arXiv:2203.11410v2 [cs.CR] 2 May 2022

Dazzle: Using Optimized Generative Adversarial Networks to Address Security Data Class Imbalance Issue

Rui Shu, Tianpei Xia, Laurie Williams, Tim Menzies
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

Background: Machine learning techniques have been widely used and demonstrate promising performance in many software security tasks such as software vulnerability prediction. However, the class ratio within software vulnerability datasets is often highly imbalanced (since the percentage of observed vulnerability is usually very low). **Goal:** To help security practitioners address software security data class imbalanced issues and further help build better prediction models with resampled datasets. **Method:** We introduce an approach called ***Dazzle*** which is an optimized version of conditional Wasserstein Generative Adversarial Networks with gradient penalty (cWGAN-GP). Dazzle explores the architecture hyperparameters of cWGAN-GP with a novel optimizer called Bayesian Optimization. We use Dazzle to generate minority class samples to resample the original imbalanced training dataset. **Results:** We evaluate Dazzle with three software security datasets, i.e., Moodle vulnerable files, Ambari bug reports, and JavaScript function code. We show that Dazzle is practical to use and demonstrates promising improvement over existing state-of-the-art oversampling techniques such as SMOTE (e.g., with an average of about 60% improvement rate over SMOTE in recall among all datasets). **Conclusion:** Based on this study, we would suggest the use of optimized GANs as an alternative method for security vulnerability data class imbalanced issues.

KEYWORDS

Security Vulnerability Prediction, Class Imbalance, Hyperparameter Optimization, Generative Adversarial Networks.

ACM Reference Format:
Rui Shu, Tianpei Xia, Laurie Williams, Tim Menzies. 2022. Dazzle: Using Optimized Generative Adversarial Networks to Address Security Data Class Imbalance Issue. In *19th International Conference on Mining Software Repositories (MSR '22), May 23–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3524842.3528437

1 INTRODUCTION

Machine learning has been used for many security tasks; e.g. security vulnerability prediction [23]. A core problem with a security dataset is class imbalance; i.e., there may be very few instances of security events within many such datasets. For example, Figure 1 shows that components with known security vulnerabilities within Mozilla are very rare. As another example, as security bug reports can describe the critical security vulnerabilities in software products, Peters et al. [49] show that only 0.8% of bug reports are known to be security bug reports in their study.

When the target class is rare, it is challenging for a learner to distinguish the goal (security target) from other event [34]. There are many ways to handle the class imbalance. For example, SMOTE (i.e. **S**ynthetic **M**inority **O**versampling **TE**chnique) [13] is a highly-cited methods that *oversamples* the minority class by generating new samples. Specifically, SMOTE works by introducing new synthetic samples along with the line segments of k nearest minority class neighbors. However, SMOTE generates new samples via a simplistic linear interpolation between minority neighbors. Also, when generating new data in some local regions, SMOTE does not use knowledge from the whole minority class samples – which means its interpolations might not be helpful. Recently, SMOTE has been used extensively in software analytics in work published at top venues such as ICSE [1, 58], TSE [8], EMSE [33], etc.

SMOTE was first proposed in 2002, and this paper explores "can we do better than SMOTE?". For example, a new approach to generate samples for resampling purposes is GANs [25]; i.e. **G**enerative **A**dversarial **N**etworks. Unlike SMOTE's issue with local inference, GANs oversampling can effectively learn the whole data characteristics and generate samples close to the distribution of original input data. Considering that GANs can achieve some impressive results in producing meaningful, realistic samples in prior studies (e.g., in

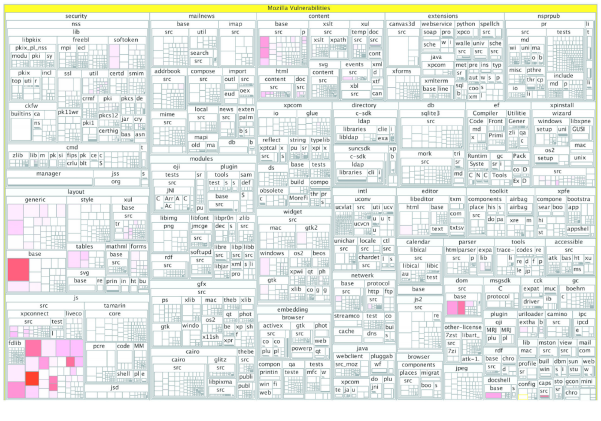

Figure 1: Mozilla code [47]. Only a few modules (seen in red) are vulnerable components.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MSR 2022, May 23–24, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9303-4/22/05…$15.00
https://doi.org/10.1145/3524842.3528437

Table 1: Recent works that use GANs as data oversampler in the security domain (as shown in column three, none of these prior works explored hyperparameter optimization on GANs). Training a GANs model is a difficult task since we have to achieve a balance between its internal components (i.e., the generator and discriminator) [52]. This paper explores this task with hyperparameter optimization with the novel Bayesian optimization.

Publication	Year	Optimized	Brief Description
[5]	2018	No	Propose text-GANs to generate phishing URLs.
[20]	2019	No	Train a GANs to output mimicked minority class example for credit card fraud detection.
[30]	2020	No	Propose IGAN-IDS to cope with imbalanced intrusion detection.
[64]	2020	No	Use an improved GANs to detect social bots on Twitter.
[14]	2021	No	Use GANs as data augmentation in Android Malware Detection.
[55]	2018	No	Apply GANs for black-box API attacks to deal with limited training data.
[15]	2019	No	Propose using CNN GANs to generate network traffic.
[53]	2020	No	Propose a GANs based intrusion detection system to counter imbalanced learning.
[66]	2019	No	Use GANs to synthesize DoS attack traces.
[51]	2019	No	GANs is used to generate flow-based network traffic.
[60]	2020	No	Adopt two existing GANs models to generate synthetic network traffic.

domains such as computer vision [31, 63]), more security practitioners have adopted variants of GANs in many security tasks [5, 15, 30] (see Table 1).

One reason to prefer SMOTE over GANs is that the SMOTE is much easier to implement and apply. GANs have two parts: a *generator* model that generates new plausible examples and a *discriminator* model that checks if it can distinguish real from fake examples. However, training a useful and *stable* GANs can be a difficult task [52]. Here, *stable* means a balance between generator and discriminator with proper coordination. For example, if one model overpowers the other, neither can learn more even with more iterations. Some other challenges with training GANs include *mode collapse* (discussed in §2.5), in which situation the generator may not explore much of the possible solution space and thus fails to produce a variety of realistic outputs.

This empirical study tries to tame the GANs training problem as well as using GANs as a data oversampler with *hyperparameter optimization* on Wasserstein GAN (WGAN) [6, 7]. WGAN applies the Wasserstein distance metric instead of the cross-entropy loss used in the traditional discriminator. The advantage of the Wasserstein distance metric is that it measures the distributions of each data feature and determines how far apart the distributions are for real and fake data. Considering the complexity of tuning two components in the GANs architecture, we use a novel optimizer called *Bayesian Optimization* [54, 57]. Our Bayesian optimizer explores the hyperparameter set of WGAN's generator and discriminator and returns an optimal solution set towards the evaluation target. We refer to our proposed combination of GANs and Bayesian Optimization as "***Dazzle***". The experiments of this paper evaluate Dazzle with three security datasets, i.e., Moodle vulnerable files, Ambari bug reports, and JavaScript function code dataset. The results show that we can achieve an average 60% improvement rate in recall across all datasets. We recommend using optimized GANs for security vulnerability dataset class rebalancing purposes based on this study.

As for the novelty and contribution of this work, we note that this paper is not the first work to apply Bayesian optimization to tune the GANs architecture. For example, prior work [17] has proposed using optimized GANs in the sign language classification. However, the main focus of this empirical study is to show that the idea of using optimized GANs is able to help solve some existing security tasks, and it is more promising than currently widely used SMOTE-based methods. We also note that this study cannot cover all security tasks as we show in Table 1, and we believe this would be an interesting future direction to explore.

The remainder of this paper is organized as follows. We discuss background and related work in Section 2 and our methodology in Section 3. We then report our experiment details in Section 4, including datasets, evaluation metrics, etc. Section 5 presents our experiment results. We discuss the threats of validity in Section 6 and provides a remark of addressing class imbalanced issues in Section 7 and then we conclude in Section 8.

2 BACKGROUND AND RELATED WORKS

2.1 Software Vulnerability Prediction

Software security vulnerabilities are critical issues that would impact software systems' confidentiality, integrity, and availability. The exploitation of such vulnerabilities would result in tremendous financial loss. To mitigate these issues, many machine learning and data mining techniques are proposed to build vulnerability prediction models to aid security practitioners [23].

Prior works have demonstrated several ways to extract useful features to train vulnerability prediction models. For example, as software bug reports can describe security vulnerabilities in software products, prior researchers [32, 49, 65] proposed a way to adopt natural language text based text mining techniques to identify security-related keywords. Vulnerability prediction models are then built by using the frequency of security-related keywords as features. Source code is another widely used avenue to derive vulnerability prediction models. Each piece of code can be represented by text, metric, token, tree, or graph. For example, in the metric-based representation, a code fragment is represented by a vector of features, such as lines of code, number of functions, total external calls, etc. These metrics can be extracted automatically with existing source code analyzers or extractors, which become ideal available resources to train prediction models [19, 41, 62]. Metric-based representation is often used at the file/component fragment level.

2.2 Software Vulnerability Dataset Class Imbalance

Software bug report based [49] or source code metric based [45] prediction models mostly require a large amount of prior knowledge of vulnerabilities, which means many known vulnerable bug reports or codes are needed to train supervised machine learning models effectively. However, the imbalance between non-security bug reports and security bug reports or non-vulnerable code and vulnerable code brings significant challenges. When training machine learning prediction models with those class imbalanced datasets, the resulting models usually demonstrate a heavy bias towards the majority class. They tend to classify new data into the majority class, but they belong to the minority class. Such a phenomenon makes prediction models difficult to detect rare vulnerabilities (which are important) since models cannot effectively learn the decision boundary, resulting in poor performance.

Many prior studies have introduced various ways to tackle this issues, such as utilizing the "sampling" idea with the imbalanced data and they mainly fall into the following categories:

- *Undersampling* to remove majority class instances;
- *Oversampling* to generate more of the minority class instances;
- Some *hybrid* of the first two methods.

How to choose an appropriate way to sample the datasets is based on the characteristics of the datasets. Machine learning researchers [27] advise that *undersampling* usually works better than *oversampling* if there are hundreds of minority instances in the datasets. When there are only a few dozen of minority instances, the *oversampling* approaches are superior to *undersampling*. In the case of large size training datasets, the *hybrid* methods would be preferred. The datasets we studied fall into the second category. Therefore, *oversampling* is a better choice.

2.3 SMOTE

A simple way to oversample data is to duplicate samples from the minority class in the training dataset before training a model. Samples from the training dataset are selected randomly with replacement. This method is called *RandomOverSampler*. It is referred to as a "naive" method because it assumes nothing about the data and provides no additional information to the model but barely balances the class distribution.

An improved way is synthesizing new samples with existing samples from the minority class. **S**ynthetic **M**inority **O**versampling **TE**chnique, also known as **SMOTE** [13], is an algorithm that oversamples the minority class by creating new synthetic samples. SMOTE works by selecting samples that are close in the feature space, drawing a line between the samples, and generating a new sample at a point along that line (as shown in Figure 2). Specifically, SMOTE calculates the k nearest neighbors for each minority class sample. Depending on the amount of oversampled instances required, one or more of the k-nearest neighbors are selected to create the synthetic samples. This amount is usually denoted by oversampling percentage (e.g., 50% by default). The next step is to create a synthetic sample connecting two minority samples randomly.

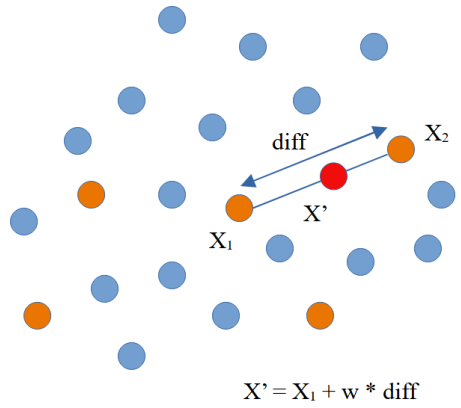

Figure 2: An example of how SMOTE works. The blue dots denote the majority class samples and the orange dots denote the minority class samples. In SMOTE, a neighbour sample X_2 is selected for sample X_1 and a new synthetic sample X' (i.e., the red dot) is created as a linear interpolation.

Algorithm 1: Pseudocode for SMOTE.

```
Function SMOTE (D_training, k, m, r);
  Input  : Training datasets - D_training,
           Number of nearest neighbours - k,
           Number of synthetic instances to create - m,
           Distance metric parameter - r
  Output : Resampled training datasets - D_resampled
while # of Minority samples < m do
    x ← random minority class samples from D_training
    neighbours ← k nearest neighbours of x
    for n_i ∈ neighbours do
        x_new ← interpolate(x, n_i)
        Add x_new to D_resampled
return D_resampled
```

Algorithm 1 describes how SMOTE works. A random sample from the minority class is firstly chosen. Then k of the nearest neighbors of that example are found. For each selected neighbor, a synthetic example is created at a randomly selected point between the two samples in feature space. The approach is more effective than the naive duplicate oversampling because new synthetic samples from the minority class are created that are plausible and relatively close in feature space to existing samples from the minority class.

Table 2 also lists several variants of SMOTE, which are used as our baseline methods for comparison purposes. For example, *ADASYN* [29] (i.e., Adaptive Synthetic Sampling) is an improved version of SMOTE, which creates synthetic data according to the data density. The synthetic data generation would be inversely proportional to the density of the minority class. It means more synthetic data are created in regions of the feature space where the density of minority examples is low and fewer or none where the density is high. *BorderlineSMOTE* [28] involves selecting those

Table 2: A list of baseline data oversampling methods used in this study.

Method	Description
RandomOverSampler	Randomly duplicate examples in the minority class.
SMOTE	Create a synthetic sample between minority sample and its neighbour.
ADASYN	Creates synthetic data according to the data density.
BorderlineSMOTE	Only select minority samples that are misclassified.
KMeansSMOTE	Apply a KMeans clustering before to over-sample using SMOTE.
SVMSMOTE	Use an SVM algorithm to detect sample to use for generating new synthetic samples.
SMOTUNED	An auto tuning version of SMOTE that optimizes its parameters.

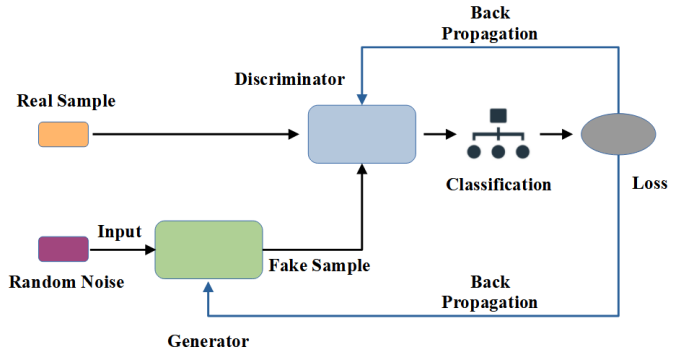

Figure 3: The architecture of a traditional GANs model.

Algorithm 2: Pseudocode for a simple GANs as an over-sampler.

```
Function simpleGAN (D_training, D, G);
  Input  : Training datasets - D_training,
           Discrinimator - D,
           Generator - G
  Output : Resampled training datasets - D_resampled
for epoch_i ∈ number of epochs do
    /* Train Discriminator D                                   */
    Sample a mini-batch of real data, train as true
    sample a mini-batch of fake data from Generator G, train as false
    Update the gradient of Discriminator D
    /* Train Generator G                                       */
    Sample a mini-batch of fake data from Generator G, which should be
      classified as true
    Update the gradient of Generator G
Generate new data X_new with Generator G
Add X_new to D_resampled
return D_resampled
```

instances of the minority class that are misclassified. Unlike with the SMOTE, where the synthetic data are created randomly between the two data, BorderlineSMOTE only makes synthetic data along the decision boundary between the two classes. *KMeansSMOTE* [38] applies a KMeans clustering before to over-sample using SMOTE, and *SVMSMOTE* [48] uses an SVM algorithm to detect samples to use for generating new synthetic samples. *SMOTETUNED* [1] is an auto-tuning version of SMOTE that explores the parameter space of SMOTE with an optimizer called differential evolution algorithm.

2.4 GANs

Compared with SMOTE, GANs is a new emerging technique, and in this work, we explore the merits of GANs over SMOTE. **G**enerative **A**dversarial **N**etworks (**GANs**) [25] are a neural network architecture that has a set of two models used to produce synthetic data. The GANs model architecture (see Figure 3) typically involves two sub-models, i.e., a *generator* and a *discriminator*. The generator model generates new plausible examples in the problem domain, while the discriminator model distinguishes whether the new generated examples by generator are real or fake, from the perspective of the domain. Both of the models are trained in a *min-max zero-sum game* since the generator tries to produce synthetic instances of data that reliably trick the discriminator, while the discriminator tries to distinguish between real and fake data.

The two models, the generator and discriminator, are trained together. The generator generates a batch of samples, and these, along with real examples from the domain, are provided to the discriminator and classified as real or fake. The discriminator is then updated to get better at discriminating real and fake samples in the next round, and importantly, the generator is updated based on how well or not, the generated samples fooled the discriminator. When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases. Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights. In fact, a really good generative model may be able to generate new examples that are not just plausible, but indistinguishable from real examples from the problem domain.

The loss function of GANs is shown as follows:

$$\min_G \max_D V(D,G) = E_x[log(D(x))] + E_z[log(1 - D(G(z))] \quad (1)$$

where $D(x)$ is the discriminator's estimate of the probability that real data instance x is real, E_x is the expected value over all real data instance. $G(x)$ is the generator's output when given noise z and $D(G(z)$ is the discriminator's estimate of the probability that a fake instance is real. E_z is the expected value over all random inputs to the generator. The goal of discriminator is to bring $D(G(Z))$ closer to 0, while the goal of generator is to bring it closer to 1. If the generator outputs a probability of 0.5, then this means the discriminator is unable to make a right decision whether the instance is real or fake.

GANs are rapidly evolving fields, delivering promising results in generating realistic examples across a range of problem domains, most notably in images tasks such as synthesizing images from text description [67], image compression [2], image classification [69], etc. In the security domain, prior work indicate that GANs would be an ideal technique to train a classification model to explore unforeseen data threats with generated data. Table 1 lists recent work that use GANs as data oversampler (used in a way similar to Algorithm 2). Those works motivate our study, however, we also note that they hardly introduce any way to optimize their GANs architecture as we do in this study.

2.5 Challenges with traditional GANs

Although GANs has achieved notable success in multiple domains, GANs also face several challenges which may cause issues such as *unstable training* [31, 52].

Nash Equilibrium. Nash Equilibrium (NE) [46] is a notion in game theory where two players come to a joint strategy in which each player select a best response (i.e., a strategy that yields the best payoff against the strategies chosen by the other player). In the context of GANs, the generator and discriminator represent the two players, which work in an adversarial way against each other. The generator and discriminator train themselves simultaneously for NE. When both generator and discriminator update their cost function independently without coordination, it is hard to achieve NE.

Vanishing Gradient. Vanishing gradient occurs when one part of GANs is more powerful than the other part. For example, if the generator model is very poor, then the discriminator can easily distinguish between real and fake samples. This further causes the probability of the generated samples being real from generator close to zero, i.e., gradients of $log(1 - D(G_z))$ will be very small. Therefore, discriminator fails to provide gradients and the generator will stop updating.

Mode Collapse. Model collapse is one of the most crucial issues with GANs training, which means the output samples from generator lacks of variety (i.e., producing same outputs). If the generator starts to produce the same output, an ideal strategy for the discriminator is to reject the output. However, if the discriminator gets stuck in local minima and does not find the strategy, then the generator tends to find the same output that seems most plausible to the discriminator.

2.6 Attempts to Address the Challenges

Prior work indicates that Wasserstein GAN (WGAN) [6, 7] is designed to prevent vanishing gradients. In WGAN, the discriminator does not classify input instances, but it outputs an exact score for each instance. WGAN does not use a threshold to decide whether an instance is real or fake but tries to make the score bigger than fake instances. WGAN also alleviates mode collapse since it prevents the discriminator from getting stuck in local minima. In this case, the generator has to try new samples since the discriminator would reject the same sample. For the Nash equilibrium problem (i.e., non-converge), prior work [18] suggests an exhaustive hyperparameter and architecture search, and hence this work. We will discuss WGAN and architecture optimization in detail in the next subsections.

2.7 cWGAN-GP

Traditional GANs is motivated to minimize the distance between the actual and predicted probability distributions for real and generated samples. Typically, there are two metrics to measure the similarity between two probability distributions, the *Kullback-Leibler divergence* and the the *Jensen-Shannon divergence*.

Kullback-Leibler divergence [37], also known as *KL divergence*, is a metric to measure *relative entropy* between two probability distributions over the same variable. Consider distributions P and Q of a continuous random variable, the KL divergence is computed as an integral as follows:

$$KL(P\|Q) = \int p(x) log(\frac{p(x)}{q(x)})\, dx \tag{2}$$

where $p(x)$ and $q(x)$ are the probability density functions of distribution P and Q, respectively. The lower the KL divergence value, the closer the two distributions are to each other.

An extension to KL divergence is the *Jensen-Shannon divergence* [21], also known as *JS divergence*. Compared with KL divergence, this metric is a symmetric version, which means calculating the divergence for distribution P and Q will result in the same score as from distribution Q and P. Define the quantity $M = (P + Q) * 0.5$, JS divergence is formulated as follows:

$$JS(P\|Q) = \frac{1}{2}KL(P\|M) + \frac{1}{2}KL(Q\|M) \tag{3}$$

Besides symmetric, JS divergence is also a smoothed and normalized version, and the square root of this score which referred as *Jensen-Shannon distance* is commonly used.

The JS divergence scores provides ways to calculate scores for cross-entropy which is commonly used as a loss function in classification models such as the discriminator in GANs. However, researchers notice that such loss function does not necessarily correlate with the sample quality and therefore does not guarantee the convergence between generator and discriminator to an equilibrium [24]. Wasserstein GAN [6, 7] improves traditional GANs' optimization goal based on Wasserstein distance, which is formulated as follows:

$$W(P, Q) = \inf_{\gamma \sim \prod(P,Q)} E_{(x,y)\sim\gamma}[\|x - y\|] \tag{4}$$

where $\prod(P, Q)$ is the set of all possible joint distributions in which P and Q are combined. For each possible joint distribution γ, a real sample x and a generated sample y can be sampled, and the sample distance $\|x - y\|$ is calculated, so that the expected value $E_{(x,y)\sim\gamma}[\|x - y\|]$ of the sample to the distance under the joint distribution γ can be calculated. This expected value can be taken to the lower bound in all possible joint distributions and defined as the Wasserstein distance of the two distributions. This distance is helpful when facing two distributions with non-overlapping, in which case JS divergence fails to provide a useful gradient.

WGAN with Gradient Penalty (WGAN-GP) further suggests to add a gradient penalty to address the concern of Lipschitz constraint. With the gradient penalty, the norm of the gradient is limited to a

value of 1 to satisfy the 1-Lipschitz continuous condition. This is helpful to build a "worse" discriminator, but provide more gradient information that helps to train a better generator. In short, the use of gradient penalty helps enhance the training stability and reduce the mode collapse of the networks.

Moreover, we adopt an extension to WGAN-GP, which is called conditional WGAN-GP. In this method, both the generator and discriminator add data category information, with which the optimization function of WGAN-GP is a maximal and minimal game with this condition.

2.8 Bayesian Optimization

Since training a new GANs model can be difficulty, this works checks if that process can be automated with hyperparameter optimization. Typically a hyperparameter has a known effect on a model in the general sense, but it is not clear how to best set a hyperparameter for a given dataset. Hyperparameter optimization or hyperparameter tuning is a technique that explores a range of hyperparameters and search for the optimal solution for a task. *Bayesian optimization* [54, 57] is a widely-used hyperparameter optimization technique that keeps track of past evaluation results. The principle of Bayesian optimization is using those results to build a probability model of objective function, and map hyperparameters to a probability of a score on the objective function, and therefore use it to select the most promising hyperparameters to evaluate in the true objective function. This method is also called *Sequential Model-Based Optimization* (SMBO).

The probability representation of the objective function is called *surrogate function* or *response surface* because it is a high-dimensional mapping of hyperparameters to the probability of a score on the objective function. The surrogate function is much easier to optimize than the objective function and Bayesian methods work by finding the next set of hyperparameters to evaluate the actual objective function by selecting hyperparameters that perform best on the surrogate function. This method continually updates the surrogate probability model after each evaluation of the objective function.

Several prior works have combined Bayesian optimization with GANs in tasks from other domains [16] [17], and this work shares the similar underlying idea with previous studies and adopts this combination in selected security tasks.

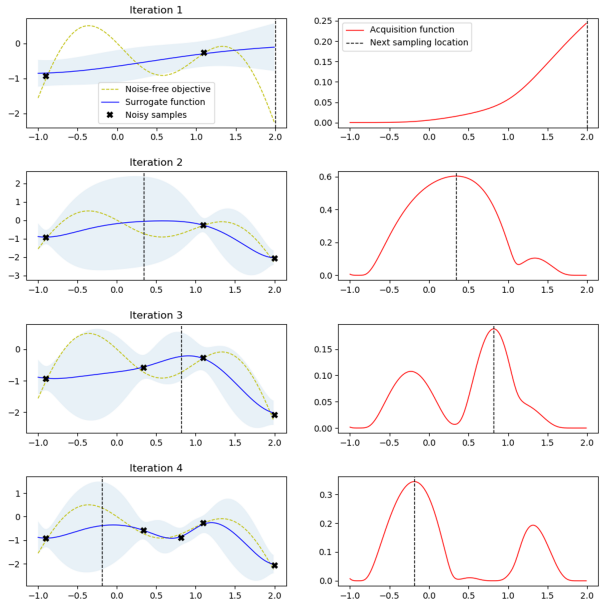

Figure 4: An example of the Bayesian optimization process. Bayesian optimization incorporates prior belief about *objective function* and updates the prior with samples drawn from objective function to get a posterior that better approximates objective function. The model used for approximating the objective function is called *surrogate function*. Bayesian optimization also uses an *acquisition function* that directs sampling to areas where an improvement over the current best observation is more likely. Note that the Bayesian optimization we use come from the HyperOpt [9] library, in which the optimization algorightm is based on Tree of Parzen Estimators (TPE).

3 METHODOLOGY

3.1 Dazzle: Optimized cWGAN-GP

In designing the network architecture of cWGAN-GP, another concern emerges as how to select the hyperparameters of the structure. GANs models might be highly sensitive to the hyperparameter selection. Prior work on DCGANs [50] introduced a deep convolutional generative adversarial networks that made several modifications to the model hyperparameters of CNN architecture to address the architectural topology constraints and made the GANs' training more stable. For example, that work

(1) replaced pooling layers with strided convolutions and fractional-strided convolutions;
(2) used batch normalization for generator & discriminator;
(3) used ReLU activation in generator;
(4) used LeakyReLU activation in discriminator.

Inspired by this work, we hypothesize that *GANs would benefit from an automatic optimized architecture*. We mean proper hyperparameter selection would help with GANs training to converge and further achieve better performance. In our case as using GANs as data oversampler, if we indicate a not well-designed GANs as GANs *A* and a well-designed GANs as GANs *B*, then if we build classification model with training data from GANs *B*, then the prediction performance is better than the models built from data with GANs *A*.

However, hyperparameter optimization is not a trivial work, especially when facing a complex system such as neural networks. The tuning process is more challenging since there are more hyperparameter with neural networks, and what's the most important, even one iteration of evaluation would be time consuming. Traditional hyperparameter optimization techniques such as "random search" or "grid search" either suffer from not ideal performance or would be costly expensive. To address this concern, we propose a method called ***Dazzle*** that adopts a novel optimizer called Bayesian optimization that fine-tunes both generator model and discriminator model.

Table 3: Hyperparameter selection ranges chosen to optimize in Dazzle.

Hyperparameter	Range
Batch Size	16, 32, 64, 128
Learning Rate for Generator	0.0005, 0.001, 0.005, 0.01, 0.05, 0.1
Learning Rate for Discriminator	0.0005, 0.001, 0.005, 0.01, 0.05, 0.1
Optimizer for Generator	Adadelta, Adagrad, Adam, Adamax, NAdam, RMSprop, SGD
Optimizer for Discriminator	Adadelta, Adagrad, Adam, Adamax, NAdam, RMSprop, SGD
Activation Function for Generator	elu, relu, selu, sigmoid, softmax, tanh, hard_sigmoid, softplus, leakyRelu
Activation Function for Discriminator	elu, relu, selu, sigmoid, softmax, tanh, hard_sigmoid, softplus, leakyRelu
No. of Epochs	quniform(5, 20, 1)
Generator Layer Normalization	True, False
Discriminator Layer Normalization	True, False

* Note: **quniform(*low, high, q*)** is a function returns a value like $round(uniform(low, high)/q) * q$, while **uniform(*low, high*)** returns a value uniformly between low and $high$. We also note that we do not tune the number of layers but with a fixed number (e.g., 4) in our study, and we find that such architecture suffices to achieve good performance on considered datasets.

Algorithm 3: Pseudocode of Dazzle's training process.

```
Function Dazzle (D_training, D_validation, D, G, θ, F);
  Input  : Training datasets - D_training,
           Validation datasets - D_validation,
           Discrinimator - D,
           Generator - G,
           Hyperparameter space - θ,
           Target function - F
  Output : Optimal resampled training dataset D_resampled_optimal,
           Optimal hyperparameter set θ_optimal
for iteration_i ∈ number of Bayesian Optimization iterations do
    Select a hyperparameter set θ_i ∈ θ
    Train D and G with θ_i
    Generate new resampled training dataset D_resampled_i
    Build classifier with D_resampled_i and evaluate with D_validation
    Compute loss with target function F
Rank all optimization iterations by loss with smallest on the top
return D_resampled_optimal and θ_optimal
```

Dazzle's training process is on the training dataset and validation dataset. During each iteration of Bayesian optimization, Dazzle selects a hyperparameters for discriminator and generator from Table 3, and generates new minority samples. These samples are used to resample the original dataset (and to build the classifier). Each time, the classifier is only evaluated with validation dataset. With the optimization goal, the loss is computed. Finally, we rank all the optimization iterations by loss with smallest on the top of the rank, and select the trained classifer from that iteration as the optimized classifier. Moreover, we choose 30 iterations for Bayesian optimization and repeat the whole experiment process 10 times.

Algorithm 3 lists the optimization steps of Dazzle. Note that our task with the security datasets is a binary classification problem. In Dazzle, we choose g-measure as our optimization goal (i.e., the target to increase). G-measure is the harmonic mean of recall and the complement of false positive rate. We choose g-measure based on the following considerations. For an imbalanced dataset where there is a skew in class distribution, we have two competing goals:

- We focus on minimizing false negatives, i.e., increase recall;
- We prefer not to predict too many non-security samples as security samples, i.e., reduce false positive rate.

Therefore, g-measure is ideal for chasing both goals.

Table 4: Statistics of security datasets used in this study. Note that the security target column indicate the number of vulnerable files, security bug reports, and JavaScript function code, respectively.

Dataset	Security Target	Total	Imbalance Rate (%)	No. of Features
Moodle Vulnerable Files	24	2,942	0.8	13
Ambari Bug Reports	29	1,000	2.9	101
JavaScript Function Code	1,496	12,125	12.3	36

4 EXPERIMENTAL EVALUATION

4.1 Datasets

Our evaluations are experimented on datasets that are widely studied in prior work. **Moodle** [62] is an open source learning management system, and the data source for Moodle vulnerabilities is the National Vulnerabilities Database (NVD), from with a variety of vulnerabilities are covered, such as code injection, path disclosure, XSS, etc. A total of 24 vulnerable files are included in this dataset. **Ambari** [49] is an open source project of Apache that aims to provision, manage and monitor Apache Hadoop cluster. Bug reports with BUG or IMPROVEMENT label from the JIRA bug tracking system are selected, and then the selected bug reports are further classified with scripts or manually into six high impact bugs (i.e., Surprise, Dormant, Blocking, Security, Performance, and Breakage bugs). All the target bug reports in the Ambari dataset all belong to Security bug reports (i.e., bug reports of the type Security). The **JavaScript** [19] function code dataset extracts data from Node Security Platform and the Snyk Vulnerability Database, and used static source code metrics as predictor features. Table 4 shows a list and description of the datasets used in this study. As we can observe from the table, all datasets suffer from different levels of class imbalanced issues.

4.2 Machine Learning Algorithms

We apply five machine learning algorithms, namely K-Nearest Neighbours (KNN), Logistic Regression (LR), Decision Tree (DT),

Random Forest (RF) and Support Vector Machine (SVM) in our experiment. We choose them since they are widely used in previous literatures in different classification tasks in security [61] or other domains such as defect prediction [40]. We implement these algorithms with open source tool called Scikit-learn. In order to reduce the influence of model hyperparameters to our evaluation results, we adopt default settings from Scikit-learn. We do not claim that the list of algorithms that we use is complete, but we note that these algorithms are enough for our study purpose.

4.3 Evaluation Metrics

For the performance of the classification models, the confusion matrix is used, where TP, TN, FP and FN indicate true positive, true negative, false positive and false negative, respectively. We report the results of recall (*pd*), false positive rate (*pf*), f-measure and g-measure as we defined in Table 5. Note that precision and accuracy in the table are not endorsed in our study, since both of these metrics can be inaccurate for datasets where the positive class is rare case. For example, Menzies et al. [44] argue that when the target class is less than 10%, the precision results become more a function of the random number generator used to divide data (for testing purposes). G-measure is a composite metric, which is the harmonic mean of recall and the complement of false positive rate. A higher g-measure indicates higher recall and lower false positive rate. As we discuss before, this metric is also our optimization target in Dazzle. We also report f-measure for completeness purpose.

Table 5: Performance evaluation metrics. Definitions of recall (pd), false positive rate (pf), precision (prec), accuracy (acc), f-measure (f1) and g-measure (g-score).

Metric	Expression
Recall (pd)	$\frac{TP}{TP+FN}$
False Positive Rate (pf)	$\frac{FP}{FP+TN}$
Precision (prec)	$\frac{TP}{TP+FP}$
Accuracy (acc)	$\frac{TP+TN}{TP+TN+FP+FN}$
F-Measure (f1)	$\frac{2*pd*prec}{pd+prec}$
G-Measure (g-score)	$\frac{2*pd*(100-pf)}{pd+(100-pf)}$

4.4 Experiment Rigs

Our datasets are split in a stratified way into two parts with a ratio of 8:2 where the latter part is used as testing set. We further split the former part into training set and validation set with the same ratio. Therefore, the final ratio between the actual training, validation, and testing part is 6.4: 1.6: 2 of the whole dataset. The training part is only used for training classifiers with selected hyperparameter set, and the validation part is used to evaluate the classifiers during optimization iterations towards optimization goal. Then the selected optimized models are evaluated on the testing dataset.

Lastly, our implementation of Bayesian Optimization is based on the tool called Hyperopt [10], which is one of the most cited hyperparameter optimizer in the literature at this time of writing. The implementation of SMOTE and its variants are based on the open-source imbalanced-learn toolbox [39] while SMOTUNED is implemented according to Agrawal et al.'s study [1]. SMOTUNED has three available parameters:

- Number of neighbours k with range $[1, 20]$.
- Minkowski distance metric r with range $[1, 6]$.
- Number of synthetic samples m to create with range $[50, 500]$.

5 RESULTS

Our study answers the following research questions:

RQ1. Will GANs based oversampling better than SMOTE based oversampling?

For each treatment in our study, we use default learners for fair comparison. Table 6 lists all the evaluations results of metrics defined in Table 5 for all three datasets. In these results, the *None* treatment indicate the training process with original dataset without any oversampling techniques, after which different oversamplers such as *RandomOversampler* and variants of SMOTE are presented. The *cWGAN-GP* treatment is GANs based oversampler with optimization. In order to configure the architecture of cWGAN-GP, we randomly select parameter set from Table 3 during each run.

As we can observe from the table, the original dataset without oversampling performs badly across all datasets, even with different machine learning algorithms. The results are no surprise as we consider the percentage of security relevant class samples in Table 4. For moodle and Ambari dataset, the positive class samples are less than 3% of the whole datasets, it is hard for machine learning algorithms to learn the traits with so few samples. As a result, none of the learners can detect any true positive during the testing phase.

Naive oversampler such as the RandomOversampler shows some advantages for some learners, for example Logistic Regression and SVM, but fails for others. SMOTE and its variants demonstrate better results than RandomOversampler, but the advantage is not obvious. Previous state-of-the-art SMOTUNED works best among all SMOTE based oversampling techniques.

cWGAN-GP is the GANs version oversampler, and there are two observations from the results:

- cWGAN-GP achieves nearly tied performance with SMOTUNED in important metric such as recall, but we note that the latter is an optimized version which requires more effort and configuration.
- Unlike other oversamplers, cWGAN-GP does not fail totally in some certain machine learning algorithms. For example, the SVMSMOTE does not detect any true positive with LR and RF in Moodle dataset, and even 4 out of 5 learners fails in Ambari dataset. This phenomenon indicates that cWGAN-GP is more practical to use in general cases.

We have to point out that the false positive rate metric is not suggested to indicate which method is better than others. For example, in Table 6, for the Random Forest results of Moodle vulnerable files dataset, several treatments have achieve zero false positive rate (therefore highlighted in blue color), however, their recall results are also zero. Thus, these treatments are not recommended.

Since Dazzle optimizes GANs with the goal of increasing g-measure, which is the harmonic mean of recall and the complement

Table 6: Median performance results (converted to range 0 - 100) from 10 repeats. Best performances are highlighted.

Metric	Treatment	Moodle Vulnerable Files					Ambari Bug Report					JavaScript Function Code				
		KNN	LR	DT	RF	SVM	KNN	LR	DT	RF	SVM	KNN	LR	DT	RF	SVM
Recall	None	0	0	0	0	0	0	0	14	0	0	63	0	68	68	11
	RandomOversampler	0	100	0	0	100	0	57	57	0	42	72	65	76	73	22
	SMOTE	40	100	20	0	100	0	100	42	42	0	76	65	78	77	22
	ADASYN	60	100	0	0	100	0	100	57	42	0	78	49	76	77	29
	BorderlineSMOTE	40	60	0	0	60	0	100	0	14	0	75	49	76	76	30
	SVMSMOTE	40	60	0	0	40	0	0	0	28	0	74	58	79	76	23
	SMOTUNED	60	100	60	60	100	0	57	57	28	57	81	100	80	83	100
	cWGAN-GP	80	60	60	80	80	28	57	57	43	57	79	79	83	78	49
	Dazzle	100	80	100	100	80	85	71	71	57	71	86	84	83	83	78
False Positive Rate	None	0	0	0	0	0	0	0	2	0	0	2	0	4	1	1
	RandomOversampler	2	39	1	0	22	0	3	3	0	7	6	43	6	3	2
	SMOTE	17	40	3	0	24	0	96	4	1	0	9	42	6	4	3
	ADASYN	17	41	0	0	25	0	96	4	2	0	12	33	7	6	9
	BorderlineSMOTE	6	19	0	0	8	0	98	2	0	0	9	35	7	5	12
	SVMSMOTE	5	11	0	0	7	0	0	1	1	0	8	42	6	4	3
	SMOTUNED	19	61	14	20	17	0	31	3	1	14	44	100	35	38	98
	cWGAN-GP	17	12	24	20	21	2	1	3	1	2	17	6	5	17	36
	Dazzle	16	20	22	19	24	2	2	2	2	1	7	11	5	7	12
G-Measure	None	0	0	0	0	0	0	0	24	0	0	77	0	80	80	19
	RandomOversampler	0	75	0	0	87	0	71	71	0	58	81	61	84	83	36
	SMOTE	53	74	33	0	86	0	6	59	59	0	83	61	85	85	36
	ADASYN	69	74	0	0	85	0	7	71	59	0	82	56	84	85	43
	BorderlineSMOTE	56	68	0	0	72	0	3	0	24	0	82	55	83	84	45
	SVMSMOTE	56	71	0	0	55	0	0	0	44	0	82	57	85	84	37
	SMOTUNED	68	55	70	68	90	0	62	71	44	68	69	0	75	73	3
	cWGAN-GP	81	71	67	80	79	44	72	72	59	72	81	86	88	80	56
	Dazzle	91	79	87	89	79	91	83	82	72	83	89	86	88	87	83
F-Measure	None	0	0	0	0	0	0	0	9	0	0	71	0	68	77	18
	RandomOversampler	0	4	0	0	7	0	26	29	0	12	67	27	69	74	31
	SMOTE	3	4	8	0	6	0	2	18	35	0	62	27	69	74	31
	ADASYN	5	4	0	0	6	0	2	23	26	0	58	25	67	70	29
	BorderlineSMOTE	8	4	0	0	10	0	2	0	22	0	61	24	67	71	27
	SVMSMOTE	9	8	0	0	7	0	0	0	26	0	64	25	70	73	32
	SMOTUNED	5	2	6	4	8	0	4	30	22	85	36	21	40	39	22
	cWGAN-GP	7	9	4	6	6	19	50	32	35	40	53	70	76	51	24
	Dazzle	10	6	7	8	5	52	50	42	42	55	72	64	76	70	58

of false positive rate. Therefore, when the g-measure is increased, there would be three cases: 1) recall increased, FPR decreased; 2) recall increased, FPR increased; and 3) recall decreased, FPR decreased. Our results would fall into these three groups. When we notice that some improvements in recall come at the cost of increments in false positive rate, while the ideal false positive rate is zero. We say that the trade-off between the increments of recall and false positive rate is still acceptable, especially in mission-critical security tasks, as we do not want to miss any security relevant target samples in the detection. Such a "price" indicates more extra effort to read more source code or bug reports for security practitioners, and it is the price of software quality assurance.

Answer

cWGAN-GP is more practical to use in the general cases, as it is not sensitive to certain machine learning algorithms.

RQ2. Will Dazzle (optimized GAN) work even better?

We optimize Dazzle with the goal of g-measure, which ideally with high recall and lower false positive rate. As we can observe from the result table, Dazzle works even better than cWGAN-GP. Benefit from optimizing, Dazzle achieves an average of improvement rate of 30%, 62% and 17% over cWGAN-GP in recall, respectively. This is explainable, as the "default" (with randomly selection in our case) hyperparameters for GANs might bring in issues in Section 3, hence is not one-size-fits-all across all scenarios and should be deprecated. We would recommend exploring and developing specialized tools for certain local domain.

Answer

Dazzle (the optimized version) shows even better performance than cWGAN-GP across all studied datasets.

RQ3. Is optimized GANs (i.e., Dazzle) impractically slow?

As shown in **RQ1**, Dazzle achieves promising improvement over baseline treatments in performance with 30 iterations of optimization trails. Table 7 lists the average runtime of each treatment of all machine learning algorithms. Considering the complexity of optimizing the architecture of neural networks, Dazzle is not surprisingly takes the most runtime cost. However, considering the mission-critical nature of the security tasks we are addressing, we would comment that the trade-off between performance and runtime is still worth. The experiment is carried out with CPU resources only, and with the help of GPU or parallel computing, Dazzle could be configured to be more practical to use.

Answer

Even with more runtime, Dazzle still worths the trade-off when considering the improvement in performance.

Lastly, we believe that there are several directions that can be explored after this work. For example, we would like to try more security tasks to check and endorse the merits of the proposed methods in other cases. Secondly, we would plan to compare with other baselines such as recent improvements on SMOTE and methods other than oversampling. Thirdly, we would like to perform more analysis of the new samples generated by the proposed method to get a better understanding of the methods.

6 THREATS TO VALIDITY

6.1 Evaluation Bias

In our work, we choose some commonly used metrics for evaluation purpose and set *g-measure* as our optimization target. We do not use some other metrics because relevant information is not available to us or we think they are not suitable enough to this specific task (e.g., precision). In addition, we use equal weight in recall and specificity in the definition of g-measure, which is widely adopted in existing literature. We agree that it is important for these two elements to be re-weighted for different tasks, and this can be further explored as one of our future directions. Our implementation is flexible and we can adjust to proper metrics or balances with minor code modification.

6.2 Parameter Bias

We have to note that default hyperparameter values have been used for the baseline machine learning algorithms, which means that the performance results reported in Table 6 might be suboptimal for baseline methods. To some degree, this also might have the effect of magnifying the advantages of the proposed method. Previous studies have also indicated that it is a good practice to avoid using default settings of machine learning algorithms [36] [56] [59]. In the case if those hyperparameters have been tuned, the conclusions from the proposed method might be different.

6.3 Learner Bias

Research into automatic classifiers is a large and active field. While different machine learning algorithms have been developed to solve different classification problem tasks. Any data mining study, such as this paper, can only use a small subset of the known classification algorithms. For this work, we select machine learning algorithms that are commonly used in classification tasks.

Table 7: Runtime of oversampling treatments in minutes for each dataset. Note that "<" means the runtime is close but less than the given results.

Treatment	Moodle Vulnerable Files	Ambari Bug Report	JavaScript Vulnerability
RandomOversampler	< 1	< 1	< 1
SMOTE	< 1	< 1	< 1
ADASYN	< 1	< 1	< 1
BorderlineSMOTE	< 1	< 1	< 1
SVMSMOTE	< 1	< 1	< 1
SMOTUNED	< 3	< 2	< 5
cWGAN-GP	< 5	< 5	< 5
Dazzle	< 25	< 25	< 30

6.4 Input Bias

Our results come from the space of hyperparameter optimization explored in this paper. In theory, other ranges might lead to other results. That said, our goal here is not to offer the *best* optimization but to argue that optimized GANs architecture is better than current state-of-the-art oversampler in addressing class imbalance. For those purposes, we would argue that our current results suffice.

6.5 Dataset Bias

This empirical study demonstrates the effectiveness of the proposed method in security vulnerability/bug report datasets. However, the internal difference between studied datasets and datasets from other security tasks (e.g., in Table 1) or from other domains cannot be ignored. Therefore, there is no guarantee that the findings in this study would still hold in other datasets.

7 OTHER NOTES ON CLASS IMBALANCE

Class-imbalance learning [43] refers to methods to hancle class imbalance issues. Data oversampling is not the only effective way to address data imbalancing issues. Other approaches can mainly fall into the following categorizations according the the problem space:

Data Sampling level. Apart from data oversampling, data undersampling [35, 42] is another alternative way to deal with class imbalance from the data level. In data undersampling, we can remove some instances from majority class. Generally, this method is suggested when there is large number of training instances. However, data undersampling might suffer from information loss due to removal of majority class instances.

Model Training level. Many prior work propose various ways to train efficient models with class imbalanced datasets. For example, bagging ensemble [22] is a technique that divides the original training datasets into several subsets of same size, while each subset is used to train a single classifier, and then the method aggregates individual classifiers into an ensemble classifier. This method is well-known for its simplicity and good generalization ability. Some other work [56] applies hyperparameter optimization on both data pre-processors and machine learning models to explore optimal

hyperparameter settings that work for imbalanced data. Several studies use cost-sensitive learning [12] [11] in the field of imbalanced learning. Cost-sensitive learning takes the costs prediction errors into consideration and does not treat all classification errors as equal. This makes sense in some security scenarios, for example, classifying a benign software as a malware (i.e., false positive case) is less of a problem than classifying a malware case as a benign software (i.e., false negative case) since security practitioners do not hope to miss any malware sample. Furthermore, transfer learning [3] is another promising technique that use auxiliary data to augment learning when training samples are not sufficient. This algorithm works by including a similar and possibly larger dataset, with which perform knowledge transfer.

Feature Selection level. Using feature-selection for addressing class imbalance is not a largely explored research area compared to previous levels. Several few work investigate feature selection with imbalanced data empirically [26, 68], and researchers [4] warns that the extra computational cost would be an issue of concern.

We note that the ***scope of this work*** is not to explore and compare all data imbalance solutions extensively. It is not fair to offer a general conclusion that one technique outperforms other techniques in all tasks. Rather, the focus of this work is to explore the merits of optimized generative adversarial network as an data oversampling technique in addressing security dataset imbalance issues. A hybrid combination of above mentioned approaches (including this work) might work even better, and we would like to explore as an interesting future direction.

8 CONCLUSION

When the target class is rare, as it is often within security datasets, it is hard for a machine learning algorithm to distinguish the goal (security target) from others (the normal events). To address such class imbalance issue, prior researchers in software engineering often use SMOTE (or its variants, see Table 2) as a solution. SMOTE was first proposed in 2002, nearly two decades ago. This paper seeks a better method than SMOTE.

One recent alternative to SMOTE is the Generative Adversarial Networks. This architecture contains two components (the generator and the discriminator) that "fight it out" to generate new examples. The experience has been that it is hard to balance these two components manually, so we experimented with addressing that problem with automatic hyperparameter optimization.

The empirical study shows that GANs with hyperparameter optimization outperforms prior SMOTE (and its variants) and standard GANs (without optimization). For example, Dazzle can achieve an average of about 60% improvement rate over SMOTE in recall on studied dataset among different classifiers. Based on this study, we recommend using GANs with hyperparameter optimization (and not off-the-shelf default settings) to train a good security vulnerability prediction model (from the view of data oversampling). More generally, we suggest using hyperparameter optimization in other tasks in SE community.

REFERENCES

[1] Amritanshu Agrawal and Tim Menzies. 2018. Is" Better Data" Better Than" Better Data Miners"?. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 1050–1061.

[2] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. 2019. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 221–231.

[3] Samir Al-Stouhi and Chandan K Reddy. 2016. Transfer learning for class imbalance problems with inadequate data. *Knowledge and information systems* 48, 1 (2016), 201–228.

[4] Aida Ali, Siti Mariyam Shamsuddin, and Anca L Ralescu. 2013. Classification with class imbalance problem. *Int. J. Advance Soft Compu. Appl* 5, 3 (2013).

[5] Ankesh Anand, Kshitij Gorde, Joel Ruben Antony Moniz, Noseong Park, Tanmoy Chakraborty, and Bei-Tseng Chu. 2018. Phishing URL detection with oversampling based on text generative adversarial networks. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1168–1177.

[6] Martín Arjovsky and Léon Bottou. 2017. Towards Principled Methods for Training Generative Adversarial Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=Hk4_qw5xe

[7] Martín Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 214–223. http://proceedings.mlr.press/v70/arjovsky17a.html

[8] Kwabena Ebo Bennin, Jacky Keung, Passakorn Phannachitta, Akito Monden, and Solomon Mensah. 2017. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering* 44, 6 (2017), 534–550.

[9] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. 2015. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery* 8, 1 (2015), 014008.

[10] James Bergstra, Dan Yamins, David D Cox, et al. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, Vol. 13. Citeseer, 20.

[11] Peng Cao, Dazhe Zhao, and Osmar Zaiane. 2013. An optimized cost-sensitive SVM for imbalanced data learning. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 280–292.

[12] Peng Cao, Dazhe Zhao, and Osmar R Zaïane. 2013. A PSO-based cost-sensitive neural network for imbalanced data classification. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 452–463.

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[14] Yi-Ming Chen, Chun-Hsien Yang, and Guo-Chung Chen. 2021. Using Generative Adversarial Networks for Data Augmentation in Android Malware Detection. In *2021 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 1–8.

[15] Adriel Cheng. 2019. Pac-gan: Packet generation of network traffic using generative adversarial networks. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 0728–0734.

[16] Pedro Ferreira da Costa, Romy Lorenz, Ricardo Pio Monti, Emily Jones, and Robert Leech. 2020. Bayesian Optimization for real-time, automatic design of face stimuli in human-centred research. In *7th ICML Workshop on Automated Machine Learning*.

[17] R Elakkiya, Pandi Vijayakumar, and Neeraj Kumar. 2021. An optimized Generative Adversarial Network based continuous sign language classification. *Expert Systems with Applications* 182 (2021), 115276.

[18] Farzan Farnia and Asuman Ozdaglar. 2020. Do GANs always have Nash equilibria?. In *International Conference on Machine Learning*. PMLR, 3029–3039.

[19] Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán, and Tibor Gyimóthy. 2019. Challenging machine learning algorithms in predicting vulnerable javascript functions. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. IEEE, 8–14.

[20] Ugo Fiore, Alfredo De Santis, Francesca Perla, Paolo Zanetti, and Francesco Palmieri. 2019. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Information Sciences* 479 (2019), 448–455.

[21] Bent Fuglede and Flemming Topsoe. 2004. Jensen-Shannon divergence and Hilbert space embedding. In *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.* IEEE, 31.

[22] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. 2011. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 4 (2011), 463–484.

[23] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Computing Surveys (CSUR)* 50, 4 (2017), 1–36.

[24] Ian Goodfellow. 2016. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).

[25] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative

Adversarial Networks. *CoRR* abs/1406.2661 (2014).
[26] Marko Grobelnik. 1999. Feature selection for unbalanced class distribution and naive bayes. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*. Citeseer, 258–267.
[27] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. 2017. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications* 73 (2017), 220–239.
[28] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. 2005. Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*. Springer, 878–887.
[29] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 1322–1328.
[30] Shuokang Huang and Kai Lei. 2020. IGAN-IDS: An imbalanced generative adversarial network towards intrusion detection system in ad-hoc networks. *Ad Hoc Networks* 105 (2020), 102177.
[31] Abdul Jabbar, Xi Li, and Bourahla Omar. 2021. A survey on generative adversarial networks: Variants, applications, and training. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–49.
[32] Yuan Jiang, Pengcheng Lu, Xiaohong Su, and Tiantian Wang. 2020. LTRWES: A new framework for security bug report detection. *Information and Software Technology* 124 (2020), 106314.
[33] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Christoph Treude. 2020. The impact of automated feature selection techniques on the interpretation of defect models. *Empirical Software Engineering* 25, 5 (2020), 3590–3638.
[34] Harsurinder Kaur, Husanbir Singh Pannu, and Avleen Kaur Malhi. 2019. A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–36.
[35] Michał Koziarski. 2020. Radial-based undersampling for imbalanced data classification. *Pattern Recognition* 102 (2020), 107262.
[36] Patrick Kwaku Kudjo, Selasie Brown Aformaley, Solomon Mensah, and Jinfu Chen. 2019. The significant effect of parameter tuning on software vulnerability prediction models. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 526–527.
[37] Solomon Kullback. 1997. *Information theory and statistics*. Courier Corporation.
[38] Felix Last, Georgios Douzas, and Fernando Bacao. 2017. Oversampling for imbalanced learning based on k-means and smote. *arXiv preprint arXiv:1711.00837* (2017).
[39] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research* 18, 1 (2017), 559–563.
[40] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering* 34, 4 (2008), 485–496.
[41] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Hanchao Qi, and Jie Hu. 2016. Vulpecker: an automated vulnerability detection system based on code similarity analysis. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 201–213.
[42] Wei-Chao Lin, Chih-Fong Tsai, Ya-Han Hu, and Jing-Shang Jhang. 2017. Clustering-based undersampling in class-imbalanced data. *Information Sciences* 409 (2017), 17–26.
[43] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. 2008. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39, 2 (2008), 539–550.
[44] Tim Menzies, Alex Dekhtyar, Justin Distefano, and Jeremy Greenwald. 2007. Problems with Precision: A Response to" comments on'data mining static code attributes to learn defect predictors'". *IEEE Transactions on Software Engineering* 33, 9 (2007), 637–640.
[45] Patrick Morrison, Kim Herzig, Brendan Murphy, and Laurie Williams. 2015. Challenges with applying vulnerability prediction models. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*. 1–9.
[46] John Nash. 1951. Non-cooperative games. *Annals of mathematics* (1951), 286–295.
[47] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. 2007. Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security*. 529–540.
[48] Hien M Nguyen, Eric W Cooper, and Katsuari Kamei. 2011. Borderline oversampling for imbalanced data classification. *International Journal of Knowledge Engineering and Soft Data Paradigms* 3, 1 (2011), 4–21.
[49] Fayola Peters, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. 2019. Text Filtering and Ranking for Security Bug Report Prediction. *IEEE Trans. Software Eng.* 45, 6 (2019), 615–631.
[50] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.06434
[51] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. 2019. Flow-based network traffic generation using generative adversarial networks. *Computers & Security* 82 (2019), 156–172.
[52] Divya Saxena and Jiannong Cao. 2021. Generative Adversarial Networks (GANs) Challenges, Solutions, and Future Directions. *ACM Computing Surveys (CSUR)* 54, 3 (2021), 1–42.
[53] Md Hasan Shahriar, Nur Imtiazul Haque, Mohammad Ashiqur Rahman, and Miguel Alonso. 2020. G-ids: Generative adversarial networks assisted intrusion detection system. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 376–385.
[54] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
[55] Yi Shi, Yalin E Sagduyu, Kemal Davaslioglu, and Jason H Li. 2018. Generative adversarial networks for black-box API attacks with limited training data. In *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE, 453–458.
[56] Rui Shu, Tianpei Xia, Jianfeng Chen, Laurie Williams, and Tim Menzies. 2021. How to Better Distinguish Security Bug Reports (Using Dual Hyperparameter Optimization). *Empirical Software Engineering* 26, 3 (2021), 1–37.
[57] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.
[58] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online defect prediction for imbalanced data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 99–108.
[59] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2018. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering* 45, 7 (2018), 683–711.
[60] Tram Truong-Huu, Nidhya Dheenadhayalan, Partha Pratim Kundu, Vasudha Ramnath, Jingyi Liao, Sin G Teo, and Sai Praveen Kadiyala. 2020. An Empirical Study on Unsupervised Network Anomaly Detection using Generative Adversarial Networks. In *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*. 20–29.
[61] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81 (2019).
[62] James Walden, Jeff Stuckman, and Riccardo Scandariato. 2014. Predicting vulnerable components: Software metrics vs text mining. In *2014 IEEE 25th international symposium on software reliability engineering*. IEEE, 23–33.
[63] Zhengwei Wang, Qi She, and Tomas E Ward. 2021. Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–38.
[64] Bin Wu, Le Liu, Yanqing Yang, Kangfeng Zheng, and Xiujuan Wang. 2020. Using improved conditional generative adversarial networks to detect social bots on Twitter. *IEEE Access* 8 (2020), 36664–36680.
[65] Xiaoxue Wu, Wei Zheng, Xin Xia, and David Lo. 2021. Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction. *IEEE Transactions on Software Engineering* (2021).
[66] Qiao Yan, Mingde Wang, Wenyao Huang, Xupeng Luo, and F Richard Yu. 2019. Automatically synthesizing DoS attack traces using generative adversarial networks. *International Journal of Machine Learning and Cybernetics* 10, 12 (2019), 3387–3396.
[67] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. 2017. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. 5907–5915.
[68] Zhaohui Zheng, Xiaoyun Wu, and Rohini Srihari. 2004. Feature selection for text categorization on imbalanced data. *ACM Sigkdd Explorations Newsletter* 6, 1 (2004), 80–89.
[69] Lin Zhu, Yushi Chen, Pedram Ghamisi, and Jón Atli Benediktsson. 2018. Generative adversarial networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing* 56, 9 (2018), 5046–5063.