Knowledge and Information Systems volume 60, pages 569–590, 2019. https://doi.org/10.1007/s10115-018-1306-7

Variational data generative model for intrusion detection

Manuel Lopez-Martin [1], Belen Carro [1], Antonio Sanchez-Esguevillas[1]

1 Dpto. TSyCeIT, ETSIT, Universidad de Valladolid, Paseo de Belén 15, Valladolid 47011, Spain ; mlopezm@acm.org ; belcar@tel.uva.es ; antoniojavier.sanchez@uva.es;

Correspondence: mlopezm@acm.org

The authors declare that there is no conflict of interest regarding the publication of this paper

Abstract

A Network Intrusion Detection System (NIDS) is a system which detects intrusive, malicious activities or policy violations in a host or hosts network. It is very important for any detection system the ability to access balanced and diversified data to train the system. Intrusion data rarely have these characteristics, since samples of network traffic are strongly biased to normal traffic, being difficult to access traffic associated with intrusion events.

Therefore, it is important to have a method to synthesize intrusion data with a probabilistic and behavioral structure similar to the original one. In this work we provide such a method.

Intrusion data have continuous and categorical features, with a strongly unbalanced distribution of intrusion labels. That is the reason why we generate synthetic samples conditioned to the distribution of labels. That is, from a particular set of labels, we generate training samples associated with that set of labels, replicating the probabilistic structure of the original data that comes from those labels.

We use a generative model based on a customized Variational Autoencoder (VAE), using the labels of the intrusion class as an additional input to the network. This modification provides an advantage, as we can readily generate new data using only the labels, without having to rely on training samples as canonical representatives for each label, which makes the generation process more reliable, less complex and faster. We show that the synthetic data are similar to the real data, and that the new synthesized data can be used to improve the performance scores of common machine learning classifiers.

Keywords: Intrusion detection system; Variational methods; Generative model; Neural Networks.

1. Introduction

The objective of a Network Intrusion Detection System (NIDS) is to automate the detection of policy and security violations and malicious activities against a host that is part of a network. As the importance and volume of data exchange increases, it is more relevant to increase the performance of NIDS. Most current NIDS are based on supervised machine learning models which require labeled data to perform model training.

It is fundamental for any detection system the possibility of accessing balanced and diversified data to train the system. Intrusion data rarely have these characteristics, as the network traffic samples are strongly biased to the normal type of traffic, being difficult to access traffic associated to anomalous intrusion events. Therefore, it would be very interesting to be able to synthesize intrusion data with a structure similar to the real data. In this way, we avoid investing significant resources in tasks such as obtaining additional intrusion data or manually simulating attacks.

The generation of synthetic data that resemble real data, being similar (in a probabilistic sense) but not identical, has long been an objective in the area of image processing [1][2]. In this area the features used to train the models are all continuous (pixel intensities) and, additionally, it is relatively easy to appreciate if a generative model is working well, as we (humans) are quite good at identifying whether the images generated corresponds to the class of images we want. Similar works have been carried out more recently in the generation of text/sentences [3][4]. In this case the features are all discrete, and we can appreciate directly, in a similar way, if the generated text corresponds to a particular topic.

In the intrusion detection area the generative data process has its own difficulties, due to several reasons: (1) the features used to identify an intrusion type (label) are both continuous and categorical, (2) the class labels are highly unbalanced, and (3) we cannot directly appreciate whether a new synthetic sample of a particular class really correspond to that class (intrusion label). The first and second imply that we need to synthesize, at the same time, discrete and continuous features, each having its own problematic. The third requires developing alternative techniques to show the similarity of original and synthetic data. We need to identify if two populations of samples (real and synthetic) belong to the same class. Taking into account that the samples represent multivariate high-dimensional vectors, with continuous and discrete values, complex joint probability distribution and with non-Gaussian marginals

(or another easily parameterized distribution), it is very difficult to apply methods based on information theory (e.g. KL divergence) or multivariate extensions of goodness-of-fit tests to identify the similarity between the probability distributions of the two populations (Section 4.1). That is why we have applied other alternative and original approaches to evaluate similarity: (1) Extended histograms of the original and synthesized features, and, (2) demonstrate that classification results are similar when using either the original or synthesized data as training or testing data for several classifiers (e.g. Random Forest, Multinomial Logistic Regression…).

The problem mentioned above is not found when generating synthetic images or text, since, as already discussed, we can discriminate samples that belong to specific objects (images) or topics (text). In the case of samples related to intrusion detection, there is not such good discriminator. To appreciate the complex and unclear relationship between the distributions of values of a high-dimensional sample with the label associated to that sample, we could consider the difficulties imposed by adversarial examples [5] to a neural network and how the addition of small perturbations to images can mislead a perfectly tuned neural network, resulting in misclassified images, even when these perturbations do not affect the discrimination capacity of humans.

In this work we provide a method to generate data of similar probabilistic structure to intrusion detection data, having both continuous and categorical features and being strongly unbalanced to some of their associated labels. We generate the data conditioned to the specific class (label) to which we want the data to belong. That is, from a particular set of labels we generate training samples associated to that set of labels, reflecting real data that comes from those labels.

We call the method Variational Generative Model (VGM). We use a generative model based on a Conditional Variational Autoencoder (VAE), using the intrusion class labels as input. This modification provides an advantage, as we can readily generate new data using only the labels, without having to rely on specific training samples that represent or are associated to specific labels. Furthermore, the new synthesized data can be used as new additional training data to improve classification results for common machine learning classifiers. These results also confirm that the synthesized data have similar structure to the original but not been identical which allows to improve the performance of a classifier.

The problem presented here can be considered similar to the one faced by classification with an imbalanced dataset, which is mainly addressed with four strategies [6][7][8]: resampling, cost-sensitive, algorithmic and ensemble. To compare VGM with equivalent approaches, we focus on resampling.

Resampling can be achieved creating new minority class samples (over-sampling) or reducing the number of majority class samples (under-sampling). An effective way to perform over-sampling is by creating new synthetic data that resembles the original data. The state-of-the-art (SOTA) algorithms in synthetic over-sampling are based on SMOTE [9] and its numerous variants [10][11]; being its main idea to create new samples close in '*distance'* to existing samples that belongs to some specific minority class. The different variants consider alternative approaches to calculate the *distance* function and the proximity to majority class samples (borderline). To avoid possible over-fitting due to synthetic data, there is the possibility to combine over-sampling and under-sampling methods [12][13]. Another interesting method is ADASYN[14], which is similar to SMOTE, but giving more weight to samples that are harder to learn (closer to other majority class samples). Finally, there is the possibility to perform ensemble sampling with methods similar to EasyEnsemble [15].

Our method (VGM) is a generative method to synthesize new data belonging to any class label. The main difference between VGM and SMOTE (and its variants) is that VGM is based in a latent probability distribution learned from data, instead of being based in a predefined *'distance'* function. VGM does not need to assume any '*distance'* function, or to impose rules on the importance of proximity to majority class samples, which would be additional hyper-parameters to explore.

We provide a comparison of the proposed model with seven SOTA synthetic data generation algorithms (SMOTE, ADASYN…), showing that synthetic data generated by VGM provides better performance metrics (average accuracy and F1) when several common classifiers are trained with this data instead of data from other alternative generation algorithms. To train the VGM model, we need original data from a well-known intrusion detection dataset, for which we have chosen the NSL-KDD data set [16]. We have explored several architectures for VGM, considering different number of layers, nodes, regularization, loss functions and probability distribution for the output layer. We present the different options and the results obtained.

As a summary, the contributions of this paper are: (1) It is the first application of a conditional variational autoencoder to generate synthetic data in the intrusion detection field. (2) We present original methods to show similarity of real and synthetic data. (3) VGM provides more useful synthetic samples than comparable SOTA over-sampling algorithms, corroborated by better performance (accuracy, F1) produced by various classifiers when using synthetic data generated by VGM.

The paper is organized as follows: Section 2 presents related works. Section 3 describes the work performed. Section 4 describes the results obtained and finally, Section 5 provides discussion and conclusions.

2. Related works

As far as we know, there is no previous application of a variational generative model based on neural networks to generate data of similar probabilistic structure to intrusion detection data. Therefore, the work presented in this paper is original in essence. There are a number of works for the application of variational generative models to generate images [1][2][17] and text [3][4], but there is none in the area of intrusion detection.

There is no work, similar to the present one, generating both continuous and categorical features. In [18] is presented a model that handles a discrete distribution for the "latent" layer, but it is applied to images with continuous features. In [19] the authors provide a solution using a VAE in the intrusion detection field, but it is used exclusively to implement a classifier, not to generate synthetic data according with the intrusion class as in the present work

There is a vast number of works applying classification algorithms to NIDS [20][21]. This paper is not related specifically with any classification technique, but we will show (Section 4.2) that the synthetic data generated with our model improves results obtained with different classifiers. Therefore, it is interesting to provide a summary of results on classification using the NSL-KDD dataset, which will help to put into perspective the results presented in this work. It is important to mention that comparison of results in this field is difficult due to: (1) diversity of reported performance metrics; (2) the aggregation of classification labels in different sets (e.g. 23 labels can be grouped hierarchically in different subsets or categories: 23, 5 or 2 final labels) making it difficult to compare results for different subsets; (3) reporting results on unclear test datasets. This last point is important to mention, because for example, for the NSL-KDD dataset, 16.6% of samples in the test dataset correspond to labels not present at the training dataset. This is an important property of this dataset and creates an additional difficulty to the classifier. These difficulties are shown in detail in [16].

Classification results for NSL-KDD are provided in several works. In [22] is achieved an accuracy of 79.9% for test data, for the 5-labels intrusion scenario. In [23] they provide, for the 2-labels scenario a recall of 75.49% on test data. Authors in [20] explain the reasons to create the NSL-KDD data set,

providing results for several algorithms, being 82.02% the best accuracy reported when using the full NSL_KDD dataset for training and testing, for the 2-labels scenario.

There is large literature related to algorithms dealing with imbalanced datasets [6][7][8], and in particular presenting algorithms for synthetic over-sampling [9][10][11], adaptive over-sampling [14], over-sampling followed by under-sampling [12], ensemble sampling [15] and specific combinations of methods [13].

3. Work description

In this section we present the dataset used for this work, a description of the variational method employed and details on different variants of the method.

3.1. Selected dataset

We have chosen the NSL-KDD [16] dataset as our reference dataset. NSL-KDD is an enhanced version of the original KDD-99 dataset, solving the problem of redundant records present in KDD-99. We consider that this dataset is useful for this work, as we are mainly interested in generation of synthetic data, for which NSDL-KDD provides a sufficient number of samples. Additionally, the distribution of samples among intrusion classes (labels) is quite unbalanced, and provides enough variability between training and test data to challenge any method that tries to reproduce the structure of the data.

The NSL-KDD dataset provides 125973 training samples and 22544 test samples, with 41 features, being 38 continuous and 3 categorical (discrete valued). Each training sample has a label output from 23 possible labels (normal plus 22 labels associated to different types of anomaly). The test data has the same number of features (41) and output labels from 38 possible values. That means that the test data has anomalies not presented at training time. The 23 training and 38 testing labels have 21 labels in common; 2 labels only appear in training and 17 labels are unique to the testing data.

Around 16% of the samples in the test dataset correspond to labels unique to the test dataset, and which were not present at training time. The existence of new labels at testing introduces an additional challenge to the learning methods, which is important to verify the robustness of the classifiers, but not for the purpose of this study, which is to synthesize samples associated to existing labels. Therefore, it seems more practical and useful to aggregate labels by categories. As presented in [16], the original labels

are associated to 5 categories: NORMAL, PROBE, R2L, U2R and DoS, with the latter four corresponding to an anomaly. The meaning of the 5 categories is as follows:

- NORMAL: There is no attack
- Denial of Service (DoS): The intention of these attacks is to interrupt some service.
- PROBE: They intend to gain information about the target host.
- User to Root (U2R): U2R attacks try to obtain root access to the system.
- Remote to Local (R2L): Unauthorized access from a remote machine.

For this work we have used these 5 categories as the labels driving our data generation model.

We have performed an additional data transformation: scaling all NSL-KDD continuous features to the range [0,1] and one-hot encoding all categorical features. This provides a final dataset with 116 features: 32 continuous and 84 with values in {0,1} associated to the three one-hot encoded categorical features.

It is important to note that the 3 categorical features: *protocol*, *flag* and *service* have respectively 3, 11 and 70 distinct values. We will show later the accuracy obtained when synthesizing these features (having as reference the original ones), and how the different number of values impact on the results.

Working with the NSL-KDD dataset, we provide all results using the full training dataset of 125973 samples and the full test dataset of 22544 samples. It is also important to mention that we do not use a previously customized training or test datasets, neither a subset of them, what may provide better alleged results but being less objective and also missing the point to have a common reference to compare.

3.2. Method explained

In Figure 1 we present a diagram comparing VGM and VAE architectures. In a VAE architecture [1] we model the internal structure of data with an initial neural network (encoder) that approximates the parameters of a probability distribution. This probability distribution is used to draw samples that are the input to a second neural network (decoder) that approximates the parameters of a second probability distribution from which the samples drawn are the final output of the VAE.

To train a VAE we use the same data for input and output, trying to implement the identity function but with an intermediate layer with a lower dimension. Contrary to an Autoencoder [24] that is used as a dimensionality reduction mechanism, with a VAE we will not obtain a deterministic lower dimension representation of the data, but a set of parameters that define an associated set of probability distributions

that represent the data. In other words, instead of mapping an input sample in $\mathcal{R}^m$ to a deterministic output (latent variable) in $\mathcal{R}^n$, where usually $n < m$, what we do in a VAE is to map samples in $\mathcal{R}^m$ to n probability distributions. The samples drawn from these n probability distributions form a new stochastic feature vector in $\mathcal{R}^n$. The new latent variable is not deterministic but stochastic (variable $\boldsymbol{Z}$ in Figure 1).

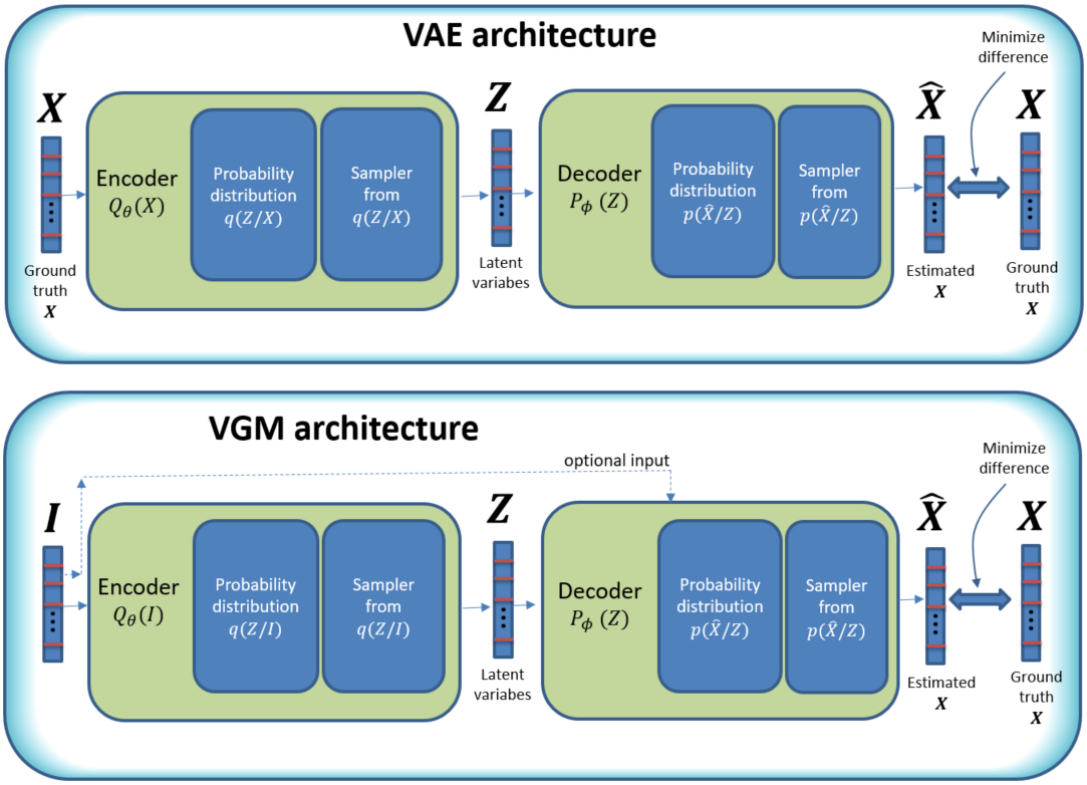

Figure 1. Comparison of VGM with a typical VAE architecture

The interest of a VAE is that, after the model is trained, it is sufficient to sample the intermediate layer (latent variables) to generate new data that resembles the data used for training.

The left part of the diagram of VAE in Figure 1 corresponds to the encoder which depends on the input data ($\boldsymbol{X}$) and some model parameters ($\boldsymbol{\theta}$) and produces the "latent" probability distributions ($\boldsymbol{q(Z/X)}$) and stochastic latent variable ($\boldsymbol{Z}$). To the right of the diagram we have the decoder which depends both on the latent variable ($\boldsymbol{Z}$) and some other model parameters ($\boldsymbol{\phi}$) and produces a new set of output probability distributions ($\boldsymbol{p(\hat{X}/Z)}$) from which we sample the final output ($\boldsymbol{\hat{X}}$).

The model parameters: $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, are used as a brief way to represent the architecture and weights of the neural network used. These parameters are tuned as part of the VAE training process and are considered constant later on.

The probability distributions $\boldsymbol{p}(\widehat{\boldsymbol{X}}/\boldsymbol{Z})$ and $\boldsymbol{q}(\boldsymbol{Z}/\boldsymbol{X})$ are conditional probability distributions, and they are parameterized, which means that they are completely defined by a set of parameters (e.g. the mean and variance for a normal distribution).

The final objective is to produce an output $\widehat{\boldsymbol{X}}$ with a minimum difference to the input $\boldsymbol{X}$. There are different ways to achieve this objective, one is to use sampling methods as Markov Chain Monte Carlo (MCMC), but the path taken by VAE is different. VAE uses a variational approach that tries to maximize the log likelihood of X by maximizing a quantity known as the Evidence Lower Bound (ELBO) [1]. The ELBO is formed by two parts: (1) a measure of the distance between the probability distribution $\boldsymbol{q}(\boldsymbol{Z}/\boldsymbol{X})$ and a reference probability distribution of the same nature (actually a prior distribution for $\boldsymbol{Z}$), where the distance usually employed is the Kullback-Leibler (KL) divergence, and (2) the log likelihood of $\boldsymbol{p}(\boldsymbol{X})$ under the probability distribution $\boldsymbol{p}(\widehat{\boldsymbol{X}}/\boldsymbol{Z})$, which is the probability to obtain the desired data ($\boldsymbol{X}$) with the probability distribution that produces $\widehat{\boldsymbol{X}}$.

Using the ELBO, we reduce the problem to an optimization problem based on a maximization of the ELBO, which allows using neural networks with stochastic gradient descent (SGD) as the optimizer. The only problem remains on how to incorporate the sampling process, required by the model, with the way SGD operates. To do this, the innovation of VAE is to use what is called the "reparameterization trick" [1]. Using this trick, all the variables involved are connected through differentiable layers on which SGD can operate.

Based on the VAE model, our proposed method (VGM) is similar to a VAE but instead of using the same vector of features for the input and output of the network, we add more flexibility allowing to have a different input to the network and to add an optional additional input on the decoder block (Figure 1, lower diagram). We represent this generic input in Figure 1 with the letter $\boldsymbol{I}$. The input $\boldsymbol{I}$ can be instantiated in two possible ways: as the vector of sample features, as in VAE, or as the vector of labels associated to the samples. That is, the input $\boldsymbol{I}$ can be either $\boldsymbol{X}$ or $\boldsymbol{L}$, where $\boldsymbol{L}$ is the vector of labels.

In the VGM architecture, in case we use the vector of features as input to the encoder (as in VAE) then we will employ an additional input to the decoder formed by the vector of labels. In this way we will always have the vector of labels as an input to the network, either as input to the encoder or decoder blocks.

To use the labels as input of the generative process is an important difference as it allows generating new synthesized samples using exclusively the labels assigned to these samples. As already pointed out,

for intrusion detection data, the generative data process is more difficult, as the features are both continuous and categorical, and we cannot appreciate directly if the synthesized data samples have features with a similar structure to the original ones, that is the reason why using directly the labels is important to be sure we are using meaningful information to characterize the generated samples.

In Section 3.3 we will present different variants to the generic architecture for VGM shown in Figure 1. In Figure 2 we present the elements of the loss function to be minimized by SGD for the VGM model. We can see that, as mentioned before, the loss function is made up of two parts: a KL divergence and a log likelihood part. The second part takes into account how probable is to generate **X** by using the distribution $\boldsymbol{p(\hat{X}/Z)}$, that is, it is a distance between $\boldsymbol{X}$ and $\boldsymbol{\hat{X}}$. The KL divergence part can be understood as a distance between the distribution $\boldsymbol{q(Z/I)}$ and a prior distribution for $\boldsymbol{Z}$, that we identify as $q_{\ reference}$ in Figure 2. By minimizing this distance, we are really avoiding that $\boldsymbol{q(Z/I)}$ departs too much from its prior, acting finally as a regularization term. The nice feature about this regularization term is that it is automatically adjusted, and it is not necessary to perform cross-validation to adjust a hyper-parameter associated to the regularization, as it is needed in other models (e.g. ridge regression, soft-margin support vector machines…).

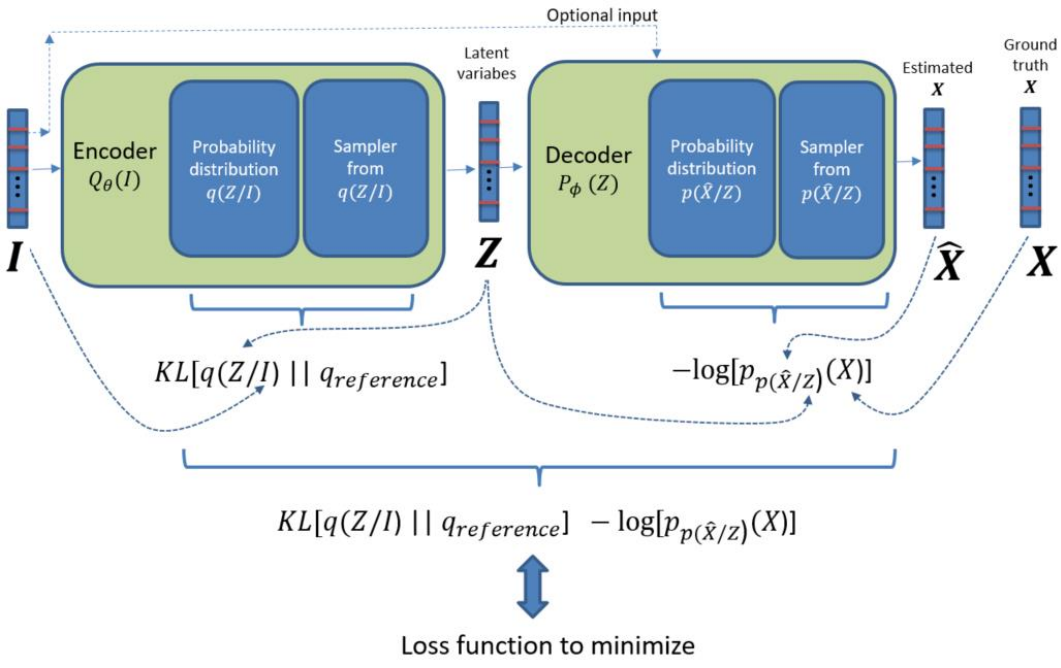

Figure 2. Details on the loss function elements for the VGM model.

3.3. Variants of the model

We have explored three options for the general VGM model discussed in the previous section.

3.3.1. Option A. Model with Gaussian and Bernoulli distributions

In this option we have the vector of labels as input to the network and the vector of features as output. Figure 3 presents option A.

We use a multivariate Gaussian as the distribution for $\boldsymbol{q(Z/L)}$, with a mean $\boldsymbol{\mu(L)}$ and a diagonal covariance matrix: $\boldsymbol{\Sigma(L)} \rightarrow \boldsymbol{\sigma_i^2(L)}$, with different values along the diagonal. We have a standard normal $\boldsymbol{N(0, I)}$ as the prior distribution for **Z.**

For the distribution $\mathbf{p(\hat{X}/Z)}$ we use a multivariate Bernoulli distribution. The nice property about the Bernoulli distribution is that we do not require doing sampling on it, as the output parameter that characterizes the distribution is the mean that is the same as the probability of success. Then, in this case, the output of the last layer is taken as our final output $\boldsymbol{\hat{X}}$.

The selection of distributions for $\boldsymbol{q(Z/L)}$ and $\mathbf{p(\hat{X}/Z)}$ is similar to the distributions selected in [1], since they are simple and provide good results.

In Figure 3 we show also (in squared boxes) the elements of the loss function to be minimized for this option.

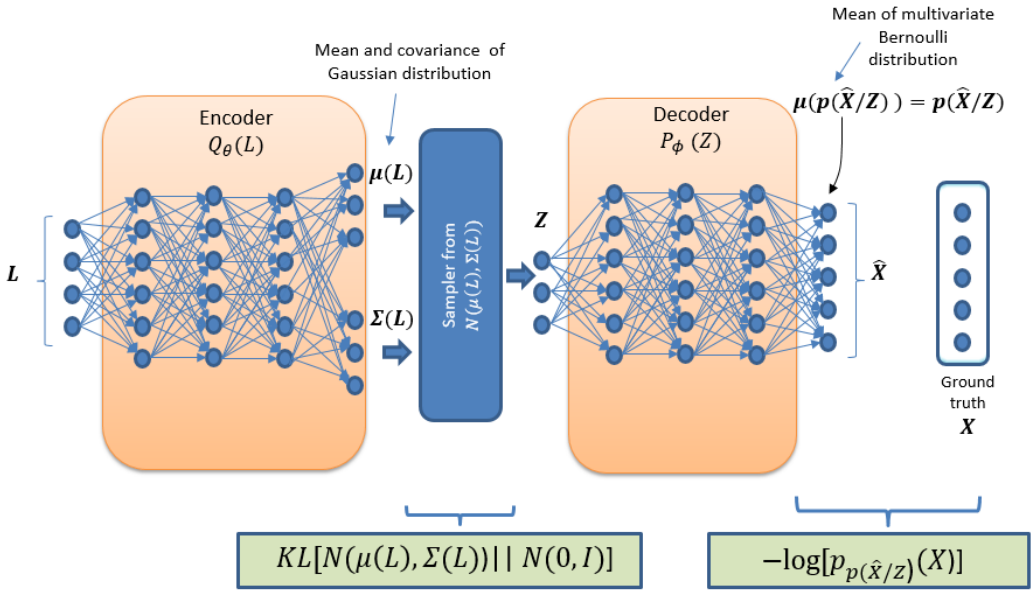

Figure 3. Option A. VGM model with Gaussian and Bernoulli distributions.

3.3.2. Option B. Model with Gaussian and Bernoulli distributions plus RMSE for continuous features

Option B is presented in Figure 4. This option is similar to option A, but we have divided the output layer according to the separation between discrete and continuous features. We treat the discrete features as in Option A, and for the continuous features we change the loss function to the root mean square error (RMSE) between original and generated continuous features, instead of the log-likelihood, as in option A.

We have tried this change in the loss function to see whether we could obtain an improvement by separating the behavior of continuous and discrete features. The change can be also justified by considering that minimizing an RMSE loss function is equivalent to minimizing the negative log-likelihood of an implicit Gaussian distribution for the last decoder layer (in accordance with ELBO theory).

In Figure 4, in squared boxes are the elements of the loss function to be minimized for this option.

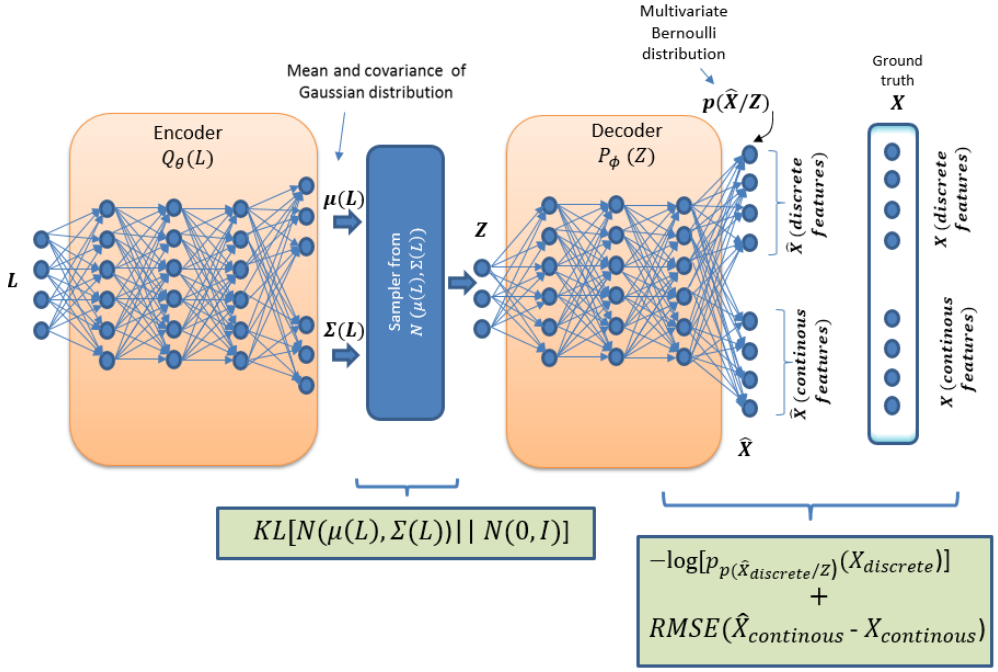

Figure 4. Option B. VGM model with Gaussian and Bernoulli distributions plus RMSE for continuous features.

3.3.3. Option C. Model based on Conditional VAE with Gaussian and Bernoulli distributions

The last option is presented in Figure 5. This option is similar to option A, but with an important difference. Instead of using the labels as the input for the encoder we use it as an additional input to the decoder**.** We leave the features vector (***X***) as input to the encoder. The architecture is similar to a normal VAE except for the inclusion of the label vector as a supplementary input to the decoder.

In order to get the label vector inside the decoder network we just concatenate it with the values of the first layer of the decoder block (Figure 5). The one-hot encoded label vector has a length of 5 that is too short to have a significative impact in the output, which is why we replicate the label vector 40 times before we concatenate it with the first decoder layer. Then, what is represented in Figure 5 as the vector ***L*** is really a vector of length 200 (the label vector replicated 40 times). The number of times we replicate the label vector can be considered as a hyperparameter of the model: this number modifies the results but not in a very significant way; it can be considered as a part of the fine tuning of the model.

In Figure 5, in squared boxes are the elements of the loss function to be minimized for this option.

This option has produced the best results.

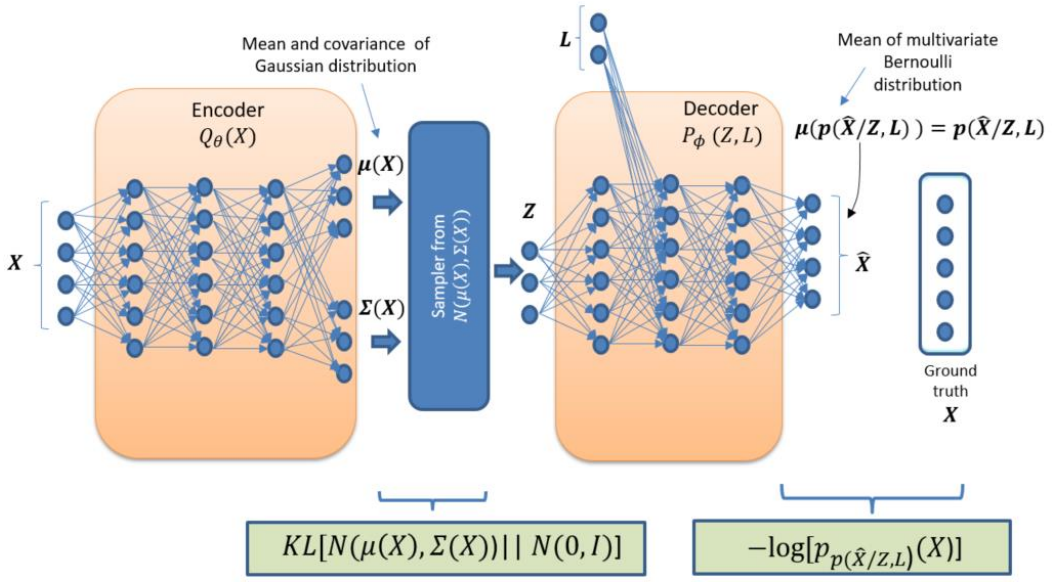

Figure 5. Option C. VGM model including the labels in the decoder with Gaussian and Bernoulli distributions. Training phase.

The process of training and data generation is different for this model when compared with previous models. In previous models, both the training and generation phases are done with the labels as input and the feature vectors as outputs, using the encoder and decoder blocks for the two phases.

For this model, after the training phase is done, we will employ only the decoder block of the trained model to generate new samples (Figure 6). To achieve that, we will provide two inputs to the decoder block: a vector of selected labels and a vector of random values sampled from a standard normal distribution with zero mean and unit variance. The output of the decoder block will be the desired generated samples. These samples are constructed using the probability distributions of the features associated to the labels given as input.

All the models presented, for all options, have the same objective: to be able to generate new data relying exclusively on the labels at the generation phase. This objective is also maintained in this option, because at generation phase we need only the labels and a vector of standard normal random values which are independent of everything else.

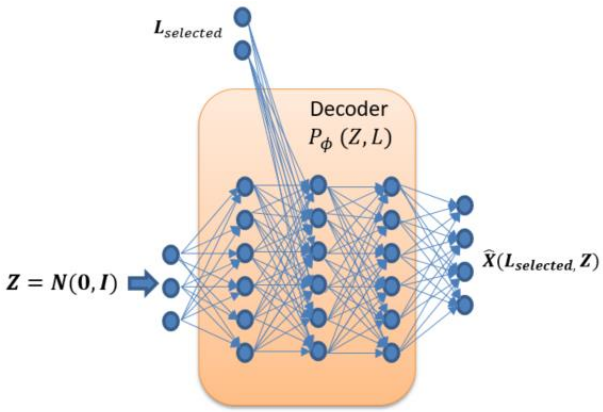

Figure 6 Option C. VGM model including the labels in the decoder with Gaussian and Bernoulli distributions. Generation phase.

4. Results

The objective of this section is first to prove that the synthetic data is similar but not identical to the original data (Section 4.1), and, this similarity is maintained when the data is conditionally partitioned by its class label. And, secondly, to show that the new synthetic data can be used as new training data, improving the results obtained with several prediction algorithms (Section 4.2)

4.1. Structure of generated data

In this section we will show that the synthetic generated data have similar probabilistic structure to the original data. Verifying this similarity is a hard problem since it involves comparing the probability distributions of multivariate vectors (116 features) with non-Gaussian marginals (discrete and continuous features) and complex joint probability distributions. The challenge is twofold: obtain the joint probability distributions and compare them. Methods based on information theory (eg Kullback-Leibler (KL) divergence) require an estimate of joint probabilities that is very difficult for high-dimensional variables [25], and many of them are not practically applicable for multivariate distributions (e.g mutual information and KL divergence) [26]. Other methods based in multivariate extensions of goodness-of-fit tests are also difficult to apply considering the high dimensionality and non-Gaussian marginal distributions [27][28][29]

Considering the difficulties mentioned above, we have developed several approaches to verify the similarity: (1) extended histograms of the original and synthesized features; and (2) classification results obtained from the application of original and synthesized data to several classification algorithms.

Considering the first approach, Figure 7 presents extended histograms for the original NSL-KDD training dataset (upper diagram) and a synthesized dataset created with the same labels as the original one (lower diagram). To visualize the data in Figure 7, we use [30] which makes possible to visualize and compare the distributions of large datasets. The columns of the diagrams correspond to features. The rightmost 4 columns are the categorical variables, respectively: protocol (3 values), service (70 values), flag (11 values) and label (5 values). All features values are ordered in accordance with the alphabetical order of the label. The rows are divided in 100 slots associated to 100 bins where the continuous features have been mapped. The colors in the slots represent where the mean value is for that slot, with a different color to show the dispersion (similar to a box-plot)

We can observe, in Figure 7, that both diagrams present a similar distribution over the features. The intention of the diagram is to show the general similarity of distributions, providing an overall impression of similarity when comparing feature to feature from original and synthetic data. This is the reason why we do not give the names of the features in the diagram, since we are not interested here in a comparison of particular features.

It is important to note that the synthesized data corresponds to data generated from a forward pass of the model (Option C, Section 3.3.3), and each time we generate a new set of synthesized data this dataset will be different, due to the stochastic nature of the layer of latent variables.

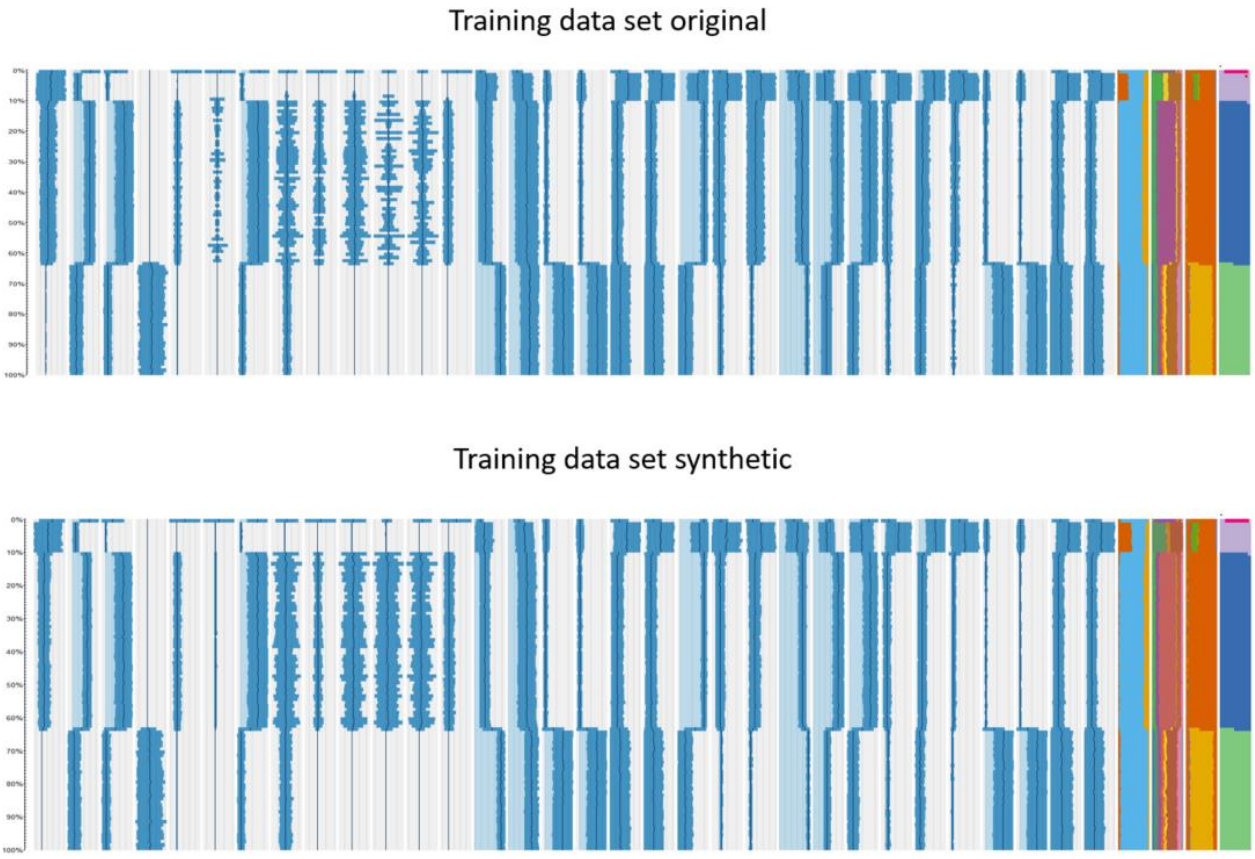

Figure 7. Extended histogram for original NSL-KDD training dataset and a synthesized one generated from the same labels as the original.

Regarding the second approach to check the similarity between original and synthesized features, in Table 1 we present the accuracy obtained in reproducing the three discrete features when using the NSL-KDD Training and Test dataset as original data. In each case, we compare the accuracy between the original dataset and a synthesized dataset composed of the same labels as the original. We see that the accuracy depends on the number of different values of the feature, and, in general, it is quite high. Option C offers the best results, providing a reconstruction accuracy for discrete features that is greater than 90% for most features. We can see that even when the much smaller NSL-KDD test dataset is used to train VGM, we still get very good accuracy results (the three columns on the right in Table 1). The values in Table 1 are color-coded; where the greenest is better and the redder is worse (comparison of values is applied column-wise). We base our definition of accuracy in the usually accepted one [20].

	Accuracy for discrete features, results with NSL-KDD Training dataset			Accuracy for discrete features, results with NSL-KDD Test dataset		
	flags	protocol	service	flags	protocol	service
Option A	0.82711	0.81517	0.46315	0.65822	0.83747	0.44451
Option B	0.81456	0.81517	0.44973	0.65277	0.83747	0.43320
Option C	0.99371	0.99660	0.88315	0.90046	0.98603	0.74676

Table 1. Accuracy when reproducing discrete features

We can see in Table 1 that the results differ for the column: *flags*. This is due to the fact that NSL-KDD has differences between the training and test datasets, as presented in Section 3.1. This difference provides an additional challenge to the data generation process.

Following the same strategy, in Table 2, we present the classification results applying original and synthesized data to several classifiers. We have tested four different classifiers: Random Forest, Logistic Regression, Linear SVM and Multilayer Perceptron (MLP). In all cases, we used the NSL-KDD Training dataset as real data and several synthesized datasets created from a similar labels distribution, as synthetic data.

The objective here is to show that we have similar accuracies when doing prediction with either the original or the synthetic datasets, that is, referring to Table 2, the objective is to have similar values on columns with the same color for each particular option and classifier (see Table 2). We see that this is the case for most tests with all options; the results are quite similar when using either the original or synthesized data as training or prediction data, what provides additional arguments on their similarity. It is also interesting to note the poor results of almost all the options when using Linear SVM and training with synthetic data; the only option that behaves well in this case is Option C, which performs well and robustly in all tests. Option C presents the best results in Table 2, since the accuracy values remain very similar for each classifier.

The classifiers are set-up with their defaults parameters (no tuning), since we are not using the classifiers to show best performance, but to show if they provide similar performance when using the original and generated data, and we do not want to modify the results by tuning the parameters.

	Accuracy: Random Forest				Accuracy: Linear SVM			
Training with....	Real dataset	Real dataset	Synthetic dataset	Synthetic dataset	Real dataset	Real dataset	Synthetic dataset	Synthetic dataset
Prediction with....	Synthetic dataset	Real dataset	Real dataset	Synthetic dataset	Synthetic dataset	Real dataset	Real dataset	Synthetic dataset
Option A	0.98497	0.99986	0.85887	0.85964	0.97502	0.98433	0.20608	0.97613
Option B	0.93022	0.99986	0.86099	0.97973	0.93713	0.98433	0.09812	0.93495
Option C	0.98400	0.99986	0.94993	1.00000	0.98790	0.98433	0.93729	0.99963

	Accuracy: Logistic Regression				Accuracy: MLP			
Training with....	Real dataset	Real dataset	Synthetic dataset	Synthetic dataset	Real dataset	Real dataset	Synthetic dataset	Synthetic dataset
Prediction with....	Synthetic dataset	Real dataset	Real dataset	Synthetic dataset	Synthetic dataset	Real dataset	Real dataset	Synthetic dataset
Option A	0.98509	0.99317	0.80487	0.98516	0.97419	0.99904	0.83356	0.97669
Option B	0.94000	0.99317	0.69931	0.93935	0.93632	0.99904	0.87717	0.94002
Option C	0.99300	0.99317	0.95297	0.99974	0.99753	0.99904	0.96736	0.99998

Table 2. Prediction accuracy with different classifiers and datasets (original and synthetic)

To expand the data provided in Table 2, we present, in Table 3, the prediction details using a contingency table for the 5 predicted labels, when we use Option C with MLP as a classifier. We can see that the predicted and actual percentages of labels are very similar for the three most frequent labels, the least frequent being the most prone to errors, as expected.

		Training with real & Prediction with synthetic						
		Prediction						
		DOS	NORMAL	PROBE	R2L	U2R	Total	%
Ground Truth	OS	45712	180	35	0	0	45927	36.46%
	NORMAL	78	67074	59	131	1	67343	53.46%
	PROBE	0	314	11321	21	0	11656	9.25%
	R2L	22	105	0	868	0	995	0.79%
	U2R	1	8	3	18	22	52	0.04%
	Total	45813	67681	11418	1038	23	125973	100%
	%	36.37%	53.73%	9.06%	0.82%	0.02%	100%	

		Training with real & Prediction with real						
		Prediction						
		DOS	NORMAL	PROBE	R2L	U2R	Total	%
Ground Truth	DOS	45927	0	0	0	0	45927	36.46%
	NORMAL	9	67299	11	22	2	67343	53.46%
	PROBE	2	29	11625	0	0	11656	9.25%
	R2L	0	33	0	962	0	995	0.79%
	U2R	0	13	0	0	39	52	0.04%
	Total	45938	67374	11636	984	41	125973	100%
	%	36.47%	53.48%	9.24%	0.78%	0.03%	100%	

		Training with synthetic & Prediction with real						
		Prediction						
		DOS	NORMAL	PROBE	R2L	U2R	Total	%
Ground Truth	DOS	45751	120	51	5	0	45927	36.46%
	NORMAL	622	65002	446	1186	87	67343	53.46%
	PROBE	22	436	11181	16	1	11656	9.25%
	R2L	2	47	10	912	24	995	0.79%
	U2R	0	12	0	9	31	52	0.04%
	Total	46397	65617	11688	2128	143	125973	100%
	%	36.83%	52.09%	9.28%	1.69%	0.11%	100%	

		Training with synthetic & Prediction with synthetic						
		Prediction						
		DOS	NORMAL	PROBE	R2L	U2R	Total	%
Ground Truth	DOS	45927	0	0	0	0	45927	36.46%
	NORMAL	0	67343	0	0	0	67343	53.46%
	PROBE	0	1	11655	0	0	11656	9.25%
	R2L	0	31	0	964	0	995	0.79%
	U2R	1	1	0	0	50	52	0.04%
	Total	45928	67376	11655	964	50	125973	100%
	%	36.46%	53.48%	9.25%	0.77%	0.04%	100%	

Table 3. Contingency table for predictions when employing different training and test data sets.

Finally, to prove that synthetic and original data are similar, but not identical, we subtract (element-wise) the original and synthetic datasets, calling the resulting dataset as difference-dataset. If the similarity between the datasets is true, the values of the difference-dataset should have zero mean (no

reproduction bias) and a relatively small standard deviation (not too small to make both datasets indistinguishable or too large to make them completely unrelated). Table 4 shows the mean and standard deviation for 20 continuous and discrete features of the above mentioned difference-dataset. We can observe, in both cases, that the behavior is the expected one. The synthetic features exhibit variability (no exact copy) and are centered on expected values (no bias). Similarly, Figure 8 gives the distribution of values for several continuous and discrete features (upper and lower diagrams, respectively) of the difference-dataset. The Y-axis of the histograms corresponds to the number of repetitions of a given value, and the X-axis represents the values of the difference-dataset. These values are, in all cases, in the interval [-1,1], due to the [0,1] scaling and one-hot encoding of the continuous and discrete features, respectively (Section 3.1).

		Difference values Original vs. Synthetic	
	Continous features	**mean**	**std**
1	duration	-0.0027	0.1706
2	src_bytes	-0.0033	0.1289
3	dst_bytes	0.0029	0.1466
4	wrong_fragment	0.0001	0.1138
5	hot	-0.0004	0.0354
6	num_failed_logins	-0.0002	0.0101
7	logged_in	0.0014	0.4817
8	num_compromised	-0.0008	0.0248
9	root_shell	-0.0015	0.0426
10	num_root	-0.0005	0.0238
11	num_file_creations	-0.0001	0.0119
12	num_access_files	-0.0003	0.0126
13	is_guest_login	0.0009	0.1227
14	count	0.0253	0.2577
15	srv_count	0.0049	0.2522
16	serror_rate	0.0080	0.3914
17	srv_serror_rate	0.0064	0.3939
18	rerror_rate	0.0061	0.4225
19	srv_rerror_rate	0.0075	0.4256
20	same_srv_rate	-0.0052	0.3355

		Difference values Original vs. Synthetic	
	Discrete features	**mean**	**std**
97	service=red_i	0.0000	0.0080
98	service=remote_job	0.0006	0.0249
99	service=rje	0.0007	0.0261
100	service=shell	0.0003	0.0266
101	service=smtp	-0.0094	0.3352
102	service=sql_net	0.0019	0.0441
103	service=ssh	0.0020	0.0546
104	service=sunrpc	0.0020	0.0633
105	service=supdup	0.0043	0.0660
106	service=systat	0.0027	0.0694
107	service=telnet	-0.0043	0.2017
108	service=tftp_u	0.0000	0.0049
109	service=tim_i	0.0000	0.0132
110	service=time	0.0029	0.0864
111	service=urh_i	0.0000	0.0089
112	service=urp_i	-0.0006	0.1002
113	service=uucp	-0.0118	0.1532
114	service=uucp_path	0.0034	0.0868
115	service=vmnet	0.0021	0.0875
116	service=whois	0.0043	0.0813

Table 4. Mean and standard deviation of difference between values (original vs. synthetic) for several features.

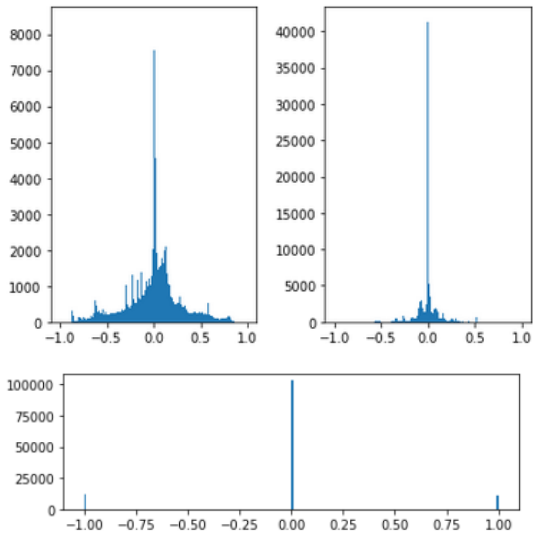

Figure 8. Distribution of value difference between original and synthetic datasets for several continuous and discrete features (upper and lower diagrams, respectively).

4.2. Improvement in classification results

The purpose of this section is to show that the new synthesized data can be used to improve classification results for common machine learning classifiers. That means that the synthetic data can be used as new training data. These results additionally confirm that the synthesized data have similar structure to the original but including enough variability to improve the performance of a classifier.

In Table 5, we present the accuracy obtained with four different classifiers: Random Forest, Logistic Regression, Linear SVM and Multilayer Perceptron (MLP); where for training data we use the NSL-KDD Training dataset alone or with additional synthesized samples, and for test data we use, in all cases, the NSL-KDD Test dataset.

It is important to see the difference between the results of Tables 2 and 5, as they try to prove two different things. Table 2 presents the results when the NSL-KDD training dataset (original or synthesized) is used both for training and prediction, differentiating only if the data set used is original or synthesized. In contrast, Table 5 shows the results when the NSL-KDD test dataset is used exclusively for prediction (without synthetic data), while a combination of original training data and synthetic data is used for the training. For each particular classifier, the six columns in Table 5 correspond to accuracy results in different scenarios: (1) using only the original NSL-KDD training dataset, (2) using the original NSL-KDD dataset repeated twice, (3) using the original NSL-KDD dataset repeated three times, (4) using the original NSL-KDD training dataset plus an additional synthetic dataset made up from the same labels as

the original one, (5) using the original NSL-KDD training dataset plus two additional synthetic datasets made up from the same labels as the original one, and (6) using the original NSL-KDD training dataset plus a synthetic dataset made up from a proportion of labels such that the final proportion of labels is balanced in the complete dataset.

	Accuracy: Random Forest						Accuracy: Linear SVM					
	Original Training dataset	Training (x2)	Training (x3)	Training + Synthetic	Training + Synthetic (x2)	Training + Synthetic (balanced)	Original Training dataset	Training (x2)	Training (x3)	Training + Synthetic	Training + Synthetic (x2)	Training + Synthetic (balanced)
Option A	0.7304	0.7303	0.7297	0.7318	0.7322	0.7319	0.7388	0.7389	0.7390	0.7324	0.7308	0.7414
Option B	0.7304	0.7303	0.7297	0.7318	0.7330	0.7338	0.7388	0.7389	0.7390	0.7414	0.7418	0.7357
Option C	0.7304	0.7303	0.7297	0.7339	0.7346	0.7361	0.7388	0.7389	0.7390	0.7549	0.7565	0.7723

	Accuracy: Logistic Regression						Accuracy: MLP					
	Original Training dataset	Training (x2)	Training (x3)	Training + Synthetic	Training + Synthetic (x2)	Training + Synthetic (balanced)	Original Training dataset	Training (x2)	Training (x3)	Training + Synthetic	Training + Synthetic (x2)	Training + Synthetic (balanced)
Option A	0.7362	0.7359	0.7370	0.7661	0.7663	0.7547	0.7752	0.7740	0.7694	0.7831	0.7865	0.7775
Option B	0.7362	0.7359	0.7370	0.7583	0.7592	0.7530	0.7752	0.7740	0.7694	0.7830	0.7950	0.7721
Option C	0.7362	0.7359	0.7370	0.7643	0.7701	0.7729	0.7752	0.7740	0.7694	0.7947	0.7925	0.7926

Table 5. Classification results when increasing the number of training samples with synthetic samples. Predictions are done with NSL-KDD Test dataset.

The values in Table 5 are color coded (same code as Table 1). We can appreciate that the increase in performance is clear when increasing the number of synthetic samples. It is also clear the difference between the first three columns and the next three, for almost all options and classifiers. This difference provides evidence that employing synthetic data increases classifiers performance, while simply repeating the original data does not provide any significative advantage. It is important to realize that this happens when using very different classifiers.

We can see in Table 5 that for Option C there is always an increase in performance for all classifiers when employing additional synthetic data. Moreover, using a balanced dataset (last columns) provides best results, at least for Option C, which is the Option we have chosen as our best model.

Finally, we compare VGM with seven SOTA synthetic over-sampling algorithms: (1) SMOTE [9], (2) SMOTE Borderline [10], (3) SMOTE+ENN [12][7] , (4) SMOTE+Tomek [12][7], (5) ADASYN [14], (6) SMOTE-SVM [11] and (7) EasyEnsemble [15][7].

Table 6 presents a comparison of several classification performance metrics: accuracy and F1 score [20], when different well known classifiers are trained with synthetic data generated by the aforementioned over-sampling algorithms. To avoid bias due to specific effectiveness of synthetic data with some particular classifier, we repeat the experiment with four classifiers: random forest, multinomial

logistic regression, linear SVM and MLP. In all cases, the classifiers are trained with a balanced dataset that is constructed using the different synthetic data generation algorithms. The base dataset used to generate the synthetic data has been the NSL-KDD Training dataset. All the prediction metrics (accuracy and F1) are obtained with the NSL-KDD Test dataset.

We can observe (Table 6) that VGM exhibits a better average performance than the other algorithms; some give better results for a specific classifier but in average VGM gives the best results. The results depend on both the classifier and the oversampling method. The intention here is to show that VGM provides average results as good as any SOTA oversampling method and in many cases better.

We used the weighted average provided by scikit-learn [31] to calculate F1 score. The values in Table 6 are color-coded in a manner similar to previous tables.

	Accuracy					F1				
	Random Forest	Logistic Regression	Linear SVM	MLP	Average Accuracy	Random Forest	Logistic Regression	Linear SVM	MLP	Average F1
VGM	0.7361	0.7729	0.7723	0.7926	0.7685	0.6908	0.7529	0.7557	0.7645	0.7410
SMOTE	0.7298	0.7717	0.7758	0.7795	0.7642	0.6840	0.7483	0.7634	0.7455	0.7353
SMOTE Borderline	0.7376	0.7597	0.7679	0.7738	0.7597	0.6920	0.7353	0.7496	0.7303	0.7268
SMOTE+ENN	0.7294	0.7533	0.7675	0.7795	0.7574	0.6819	0.7208	0.7517	0.7428	0.7243
SMOTE+Tomek	0.7324	0.7548	0.7657	0.7889	0.7605	0.6847	0.7254	0.7501	0.7573	0.7294
SMOTE SVM	0.7425	0.7629	0.7799	0.7798	0.7663	0.6993	0.7474	0.7723	0.7406	0.7399
Easy Ensemble	0.7373	0.7749	0.7721	0.7782	0.7656	0.6910	0.7528	0.7541	0.7579	0.7390
ADASYN	0.7312	0.7521	0.7512	0.7800	0.7536	0.6833	0.7236	0.7293	0.7458	0.7205

Table 6. Classification metrics when using training data generated by several over-sampling algorithms.

4.3. Model training

As part of the different experiments performed we have learned that the inclusion of regularization by using drop-out provides worse results, similarly to increasing the number of layers for the encoder and decoder beyond 2 or 3 layers. We have seen also that results are sensitive to the number of training epochs, having better results when this number is over 50. All the models converged easily.

Table 7 presents the parameters used for the training of the different models.

We have used Tensorflow to implement all the VAE models, and the python package scikit-learn to implement the different classifiers. All computations have been performed in a commercial PC (i7-4720-HQ, 16GB RAM).

	Model parameters									
	Input dimension	Output dimension	Latent space dimension	Activation function last encoder layer	Activation function last decoder layer	Activation function other layers	Batch size	# epochs	Dropout	Nodes per layer
Option A	Labels: 5 (one-hot encoding)	116 (32 continuous and 84 discrete)	25	Linear	Sigmoid (all features)	ReLU	200	80	No	[500,500,500,25,500,500,500]
Option B	Labels: 5 (one-hot encoding)	116 (32 continuous and 84 discrete)	25	Linear	Sigmoid (discrete features) and Linear (continuous features)	ReLU	200	80	No	[500,500,500,25,500,500,500]
Option C	X: 116 Labels: 5 (one-hot encoded)	116 (32 continuous and 84 discrete)	25	Linear	Sigmoid (all features)	ReLU	200	80	No	[500,500,500,25,(300+200),500,500]

Table 7. Parameters used to train the models

5. Discussion and conclusion

This work is unique in presenting the application of a VAE as a generative model for intrusion detection. The model is able to synthesize data with both continuous and categorical features. We have demonstrated that the data generated is similar to the original data, and, at the same time, have enough variability to be effective in improving the detection performance of several classifiers when used together with the original data.

Other aspect unique to this work is the ability to synthesize the new samples from the intrusion labels to which the synthetic data should belong, with the advantage of not relying on particular samples associated with the labels. This association is usually noisy and identifying a canonical set of samples associated with each label can be complex. Therefore, the proposed model streamlines the data generation process based on the intrusion label.

We have analyzed different VAE architecture variants for the proposed model, providing an extensive study on the alternatives. When considering all experiments carried out to determine the similarity of synthetic data to real intrusion detection data, and its capacity to be used as new training data we can conclude that the model based on conditional VAE with Gaussian and Bernoulli distributions presents the best results.

Also, we provide a comparison of our best model with seven common SOTA over-sampling algorithms (SMOTE, ADASYN…), showing that the synthetic data generated by our proposed model offer better metrics of average performance (accuracy, F1) when four common classifiers are trained with this data, compared to the results obtained with the data generated by the alternative algorithms. This indicates that the data generated with the proposed model is closer to the original data and can better reproduce the probability distribution of its features.

Acknowledgments

This work has been partially funded by the Ministerio de Economía y Competitividad del Gobierno de España and the Fondo de Desarrollo Regional (FEDER) within the project "Inteligencia distribuida para el control y adaptación de redes dinámicas definidas por software, Ref: TIN2014-57991-C3-2-P", in the Programa Estatal de Fomento de la Investigación Científica y Técnica de Excelencia, Subprograma Estatal de Generación de Conocimiento.

References

[1] Kingma DP, Welling M (2014) Auto-Encoding Variational Bayes. arXiv:1312.6114v10 [stat.ML]

[2] Goodfellow IJ, Pouget-Abadie J, Mirza M et al (2014) Generative Adversarial Networks. arXiv:1406.2661v1 [stat.ML]

[3] Miao Y, Yu L, Blunsom P (2015) Neural Variational Inference for Text Processing. arXiv:1511.06038 [cs.CL].

[4] Yang Z, Hu Z, Salakhutdinov R et al (2017) Improved Variational Autoencoders for Text Modeling using Dilated Convolutions. arXiv:1702.08139 [cs.NE].

[5] Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML].

[6] Galar M, Fernandez A, Barrenechea E et al (2012) 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), vol. 42, no. 4, pp. 463-484

[7] He H, Garcia EA (2009) Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 9, pp. 1263-1284.

[8] Weiss G (2004) Mining with rarity: a unifying framework. ACM SIGKDD Explorations, vol. 6. no. 1. pp. 7-19.

[9] Chawla NV, Bowyer KW, Hall LO et al (2002) SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, vol. 16. pp. 321-357.

[10] Han H, Wen-Yuan W, Bing-Huan M, (2005) Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. Advances in intelligent computing, pp. 878-887.

[11] Nguyen HM, Cooper EW, Kamei K (2011) Borderline over-sampling for imbalanced data classification. International Journal of Knowledge Engineering and Soft Data Paradigms. vol. 3. no. 1. pp. 4-21.

[12] Batista G, Prati RC, Monard MC (2004) A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsletter. vol. 6. no. 1. pp. 20-29.

[13] Cieslak DA, Chawla NV, Striegel A (2006) Combating imbalance in network intrusion datasets. IEEE International Conference on Granular Computing, pp. 732-737.

[14] He H, Bai Y, Garcia EA et al (2008) ADASYN: Adaptive synthetic sampling approach for imbalanced learning. IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp. 1322-1328.

[15] Liu XY, Wu J, Zhou ZH (2009) Exploratory Undersampling for Class-Imbalance Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 39, no. 2, pp. 539-550.

[16] Tavallaee M, Bagheri E, Lu W et al (2009) A Detailed Analysis of the KDD CUP 99 Data Set. Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2009), pages 53-58.

[17] Gregor K, Danihelka I, Graves A et al (2015) DRAW: A Recurrent Neural Network For Image Generation. arXiv:1502.04623 [cs.CV].

[18] Jang E, Gu Sh, Poole B (2016). Categorical Reparameterization with Gumbel-Softmax. arXiv:1611.01144v2 [stat.ML].

[19] An J, Cho S (2015) Variational Autoencoder based Anomaly Detection using Reconstruction Probability. SNU Data Mining Center, 2015-2 Special Lecture on IE

[20] Bhuyan MH, Bhattacharyya DK, Kalita JK (2014) Network Anomaly Detection: Methods, Systems and Tools. IEEE Communications Survey & Tutorials, vol. 16, no. 1.

[21] Sommer R, Paxson V (2010) Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. IEEE Symposium on Security and Privacy.

[22] Ingre B, Yadav A (2015) Performance Analysis of NSL-KDD dataset using ANN. 2015 International Conference on Signal Processing and Communication Engineering Systems, Guntur, pp. 92-96.

[23] Ibrahim LM, Basheer DT, Mahmod MS (2013) A comparison study for intrusion database (KDD99, NSL-KDD) based on self-organization map (SOM) artificial neural network. Journal of Engineering Science and Technology, vol. 8, no. 1 pp. 107-119, School of Engineering, Taylor's University.

[24] Hinton GE, Zemel RS (1993) Autoencoders, minimum description length and Helmholtz free energy. Proceedings of the 6th International Conference on Neural Information Processing Systems, pp. 3-10.

[25] Bengio S, Bengio Y (2000) Taking on the curse of dimensionality in joint distributions using neural networks. IEEE Transactions on Neural Networks, vol. 11, no. 3, pp. 550-557.

[26] Siracusa MR, Tieu K, Ihler AT et al (2005) Estimating dependency and significance for high-dimensional data. Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005, vol. 5, pp. v/1085-v/1088.

[27] Dhar SS, Chakraborty B, Chaudhuri P (2014) Comparison of multivariate distributions using quantile–quantile plots and related tests. arXiv:1407.1212 [math.ST].

[28] Burke MD (1977) On the multivariate two-sample problem using strong approximations of the EDF". Journal of Multivariate Analysis. 7. pp. 491–511.

[29] Justel A, Peña D, Zamar R (1997) A multivariate Kolmogorov–Smirnov test of goodness of fit. Statistics & Probability Letters, vol.35, Issue 3, pp. 251-259.

[30] Tennekes M, Jonge E, Daas PJH (2013) Visualizing and Inspecting Large Datasets with Tableplots. Journal of Data Science 11, pp. 43-58.

[31] Pedregosa F, Varoquaux G, Gramfort A et al (2011)Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, pp. 2825-2830.

Manuel Lopez-Martin is a research associate and Ph.D. candidate at Universidad de Valladolid, Spain. His research activities include the application of machine learning models to data networks. He received his M.Sc. in telecommunications engineering in 1985 from Universidad Politécnica de Madrid (UPM), Spain and his M.Sc. in computer sciences in 2013 from Universidad Autonoma de Madrid. He has worked as a data scientist at Telefonica and has more than 25 years of experience in the development of IT software projects.

Belen Carro received a Ph.D. degree in the field of broadband access networks from the Universidad de Valladolid in 2001. She is a professor and director of the Communications Systems and Networks (SRC) laboratory at Universidad de Valladolid, working as a research manager in NGN communications and services, VoIP/QoS, and machine learning. She has supervised a dozen Ph.D. students and has extensive research publications experience as author, reviewer, and editor.

Antonio Sanchez-Esguevillas received a Ph.D. degree in the field of QoS over IP networks from Universidad de Valladolid in 2004. He has managed innovation at Telefonica and has been an adjunct professor and honorary collaborator at Universidad de Valladolid, supervising several Ph.D. students. He has coordinated very large international R&D projects and has over 50 international publications and several patents. His current research interests include digital services and machine learning.