2.33	0.70	0	7
0	0	0	0	0	0	0.03	0.29	0.03	8

Table 4: Confusion matrix for AES_RD imbalanced dataset, SVM with $C = 1, \gamma = 1$, 10 000 measurements in the training phase and 25 000 measurements in the testing phase. Results are given in percentages.

Predicted (%)									Actual
0	1	2	3	4	5	6	7	8	
0	0	0	0	0.39	0	0	0	0	0
0	0	0	0	2.90	0	0	0	0	1
0	0	0	0	11.06	0	0	0	0	2
0	0	0	0	21.92	0	0	0	0	3
0	0	0	0	27.26	0	0	0	0	4
0	0	0	0	21.68	0	0	0	0	5
0	0	0	0	11.10	0	0	0	0	6
0	0	0	0	3.23	0	0	0	0	7
0	0	0	0	0.41	0	0	0	0	8

When considering the AES_RD dataset, we see that the problem is much more difficult (see Figure 1b and Table 8). In fact, for the imbalanced dataset, only in a few cases are we 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 we are now able to reach the thresholds on two more cases when considering GE and in 4 cases when considering SR. SMOTE, although, strictly speaking, is successful in one less occasion, brings even more significant improvements, since we now need fewer traces to successfully reach the thresholds. We emphasize the 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. With SMOTE, we are able to reach an 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 to the class weight balancing technique. Generally speaking, we observe that RF is more successful than SVM, which we attribute to the RF's capability to deal with noisy measurements. Finally, this dataset is a good example of the problem of assigning all the measurements to the majority class, as seen in Table 4. Regardless of the number of measurements, with such imbalancedness, we would never be able to break this target, despite a relatively good accuracy of 27.3%.

Finally, for the AES_HD dataset, the results could be considered somewhere in between the previous two cases: the dataset characteristics and imbalancedness represent bigger problems than for DPAcontest v4, but not as significant ones as for the AES_RD dataset. The results are given in Figure 1c and Table 9. We observe that, for this scenario, class weight balancing is actually deteriorating the behavior of classifiers, as in fewer cases are we able to actually reach the threshold. Contrarily, 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 fewer 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 a 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 a suitable measure since the 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 the imbalanced scenario with RF and 10 000 measurements, and for SMOTE, RF and 10 000 measurements, in Tables 5 and 6, respectively. Differing from Table 3, 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 correct predictions for 7 classes.

Table 5: Confusion matrix for the AES_HD imbalanced dataset, RF with 1 000 iterations, 10 000 measurements in the training phase and 25 000 measurements in the testing phase. Results are given in percentages.

Predicted (%)									Actual
0	1	2	3	4	5	6	7	8	
0	0	0.004	0.05	0.28	0.06	0	0	0	0
0	0	0.02	0.32	2.32	0.36	0	0	0	1
0	0	0.05	1.09	8.18	1.54	0	0	0	2
0	0	0.11	2.26	16.69	2.97	0.01	0	0	3
0	0	0.06	2.38	20.45	4.11	0	0	0	4
0	0	0.10	2.05	16.70	3.22	0	0	0	5
0	0	0.03	0.91	8.32	1.74	0.004	0	0	6
0	0	0.01	0.27	2.32	0.50	0	0	0	7
0	0	0.004	0.02	0.28	0.06	0	0	0	8

Table 6: Confusion matrix for the AES_HD after SMOTE, RF with 1 000 iterations, 10 000 measurements in the training phase (plus the measurements obtained with SMOTE in latter) and 25 000 measurements in the testing phase. Results are given in percentages.

Predicted									Actual
0	1	2	3	4	5	6	7	8	
0	0.01	0.09	0.08	0.08	0.08	0.04	0.004	0.004	0
0	0.07	0.63	0.49	0.78	0.58	0.30	0.13	0.3	1
0.01	0.17	2.13	1.78	2.92	2.09	1.23	0.45	0.08	2
0.03	0.34	4.36	3.45	5.76	4.44	2.46	1.10	0.08	3
0.01	0.41	4.76	3.98	7.70	5.91	3.25	1.49	0.11	4
0.02	0.30	3.79	3.36	5.83	4.63	2.60	1.40	0.12	5
0.01	0.17	1.73	1.65	2.89	2.50	1.32	0.69	0.04	6
0.004	0.02	0.49	0.46	0.86	0.63	0.42	0.19	0.01	7
0	0.01	0.4	0.4	0.12	0.10	0.03	0.01	0	8

In Figures 2a until 2d, we depict guessing entropy and success rate results for all datasets, when using either imbalanced datasets (full lines) or those after applying SMOTE (dashed lines). We depict the results for both SVM and RF classifiers illustrating the significant improvements for the AES_RD and AES_HD datasets. Observe how guessing entropy and success rate clearly show improvements after SMOTE despite the fact that accuracy indicates worse performance (cf. Figures 1a until 1c).

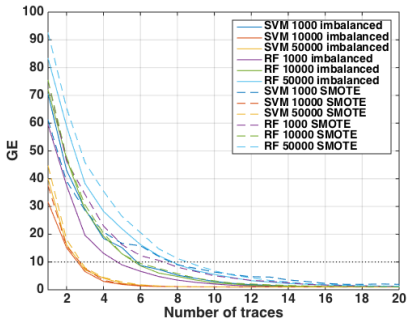
Remark 2. Even though our previous experiments demonstrated the beneficial impact of balancing techniques like SMOTE, a straightforward approach to compensate the effect of global vs class accuracies may be not to consider the Hamming weight and directly use the intermediate value e.g., $\texttt{Sbox}[t_{a_i} \oplus k_a]$. 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 a 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 the absence of an infinite number of traces, considering HW/HD classes with proposed data balancing techniques can prove as a practical solution.

5.2 Discussion

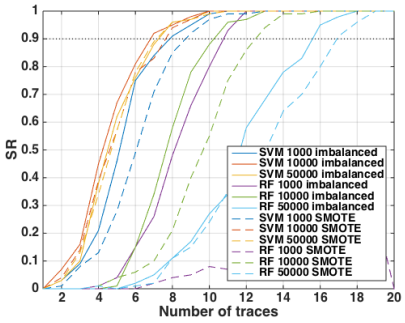
On a more general level, our experiments indicate that none of the ML metrics we tested can be used as a reliable indicator of SCA performance when dealing with imbalanced data. In the best case, machine learning metrics can serve as an indicator of performance, where high value means the attack should be possible, while low value could indicate that the attack would be difficult or even impossible. But as it can be seen from our results, those metrics are not reliable. Sometimes a small difference in the machine learning metric means a large difference in the SCA metrics, but this cannot be said in general. We also see situations where ML metrics indicate a 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 have higher values of SCA metrics. To conclude, we formally and experimentally show that there is no clear connection between accuracy and GE/SR. Still, there are general answers (or intuitions) we can give.

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

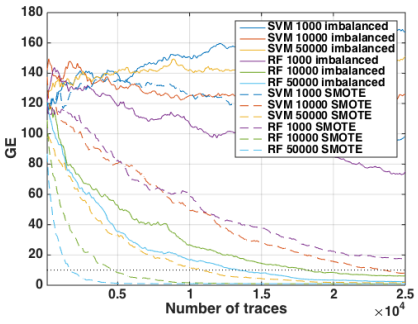
A The answer is no since our experiments clearly show that sometimes accuracy can be used to infer about SCA success, but is often misleading. This is also very important from the perspective where SCA community questions whether a small difference in accuracy (or other machine learning metrics) means anything for SCA. Unfortunately, our experiments show there is no



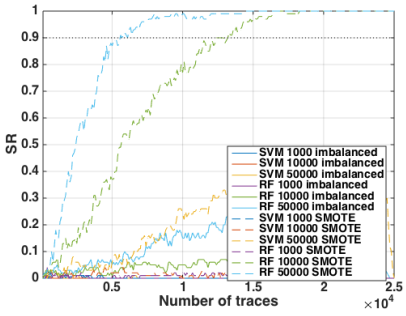
(a) Guessing entropy (GE) on DPAcontest v4.



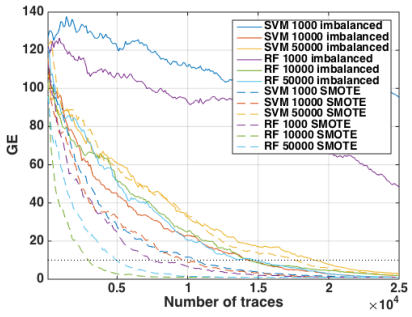
(b) Success rate (SR) on DPAcontest v4.



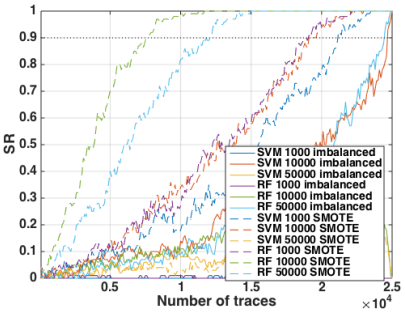
(c) Guessing entropy (GE) on AES_RD.



(d) Success rate (SR) on AES_RD.



(e) Guessing entropy (GE) on AES_HD.



(f) Success rate (SR) on AES_HD.

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

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 no, again. We experimented with 7 different machine learning metrics and none of them gave a good indication of SCA behavior over different scenarios.

Q If we concluded that accuracy is not an appropriate measure, what sense does it make to evaluate other ML metrics on the test set, since, still, accuracy is used in the 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 the 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.

6 SMOTE and Other Classifiers

Our results showed how various balancing techniques, and especially SMOTE, can help ML classifiers to achieve better results. Such results are usually not characterized by an improved accuracy, but by an improved success rate and/or guessing entropy. The question is whether such an improvement in performance

can be observed with only "standard" machine learning techniques, or other classifiers can benefit from it also. Here, we experiment with 2 types of deep learning: multilayer perceptron (MLP) and Convolutional Neural Networks (CNN), and with a standard technique in SCA community: template attack (TA) [2], its pooled version (TA p.) [41], and stochastic attack (SA) [42].

The multilayer perceptron is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers (at least three) of nodes in a directed graph, where each layer is fully connected to the next one and training of the network is done with the backpropagation algorithm. If there is more than one hidden layer, we can already talk about deep learning. We experiment with activation function $[relu, tanh]$ and number of hidden layers/nodes $[(50, 10, 50), (50, 30, 20, 50), (50, 25, 10, 25, 50)]$.

CNNs are a specific type of neural networks which were first designed for 2-dimensional convolutions as it was inspired by the biological processes of animals' visual cortex [43]. We use computation nodes equipped with 32 NVIDIA GTX 1080 Ti graphics processing units (GPUs). Each of it has 11 Gigabytes of GPU memory and the 3 584 of GPU cores. Specifically, we implement the experiment with the Tensorflow [44] computing framework and PyTorch [45]. Here, we tested a number of architectures given in related work [46,11,47] and we found the best for our experiments the one from Maghrebi et al. [46]. Naturally, all the architectures needed to be adjusted to the case that we use only 50 features. The CNN we use consists of: a convolutional layer with 8 filters, activation size of 16, and *relu* activation function, dropout, Max Pooling layer, convolutional layer with 8 filters, activation size of 8, and *tanh* activation function, dropout, and fully connected layer. Finally, we use the Softmax activation function in the classification layer combined with the Categorical Cross Entropy loss function. The learning rate is 0.0001, the optimizer is *adam*, batch size is 256, and the number of epochs is 1 000.

Note that by design (pooled) template attack (and similarly, stochastic attack) do not suffer from the problem of imbalanced classes per se. TA does not rely on an optimization problem maximizing the accuracy as (most) "standard" machine learning techniques, but on using the maximum likelihood principle over each class. Accordingly, imbalancedness may only affect the performance if some classes do not contain a sufficient amount of traces such that the practical estimation of probabilities (i.e., covariance matrices in case of the normal assumption) pose statistical imprecision.

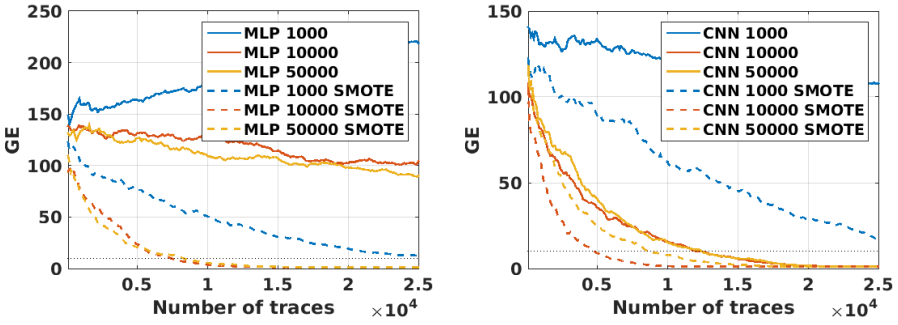
Since the AES_HD dataset improves the most after using SMOTE (and due to the lack of space), we provide the guessing entropy results here. See Appendix A for detailed results with different metrics and Appendix B for DPAv4 and AES_RD datasets for guessing entropy.

For DPAcontest v4, when considering MLP, the results are similar to the behavior observed with SVM/RF. The improvements after SMOTE, if any, are quite small, which is to be expected since the results on the imbalanced dataset are already very good and do not require further augmentation. The worst behavior can be seen for SMOTE when augmenting the dataset with 1 000

measurements. The problem here is that the augmentation procedure does not have enough information from the original dataset (when considering those rare classes) to build high quality synthetic examples. Almost identical behavior can be seen for the CNN experiments. When considering TA and TA pooled, we see that SMOTE actually deteriorates the results significantly. Stochastic attack works much worse after applying SMOTE than when considering imbalanced datasets. We depict guessing entropy for MLP, CNN, TA, and pooled TA in Figures 5a until 5d, respectively.

For the AES_RD dataset we see that for MLP, SA, and CNN, SMOTE does not bring (significant) improvements. Actually, the only improvement can be seen for the case when using SMOTE on a training dataset of size 1 000. Differing from the previous scenario, here we see that SMOTE also helps the smallest dataset when using template attack and (in smaller extent) pooled template attack. Detailed guessing entropy results for the AES_RD dataset are depicted in Figures 6a until 6d.

Finally, when considering AES_HD, Figures 3a until 4b depict guessing entropy for MLP, CNN, TA, and TA pooled. When considering MLP, we observe significant improvements after SMOTE, where we are actually able to break implementation even with the smallest training set size. At the same time, when considering the imbalanced dataset, for the same result, not even the biggest dataset was sufficient (which is 25 times larger). For CNN the improvements after SMOTE are also significant, reducing the number of required measurements several times. As for the AES_RD, similarly here we see that SMOTE also improved the results for template attack when considering 1 000 measurements. For other dataset sizes, as well as for pooled template attack, we see a deterioration of results after SMOTE. When considering SA, we see that the results are worse for the SMOTE scenario than for the imbalanced dataset.

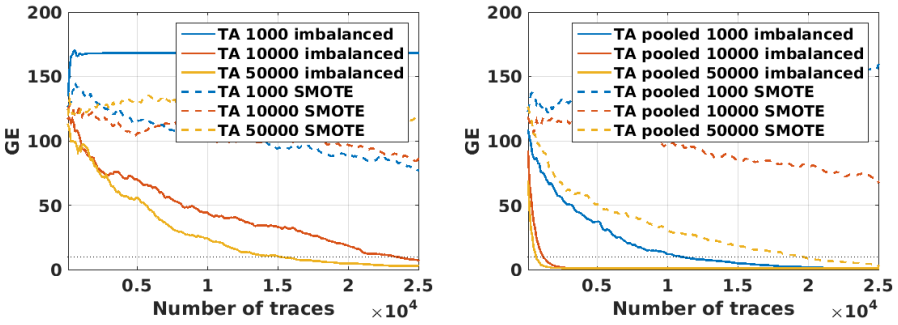


(a) MLP (b) CNN

Fig. 3: Guessing entropy for imbalanced and SMOTE on AES_HD, deep learning

After experimenting with 3 different classifier techniques in this section, we can observe 3 distinct behaviors. For the first deep learning technique we consider – MLP, we see that SMOTE is significantly helping, which puts this technique in the same group with SVM and RF. We believe this is a natural (expected) behavior on the basis of the previous results. Although MLP belongs to a different type of machine learning algorithms than SVM or RF, SMOTE was designed to work well in a general case, so observing improvements after augmenting datasets with it comes as no surprise. The second type of behavior is observed with CNN. Here, SMOTE sometimes helps but, in other instances, actually decreases the performance of CNN significantly. First, we note that CNN does have problems with the imbalanced datasets, which can be observed here, but is also reported in [11,47]. Still, in our imbalanced datasets, we see somewhat less of such a behavior than in the related work. The reason for that comes from the fact that CNN is primarily intended to work with row measurements that usually have a large number of features. Having a large number of features allows one to take advantage of the deep network architecture and obtain a powerful classifier. Here, we use only the 50 most important features (to be comparable to previous cases), which forces our architecture to be shallow. Consequently, CNN is not able to train a high-performing model, which is then not maximizing its performance by setting all the measurements into the majority class. Although this sounds like a positive behavior and even something that should be desired in an effort to alleviate the consequences of the imbalanced datasets, such models also generalize to unseen data much less accurately, which results in a significantly lower classifier performance. Indeed, by comparing the results for SVM/RF and CNN, we can see that SVM/RF give better results, when considering both imbalanced and SMOTE datasets. Still, imbalanced CNN is better than imbalanced SVM/RF in some scenarios, e.g., the AES_HD dataset. The third type of behavior happens with SA, TA and TA pooled, where SMOTE is not beneficial, except in the case when the training set is very small (i.e., 1 000 measurements).

Finally, we ask a question how far are the results obtained with SMOTE if one compares it with a perfectly balanced dataset of the same size. To that end, we construct a perfectly balanced dataset where each class has 195 examples and compare it with SMOTE where the resulting classes consist of 195 measurements. We consider here relatively small datasets, which is a consequence of having only a small number of minority class representatives from which we can build the perfectly balanced dataset. The results show that for DPAcontest v4, SVM performs much better for the perfectly balanced dataset than for SMOTE dataset. At the same time, for instance, for GE there is no difference when considering RF. For AES_HD, the difference is again clearly visible, but less pronounced when compared to DPAcontest v4. This behavior is expected since perfectly balanced dataset must provide more information than the dataset that was balanced with artificial examples. The advantage of perfectly balanced dataset depends on the number of examples we have and on the level of noise, so it is difficult to stipulate exactly how much is the advantage of perfectly balanced datasets.



(a) Template attack (b) Template attack pooled

Fig. 4: Guessing entropy for imbalanced and SMOTE on AES_HD, template attack

7 Conclusions and Future Work

This paper explores the problem of highly imbalanced datasets and classification. SCA offers realistic scenarios, where we encounter datasets with large amounts of noise, with high imbalance (where some classes are on average 70 times more represented than other classes). Additionally, SCA 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 of techniques that can help in imbalanced data scenarios and we show that SMOTE is especially useful in a number of difficult (noisy) scenarios over a range of ML techniques. We observe a 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.

Further, we plan to investigate the last two questions from Section 5.2. Designing a new ML metric that reflects the SCA behavior better seems to be very difficult (or even impossible), but using SCA metrics in the ML process is possible. The main question is whether such an approach would offer reasonable computational complexity.

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. 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.
3. 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

4. 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.
5. 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
6. 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
7. 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.
8. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
9. 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
10. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99) (2017) 1–1
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. Bhasin, S., Guilley, S., Flamant, F., Selmane, N., Danger, J.L.: Countering Early Evaluation: An Approach Towards Robust Dual-Rail Precharge Logic. In: WESS, ACM (oct 2010) DOI: 10.1145/1873548.1873554.
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. Brier, É., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: CHES. Volume 3156 of LNCS., Springer (August 11–13 2004) 16–29 Cambridge, MA, USA.
25. Batina, L., Gierlichs, B., Prouff, E., Rivain, M., Standaert, F.X., Veyrat-Charvillon, N.: Mutual Information Analysis: a Comprehensive Study. J. Cryptology **24**(2) (2011) 269–291
26. Cooper, J., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test Vector Leakage Assessment (TVLA) Methodology in Practice (Sept 24–26 2013) International Cryptographic Module Conference (ICMC), Holiday Inn Gaithersburg, MD, USA.
27. Durvaux, F., Standaert, F.X.: From improved leakage detection to the detection of points of interests in leakage traces. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2016) 240–262
28. 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
29. 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
30. 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
31. 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
32. Breiman, L.: Random Forests. Machine Learning **45**(1) (2001) 5–32
33. Krawczyk, B.: Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence **5**(4) (Nov 2016) 221–232
34. Longadge, R., Dongre, S.: Class Imbalance Problem in Data Mining Review. CoRR **abs/1305.1707** (2013)
35. 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
36. 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
37. 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

38. 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
39. James, G., Witten, D., Hastie, T., Tibsihrani, R.: An Introduction to Statistical Learning. Springer Texts in Statistics. Springer New York Heidelbert Dordrecht London (2001)
40. 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
41. Choudary, O., Kuhn, M.G.: Efficient template attacks. In Francillon, A., Rohatgi, P., eds.: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Volume 8419 of LNCS., Springer (2013) 253–270
42. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, ed.: CHES. Volume 3659 of LNCS., Springer (Sept 2005) 30–46 Edinburgh, Scotland, UK.
43. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10) (1995)
44. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org.
45. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS-W. (2017)
46. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. (2016) 3–26
47. Picek, S., Samiotis, I.P., Heuser, A., Kim, J., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. Cryptology ePrint Archive, Report 2018/004 (2018) `https://eprint.iacr.org/2018/004`.

A Detailed Results for Different Classifiers and Balancing Techniques

For all considered machine learning techniques we first run a tuning phase and present results with the best obtained parameters. SVM – Support Vector Machines, RF – Random Forest, MLP – Multilayer Perceptron, CNN – Convolutional Neural Networks, TA – template attack [2], TA p. – pooled template attack [41], SA – stochastic attack [42].

Table 7: DPAcontest v4 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
MLP	1 000	$tanh$, (50, 10, 50)	56.9	52	57	53	6	15
MLP	10 000	$tanh$, (50, 10, 50)	54.1	53	54	51	4	12
MLP	50 000	$tanh$, (50, 30, 20, 50)	61.1	61	61	60	4	11
CNN	1 000	[46]	65.1	62	61	62	3	12
CNN	10 000	[46]	38.8	35	36	38	3	9
CNN	50 000	[46]	62.4	61	61	61	4	10
TA	1 000	–	11.2	–	–	–	–	–
TA	10 000	–	13.4	–	–	–	–	–
TA	50 000	–	82.4	–	–	–	3	6
TA p.	1 000	–	64.7	–	–	–	3	9
TA p.	10 000	–	81.0	–	–	–	3	7
TA p.	50 000	–	81.6	–	–	–	3	6
SA	1 000	–	70.4	–	–	–	8	45
SA	10 000	–	82.0	–	–	–	3	56
SA	50 000	–	82.2	–	–	–	3	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
			Random undersampling					
SVM	1 000	C=1, γ=1	34.6	34	35	28	10	30
SVM	10 000	C=1, γ=1	49	48	49	44	8	20
SVM	50 000	C=1, γ=1	63.1	62	63	61	4	12
RF	1 000	I=1 000	44.7	47	44	37	6	17
RF	10 000	I=1 000	62.2	66	62	61	4	13
RF	50 000	I=1 000	74.4	75	74	74	4	12
			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
MLP	1 000	$tanh$, (50, 30, 20, 50)	57.8	53	58	53	7	17
MLP	10 000	$relu$, (50, 30, 20, 50)	63.9	63	64	62	3	7
MLP	50 000	$tanh$, (50, 30, 20, 50)	70.3	71	70	70	3	8
CNN	1 000	[46]	53.0	49	51	51	5	13
CNN	10 000	[46]	61.0	59	60	59	4	12
CNN	50 000	[46]	67.3	66	66	66	5	14
TA	1 000	–	0.4	–	–	–	–	–
TA	10 000	–	0.3	–	–	–	–	–
TA	50 000	–	27.5	–	–	–	13 600	–
TA p.	1 000	–	10.9	–	–	–	3 600	8 900
TA p.	10 000	–	15.2	–	–	–	1 400	3 300
TA p.	50 000	–	10.9	–	–	–	2 200	5 300
SA	1 000	–	3.6	–	–	–	2 478	4 528
SA	10 000	–	21.9	–	–	–	1 055	2 484
SA	50 000	–	10.9	–	–	–	1 438	3 582
			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

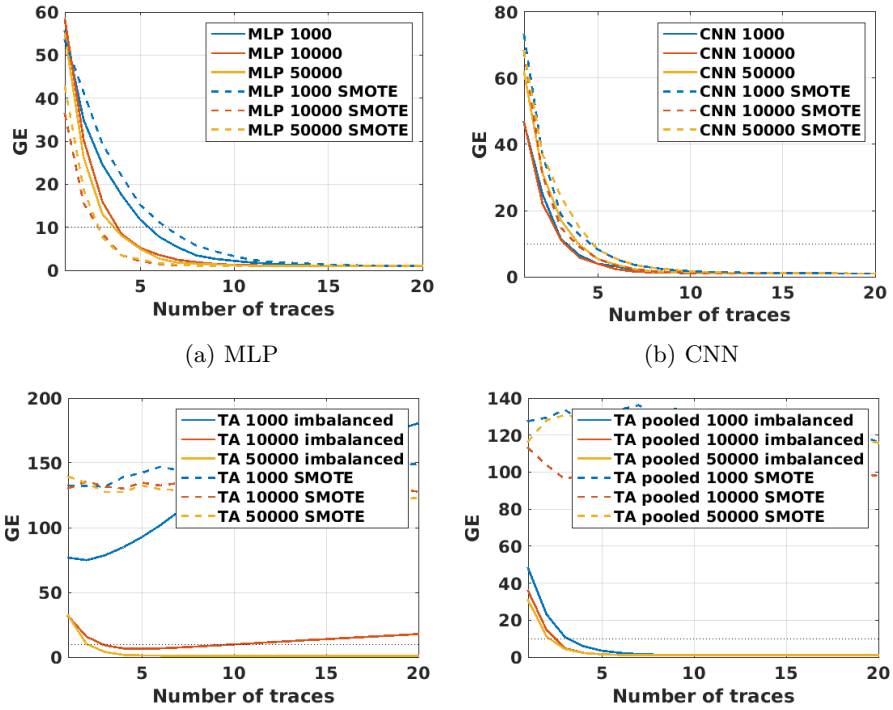
Table 8: AES_RD dataset. Values are given as percentages (ML metrics) and number of traces (for GE/SR) on test set.

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	–
MLP	1 000	$relu$, (50, 25, 10, 25, 50)	25.5	16	26	16	–	–
MLP	10 000	$tanh$, (50, 25, 10, 25, 50)	27.3	7	27	12	–	–
MLP	25 000	$tanh$, (50, 10, 50)	27.2	11	27	12	–	–
CNN	1 000	[46]	27.3	7	27	12	–	–
CNN	10 000	[46]	25.3	12	24	15	–	–
CNN	50 000	[46]	27.0	12	24	14	–	–
TA	1 000	–	2.2	–	–	–	–	–
TA	10 000	–	0.6	–	–	–	–	–
TA	25 000	–	17.6	–	–	–	–	–
TA p.	1 000	–	12.9	–	–	–	–	–
TA p.	10 000	–	6.8	–	–	–	13 500	–
TA p.	25 000	–	5.3	–	–	–	8 900	20 700
SA	1 000	–	5.3	–	–	–	–	–
SA	10 000	–	2.4	–	–	–	–	–
SA	25 000	–	2.0	–	–	–	–	–
			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	25 000	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	25 000	I=1 000	26.3	25	26	19	7 173	20 650
			Random undersampling					
SVM	1 000	C=1, γ=1	7.5	19	8	8	–	–
SVM	10 000	C=1, γ=1	14.7	20	15	17	–	–
SVM	25 000	C=1, γ=1	9.8	20	10	12	–	–
RF	1 000	I=200	13.9	19	14	14	–	
RF	10 000	I=200	14.2	20	14	16	–	
RF	25 000	I=1 000	11.3	20	11	14	10 500	22 400
			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	25 000	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	25 000	I=1 000	24.1	22	24	22	1 619	5 593
MLP	1 000	$tanh$, (50, 30, 20, 50)	14.9	20	15	16	–	–
MLP	10 000	$relu$, (50, 30, 20, 50)	25.1	19	25	17	–	–
MLP	25 000	$relu$, (50, 30, 20, 50)	26.9	21	27	13	–	–
CNN	1 000	[46]	16.8	16	17	17	–	–
CNN	10 000	[46]	18.9	17	17	17	–	–
CNN	25 000	[46]	19.4	16	17	17	–	–
TA	1 000	–	0.4	–	–	–	–	–
TA	10 000	–	25.0	–	–	–	–	–
TA	25 000	–	23.3	–	–	–	–	–
TA p.	1 000	–	0.4	–	–	–	23 300	–
TA p.	10 000	–	0.4	–	–	–	–	–
TA p.	25 000	–	0.4	–	–	–	–	–
SA	1 000	–	0.4	–	–	–	–	–
SA	10 000	–	0.4	–	–	–	–	–
SA	25 000	–	0.4	–	–	–	–	–
			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	25 000	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	25 000	I=1 000	8.9	19	9	5	20 400	–

Table 9: 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
MLP	1 000	$tanh$, (50, 30, 20, 50)	25.4	12.0	25	16	–	–
MLP	10 000	$tanh$, (50, 10, 50)	27.0	12	27	12	–	–
MLP	50 000	$tanh$, (50, 25, 10, 25, 50)	27.0	7	27	11	–	–
CNN	1 000	[46]	25.1	9	26	11	–	
CNN	10 000	[46]	27.0	7	27	11	12 500	22 900
CNN	50 000	[46]	27.0	7	27	12	12 600	23 000
TA	1 000	–	0.3	–	–	–	–	–
TA	10 000	–	6.6	–	–	–	22 900	–
TA	50 000	–	14.0	–	–	–	14 000	
TA p.	1 000	–	12.8	–	–	–	11 000	22 300
TA p.	10 000	–	7.4	–	–	–	1 200	3 100
TA p.	50 000	–	5.6	–	–	–	700	1 600
SA	1 000	–	6.7	–	–	–	–	–
SA	10 000	–	4.6	–	–	–	21 963	–
SA	50 000	–	3.9	–	–	–	23 094	–
			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	–
			Random undersampling					
SVM	1 000	C=1, γ=1	16.7	18	17	10	–	–
SVM	10 000	C=1, γ=1	7.8	19	8	9	15 000	–
SVM	50 000	C=1, γ=1	10.6	19	11	12	1 800	4 900
RF	1 000	I=500	14.2	17	14	9	–	–
RF	10 000	I=1 000	10	20	10	12	–	–
RF	50 000	I=1 000	11.9	20	12	14	9 000	20 100
			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
MLP	1 000	$tanh$, (50, 30, 20, 50)	11.6	19	12	11	–	–
MLP	10 000	$relu$, (50, 30, 20, 50)	15.3	21	15	16	7 400	16 200
MLP	50 000	$tanh$, (50, 30, 20, 50)	26.4	23	26	15	8 300	18 100
CNN	1 000	[46]	21.0	19	21	20	–	–
CNN	10 000	[46]	21.1	19	21	20	4 800	11 400
CNN	50 000	[46]	23.2	22	23	23	8 600	19 200
TA	1 000	–	0.4	–	–	–	–	–
TA	10 000	–	0.4	–	–	–	–	–
TA	50 000	–	0.4	–	–	–	–	–
TA p.	1 000	–	0.4	–	–	–	–	–
TA p.	10 000	–	0.4	–	–	–	–	–
TA p.	50 000	–	0.3	–	–	–	19 500	–
SA	1 000	–	0.38	–	–	–	–	–
SA	10 000	–	0.41	–	–	–	–	–
SA	50 000	–	0.38	–	–	–	–	–
			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	–	–

B Guessing Entropy for Deep Learning and TA



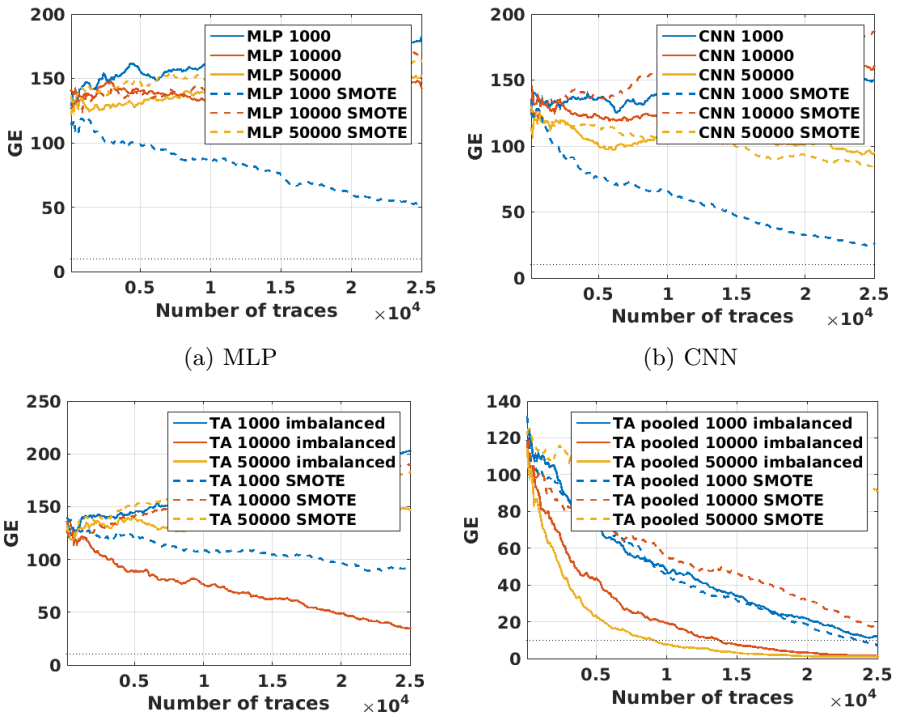
(a) MLP

(b) CNN

(c) Template attack

(d) Template attack pooled

Fig. 5: Guessing entropy for imbalanced and SMOTE on DPAcontest v4, deep learning and template attack (pooled)



(a) MLP

(b) CNN

(c) Template attack

(d) Template attack pooled

Fig. 6: Guessing entropy for imbalanced and SMOTE on AES_RD, deep learning and template attack