Synthetic Data Generation Using Combinatorial Testing and Variational Autoencoder

Krishna Khadka[1], Jaganmohan Chandrasekaran[2], Yu Lei[1], Raghu N. Kacker[3], D. Richard Kuhn[3]

[1]Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington,Texas 76019, USA
[2]National Security Institute, Virginia Tech, Arlington, Virginia 22203, USA
[3]Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, Maryland 20899, USA
krishna.khadka@mavs.uta.edu, jagan@vt.edu, ylei@cse.uta.edu, {raghu.kacker, d.kuhn}@nist.gov

***Abstract*— Data is a crucial component in machine learning. However, many datasets contain sensitive information such as personally identifiable health and financial data. Access to these datasets must be restricted to avoid potential security concerns. Synthetic data generation addresses this problem by generating artificial data that are similar to, and thus could be used in place of, the original real-world data. This research introduces a synthetic data generation approach called CT-VAE that uses Combinatorial Testing (CT) and Variational Autoencoder (VAE). We first use VAE to learn the distribution of the real-world data and encode it in a latent, lower-dimensional space. Next, we use CT to sample the latent space by generating a t-way set of latent vectors, each of which represents a data point in the latent space. A synthetic dataset is generated from the t-way set by decoding each latent vector in the set. Our experimental evaluation suggests that machine learning models trained with synthetic datasets generated using our approach could achieve performance that is very similar to those trained with real-world datasets. Furthermore, our approach performs better than several state-of-the-art synthetic data generation approaches.**

Keywords*— *synthetic data generation, variational autoencoders, t-way testing, combinatorial testing, latent space sampling, machine learning

I. Introduction

Machine Learning (ML) has found a wide range of applications such as image classification, natural language processing, medical diagnosis, fraud detection, and autonomous driving. Data plays a pivotal role in training and evaluating ML models. However, data may contain private or confidential information that must be protected. Synthetic data generation addresses this problem by producing artificial data that resembles real-world data. Synthetic data may be used in place of real-world data to train and evaluate ML models.

A popular kind of generative model for generating synthetic data is variational autoencoders (VAEs) [9][23]. A VAE comprises two major components: an encoder and a decoder. Given a training dataset, the encoder and decoder are trained together to learn the distribution of the real-world data represented by the training dataset and encode it in a latent, lower-dimensional space. Specifically, the encoder maps an input vector (representing a real-world data point) to a latent vector. The decoder takes as input the latent vector and tries to reconstruct the original input vector, in the sense that it tries to generate a synthetic data point that follows the distribution of the real-world data. The objective of the training process is to optimize the reconstruction error.

After a VAE is trained, the decoder could be used to generate synthetic data in the following two steps. First, we sample the latent space by generating a set of latent vectors, each of which represents a data point in the latent space. Second, we feed each sampled latent vector to the decoder, which produces a synthetic data point that follows the distribution of the real-world data. Existing work that uses a VAE to generate synthetic data typically samples the latent space in a random manner [11]. Random sampling may result in oversampling or under-sampling in certain portions of a latent space that have an irregular structure [3].

In this paper, we present a synthetic data generation approach called CT-VAE that combines the use of CT and VAE. Our approach takes as input a real-world training dataset and produces as output a synthetic dataset that when used for model training, achieves model performance similar to the original, real-world training dataset. The novelty of our approach stems from the fact that CT is employed to sample the latent space. Unlike existing work that randomly samples the latent space, our approach constructs a set of latent vectors that cover all possible t-way interactions between different dimensions in the latent space. Our hypothesis is that like software bugs, many ML model decisions only involve a small number of features, and covering all the t-way interactions in the latent space helps construct a synthetic dataset that when used for model training, can achieve comparable performance to the original dataset.

In our approach, we first preprocess the real-world dataset, e.g., one-hot encoding of categorical features, and train the VAE to learn the latent space of the real-world data. Second, we use CT to generate a t-way set of latent vectors. Before CT is applied, an input parameter model must be created. We treat each latent dimension, also referred to as a latent feature, as an input parameter, or simply a parameter. We identify representative values for each parameter by using an entropy-based discretization approach [10] to discretize the corresponding latent dimensions that are continuous. Finally,

these t-way latent vectors are used as input for the VAE's decoder network to generate a synthetic dataset.

We evaluated the effectiveness of our synthetic data generation approach by creating synthetic training datasets from several real-world datasets, including Travel Customer [22], Heloc [16], Adult Income [2], and Credit Default [27]. Each synthetic dataset is used to train four different models, including Linear Regression (LR), Decision Tree (DT), Random Forest (RF), and Support Vector Machines (SVM). There are a total of 16 experiments, one for each of the four models trained on each of the four datasets. We found that in 14 experiments, the accuracy of the model trained using a synthetic dataset was no more 5% less than the accuracy of the same model trained using the real-world counterpart. Furthermore, our approach outperformed several state-of-the-art approaches, including TVAE [26], CTGAN [26], and CopulaGAN [15]. Out of the 16 experiments, our approach achieved the highest accuracy in 10 experiments, and the second-highest accuracy in 5 experiments.

The rest of this paper is structured as follows: Section 2 provides some background information that is necessary to understand our approach. Section 3 presents the details of our approach for generating synthetic data. In Section 4, we present our experimental design and discuss the experimental results. In Section 5, we review existing work that is closely related to our work. Section 6 provides concluding remarks and outlines potential future research directions.

II. Background

Here, we present several concepts and techniques upon which our approach is built.

A. Variational Autoencoder

The VAE is a generative model that may be used to produce new data samples from a compressed representation of the data known as the latent space [8]. Because of this, VAEs can be used for a range of applications, including synthetic tabular dataset generation, picture production, and anomaly detection.

VAE is a specific type of autoencoder that adds a probabilistic aspect to the encoding process. Instead of learning a single encoding, the VAE learns a probability distribution of possible encodings. This allows VAE to generate new data samples that are similar to training data. In contrast, a regular autoencoder only learns a deterministic hidden representation which may be useful for tasks such as dimensionality reduction, removing noise, or reconstructing missing data, but may not be suitable to generate new data [3].

An encoder network and a decoder network make up a VAE. An input data sample, X, is received by the encoder network, which then, through one or more hidden layers, transfers it to a latent representation, Z. To reconstruct the input data, X, the latent representation is then sent through the decoder network involving one or more hidden layers [25]. Fig. 1 shows the VAE architecture, where the mean of the latent space Z is represented by $\mu_{z|x}$, the standard deviation is represented by $\sum_{z|x}$, and the ε is a random variable.

Formally, we can say that the encoder network learns the distribution $q_{\emptyset}(z|x)$, parameterized by ∅, whereas the decoder network learns the distribution $p_{\theta}(x|z)$, parameterized by θ.

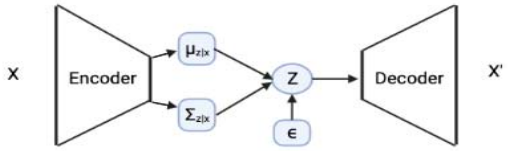

Fig. 1. Variational Autoencoder Architecture

The objective of the VAE is to maximize the evidence lower bound (ELBO) of the data's log-likelihood under the assumption that the prior distribution p(z) follows a Gaussian distribution [9]. This objective is expressed as follows:

$$\text{ELBO} = E_{q_{\emptyset}(z|x)}[\log p_{\theta}(x|z)] - \text{KLD}[q_{\emptyset}(z|x) \,||\, p(z)] \quad (1)$$

In the ELBO, the first term, $E_{q_{\emptyset}(z|x)}[\log p_{\theta}(x|z)]$ is the reconstruction likelihood term. It represents the probability of reconstructing the data x' from the latent variable z given x. The second term, $\text{KLD}[q_{\emptyset}(z|x) \,||\, p(z)]$ is the Kullback-Leibler divergence between the learned distribution $q_{\emptyset}(z|x)$ and the prior distribution $p(z)$. The KLD measures the difference between the two distributions. To maximize the ELBO, an optimization approach, like stochastic gradient descent, can be used to train both the encoder and decoder networks [26].

B. T-way Combinatorial Testing

T-way combinatorial testing is a technique for generating test cases that cover every possible combination of values of *t* parameters [13]. Each t-way combination represents a possible interaction between the *t* parameters involved in the combination. T-way combinatorial testing is frequently used in software testing and quality assurance because it can detect problems that may not be identified by conventional testing techniques.

T-way combinatorial testing requires establishing a set of parameters and a set of representative values for each parameter. Systems with many parameters benefit greatly from t-way combinatorial testing because it reduces the number of tests needed compared to testing the full parameter space, while still preserving fault detection effectiveness [30].

ACTS is a t-way test generation tool for systems that are represented as an Input Parameter Model (IPM) [29]. An IPM consists of k parameters, each with a set of representative values. Table I presents an example t-way test set produced using ACTS for a system that consists of three parameters X1, X2, and X3, each of which has two values 0 and 1.

TABLE I
AN EXAMPLE T-WAY TEST SET WHERE T=2

	X1	X2	X3
1	1	0	1
2	1	1	0
3	0	0	0
4	0	1	1

C. Entropy-Based Discretization

In entropy-based discretization, a decision tree is used to divide continuous variables into discrete bins or categories relying on the entropy of the split [20][21]. The notion of entropy is used to measure the purity of the resulting subsets after a split is made. Depending on the values of the input variables, the decision tree algorithm divides the data into progressively homogeneous groupings. The optimal split is determined by calculating the entropy of each split and selecting the split with the lowest entropy. A split is made in the decision tree until a specified criterion is satisfied. This criterion can be the height of the tree or the value of the entropy of the split. The tree continues to grow and split until one of these criteria is met [10].

For example, consider a feature with 8 values, each belonging to either class 0 or class 1 as shown in Fig 2. To determine the best-split point, we calculate the entropy for each possible split point and select the split that results in higher information gain. If we were to split the feature at the value of 4, two subsets would be created. Subset 1 would contain the values 1.15, 2.35, and 3.15, while subset 2 would contain the values 4.5, 5.5, 6.6, 7.2, and 8.2. The entropy (E) for each subset is calculated using the formula:

$$\mathrm{E} = -\mathrm{p}(\mathrm{class0}) * \log_2\big(\mathrm{p}(\mathrm{class0})\big) - \mathrm{p}(\mathrm{class1}) * \log_2\big(\mathrm{p}(\mathrm{class1})\big) \quad (2)$$

For subset 1, the entropy is calculated to be 0.918, and for subset 2, the entropy is calculated to be 0.81. The entropy before the split is calculated to be 0.811. The information gain (IG) is calculated as:

$$\mathrm{IG} = \mathrm{E}\ (\text{before split}) - \mathrm{E}\ (\text{average of entropy after split}) \quad (3)$$

The information gain for the split at 4 is 0.093. If we split at 5, subset 1 would contain 1.15, 2.35, 3.15, and 4.15, and subset 2 would contain 5.5, 6.6, 7.2, and 8.2. The entropy for subset 1 would be 1 and for subset 2 would be 0.81. The information gain is calculated as 0.01. The best split is at 4 as it results in higher information gain. The split is performed until a certain criterion is met, such as a minimum number of samples in a leaf node or a maximum depth of the tree is reached.

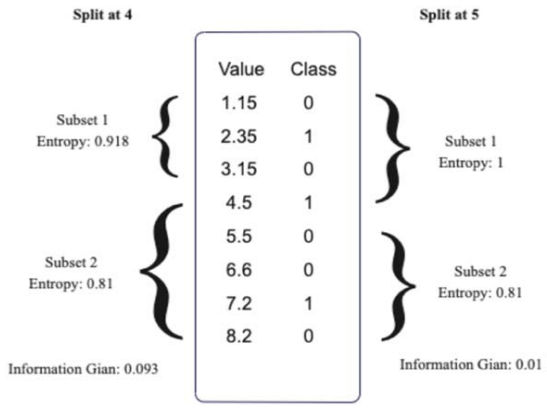

Fig 2: Finding the Best Split for Entropy-based Discretization

III. Approach

This section presents our approach of using t-way combinatorial testing (CT) in conjunction with a variational autoencoder (VAE) to generate synthetic data. Our approach takes a tabular dataset X as input and produces a synthetic dataset X' as output, which can be used as an alternative to real-world training data; the tabular dataset X includes various features that may be numerical, categorical or a combination of both. To generate synthetic data, we generate a latent space of the original data using the VAE. The latent space is a lower-dimensional representation of the original data that contains essential information needed to reconstruct the original data. From this latent space, we generate a set of t-way latent vectors using CT, which are then used as input to the decoder of the VAE to generate a set of synthetic data. The following steps outline our approach in detail.

A. Step 1. Generation of Latent Space using VAE

In step 1, the goal is to represent the tabular data with fewer feature values that capture the important properties of the original data. This can be achieved using a latent representation, which is a lower-dimensional representation of the original data. Lower dimensional representation helps to reduce the complexity of data, remove noise, and make it more efficient for data generation [9]. The selection of latent dimension for a dataset is discussed in greater detail in Section III.D.1.

To obtain this latent representation, we first train a VAE on the original data. Before training the VAE, the original data need to be preprocessed. This can involve one-hot encoding of categorical attributes and normalizing numerical attributes. With one-hot encoding, a categorical feature with n values would be transformed into n new columns, with a value of 1 or 0 in each column indicating the presence or absence of the corresponding feature value. The class label is also added as a feature, as this allows to automatically generate labels for synthetic data. The preprocessed data is then provided as input to the VAE. When the VAE is trained, the encoder network is used to generate a latent vector representation of the original data by forward propagating the preprocessed data through the encoder network.

This latent vector is typically smaller in size than the number of features in the original data. To generate synthetic data from a latent space, it is necessary to sample from the latent space. However, each latent feature is continuous and has an infinite number of possible values to choose from. To facilitate the sampling process, we discretize the latent features, which involves choosing a small number of feature values for each feature, as discussed in detail next.

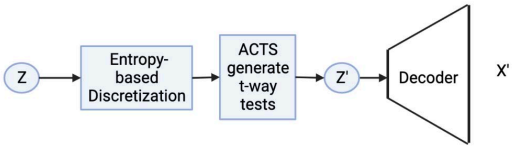

Fig 3: Generation of Synthetic Dataset using CT and VAE

B. Step 2. Generation of t-way Latent Vector

In step 2, we generate t-way test sets using CT, which are called t-way latent vectors. To do this, we begin by designing the Input Parameter Model (IPM) based on the latent vector obtained in the previous step. Each feature from the latent vector is mapped as a parameter in the IPM, and all of their feature values are mapped as their respective parameter values.

The latent vector is continuous, so to select parameter values, we discretize it using an entropy-based discretization method to identify representative values for each parameter in the IPM. This approach takes in continuous feature values from the latent vector and gives discretized feature values. It tries to find bins or splits with maximum information gain, and the number of discretized feature values depends on the depth of the tree. For example, a tree with a depth of 3 will have a maximum of 8 splits for each feature. These splits are the discrete feature values of the latent vector that are used as the feature values of IPM.

Now, we use CT to generate t-way latent vectors based on the IPM created earlier. The value of *t* determines the number of features that are considered in the interaction. A larger value of *t* results in a higher number of test instances. The CT generates t-way tests using the IPM and the specified value of *t*, resulting in the t-way latent vectors, which can be accomplished using tools such as ACTS [29].

C. Step 3. Generating Synthetic Data

In this final step, we use the t-way latent vectors generated in the previous step to generate synthetic samples of the original dataset. The decoder network of the VAE takes the t-way latent vectors as input and produces synthetic data as output. Fig. 3 presents the generation of synthetic data using CT and VAE. This synthetic data can be used for various purposes, such as training and testing ML models.

D. Hyperparameter Selection

Here we discuss the hyperparameters that are used in our approach.

1) Selecting the Number of Dimensions in the Latent Vector: The capacity of a VAE to capture the underlying structure of the data is determined by the number of dimensions in the latent vector. To select an appropriate number of latent dimensions, one can plot the reconstruction error against the latent dimensions for different datasets and observe where the reconstruction loss is the smallest. For example, Fig 4 shows how the reconstruction loss reaches its minimum in the middle, with slight spikes and fluctuations at both the beginning and end. This pattern has been observed across different datasets. As the number of latent dimensions can be large for a dataset, and testing the appropriate latent dimension value can be computationally expensive, the latent vector dimension can be chosen by starting with half of the original dimensions denoted as h and then plotting the reconstruction loss with respect to the number of latent dimensions in a certain range, e.g., [h +4, h -4]. The value that results in the smallest reconstruction error is then chosen.

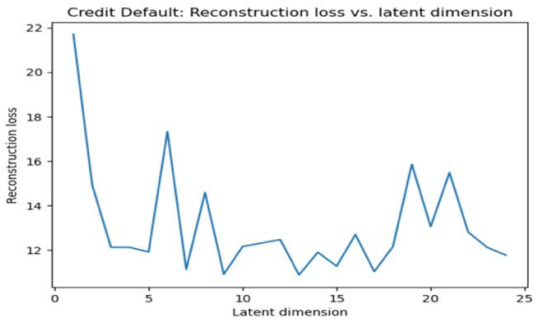

Fig. 4. Reconstruction Loss Vs Latent Dimension for Credit Default Dataset

2) Decision Tree Depth for Entropy-Based Discretization: For entropy-based discretization, the lime library can be used [14]. It takes the depth of the tree as a parameter to generate discrete values for a feature. Increasing the depth of the tree results in more discrete values for each latent feature, which can lead to the generation of more samples while being computationally expensive. By default, the depth of a tree in lime's entropy-based discretization is set to 3, resulting in a maximum of 8 bins (2^3).

3) Value of t for t-way testing: The value of *t* for t-way testing is another important hyperparameter that determines the number of test cases that are generated. The value of *t* is the strength of coverage and is typically set to a small value [13]. When selecting the value of *t*, it is recommended to start from a small value such as 3, and adjust it based on the number of test cases generated.

E. Example

To illustrate, we use a loan dataset (L) [32] and generate a synthetic dataset using our approach. The loan dataset is a binary class label dataset that includes 11 features, 10 of which are numerical, and the remaining feature is categorical. There are a total of 480 samples in the dataset, with 148 belonging to class 0 and 332 belonging to class 1. An 80/20 train-test split is used to obtain 384 instances for training and 96 instances for testing.

First, in the training set, one-hot encoding is applied to the categorical feature and the entire dataset is normalized. We then add the class labels to the normalized dataset as an additional feature to create the input for our VAE. The input dimension for the VAE after one-hot encoding and addition of class labels is 14, which includes 10 numerical features, 3 one-hot encoded features for a categorical feature with 3 distinct values, and 1 class label.

The VAE is implemented from [26] and has an encoder-decoder network architecture, with two hidden layers in each network. We selected 4 as the latent dimension for our VAE model. This was determined by starting with half of the input dimensions (14) as the initial value (h =7) and then plotting the reconstruction loss with respect to the number of latent dimensions in a range of [3, 11]. The lowest reconstruction error was achieved with a latent dimension of 4. This was sufficient to preserve the original properties of the dataset as it resulted in the least reconstruction error.

TABLE II
INPUT PARAMETER MODEL: FEATURE AND VALUES

Feature 1	Feature 2	Feature 3	Feature 4
-2.204	-2.127	-1.545	-2.391
-2.127	0.322	1.705	-1.313
-1.918	0.352	1.764	-0.516
0.326	0.379	2.378	0.795
2.711	0.415	2.601	0.828
			1.557
			2.445

TABLE III
FIRST FIVE 3-WAY LATENT VECTORS GENERATED USING ACTS

Feature 1	Feature 2	Feature 3	Feature 4
-2.204	-2.127	-1.545	-2.391
-2.204	0.322	1.705	-2.391
-2.204	0.352	1.764	-2.391
-2.204	0.379	2.378	-2.391
-2.204	0.415	2.601	-2.391

To train the VAE, we pass the normalized dataset with class labels through the network. The output of the encoder is the latent space, which consists of 384 latent vectors, each with four latent features. These feature values are then discretized using entropy-based discretization. The height of the tree for entropy-based discretization is set to 3, resulting in a varying number of discrete values for each feature, 5, 5, 5, and 7, respectively.

An IPM is created with the latent features and their discretized values, as shown in Table II. Then, the IPM is fed as input to the ACTS tool which generates 206 3-way latent vectors, where 3 is the largest t-value while keeping the number of generated instances lower than the original instances. Table III shows the first five 3-way latent vectors from a total of 206 3-way latent vectors. The 3-way latent vector is then passed through a decoder network to generate a synthetic dataset consisting of 206 instances. We trained eight classifiers, including LR, DT, RF, and SVM, using both the 384 original instances and 206 synthetic instances. We evaluated the accuracy of the models on 96 test instances, resulting in 0.83, 0.71, 0.78, and 0.84 for the original dataset and 0.78, 0.8, 0.79, and 0.77 for the synthetic dataset, respectively.

IV. Experiments

In this study, we aim to determine the effectiveness of our approach through comparison with several models on chosen datasets. To accomplish this, we divide each dataset into a training set and a test set. The training set is used to create a synthetic dataset, and the performance of this synthetic dataset is assessed using the test set. Note that the test set is from the original dataset. The accuracy of a classifier model trained using the synthetic data and tested on the test set serves as our metric for evaluating performance. We then present the results of our experiments.

A. Research Questions

- RQ1: How effective is our approach in generating synthetic datasets compared to state-of-the-art synthetic data generation approaches?
- RQ2: How does the value of *t* impact the effectiveness of our approach?

We measure the effectiveness of different approaches in terms of the accuracy of the classifier models trained on synthetic datasets but tested on real-world test datasets.

B. Dataset

To answer our research question, we conducted experiments on four different tabular datasets: Travel Customer [22], Heloc [16], Adult Income [2], and Credit Default [27]. These datasets include binary classes, each with a combination of categorical, numerical, or both types of information. Each dataset was split into train and test sets using an 80:20 stratified split, maintaining the same proportion of examples in each class as observed in the original dataset. The categorical features are transformed using one-hot encoding and the entire dataset is normalized. The class labels are also included as a feature. The input dataset for the VAE is composed of normalized features and a class label feature, which form the dimension of the input dataset. Synthetic datasets were generated using the training samples, and the performance of these synthetic datasets was evaluated on the test samples. Table IV presents information about the datasets, number of samples, training size, testing size, numerical features, categorical features, and classes.

TABLE IV
INFORMATION ON DATASETS: NUMBER OF SAMPLES, TRAIN/TEST SIZE, NUMERICAL AND CATEGORICAL FEATURES, AND CLASSES

Dataset	Samples	Training Size	Testing Size	Numerical Features	Categorical Features	Class
Travel Customer	954	763	191	2	4	2
Heloc	10459	8367	2092	23	0	2
Adult Income	48842	39073	9769	6	7	2
Credit Default	30000	24000	6000	23	0	2

C. Models

We build classifier models using four ML algorithms: Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), and Support Vector Machine (SVM). These models were applied using the scikit-learn in standard configurations [31].

In comparison to the baselines outlined in subsection D, where we train the models using synthetic samples of the same size as the original training set, our approach utilizes t-way synthetic samples with a reduced number of instances to train the models. This design allows for the examination of the effect

of varying *t* on the performance of our approach in relation to the baselines.

All the experiments were performed on a MacBook Pro powered by an 8-core Apple M1 chip, with 8GB memory.

D. Baselines

To compare with our approach, we use three synthetic data generation approaches as baselines. These approaches sample the latent space randomly.

1. TVAE (Tabular Variational Autoencoder) - a variational autoencoder-based deep learning data synthesizer implemented using Synthetic Data Vault (SDV). An instance of the TVAE model is constructed, fitted to the training set, and used to generate synthetic data [26].
2. CTGAN (Conditional GAN) - a Generative Adversarial Network (GAN)-based deep learning data synthesizer implemented using SDV. After instantiation, the CTGAN model is fitted to the training data and used to generate synthetic data [26].
3. CopulaGAN - a variation of CTGAN that uses a CDF-based transformation (applied by Gaussian Copulas) to make learning the data easier, implemented using SDV. The model is instantiated, fitted to the training data, and used to generate synthetic data [15].

E. CT-VAE Synthetic Data Generation

Our CT-VAE approach uses VAE [26] to generate a latent distribution for our training dataset. The dimension of this latent distribution is determined based on the approach outlined in Section III.D.1. The continuous latent distribution is discretized using a LIME tool [14] that relies on entropy-based discretization.

The ACTS-3.2 tool [29] is utilized to generate t-way latent vectors, with *t* ranging from 3 to 6, for each dataset. These t-way latent vectors are then decoded to produce synthetic data. To evaluate the performance of our approach against other baselines in Section IV.D, we will select synthetic data with the highest *t* value that generated a smaller number of samples than the original data.

F. Performance Metric

The accuracy score is used to evaluate the performance of the classifier models. The accuracy of a trained ML model is calculated as the ratio of the number of correct predictions made by the model to the total number of predictions.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ Predictions} \quad (2)$$

For instance, if a model makes 8 correct predictions out of 10, then its accuracy is 0.8. Note that while each model is trained on a synthetic dataset, it is tested on a real-world test dataset.

G. Results and Discussions

RQ1: We present the comparison of our approach's performance to other baseline methods. In this comparison, we evaluate the performance of classifiers trained on synthetic data instead of real-world data on a test set, to assess the ability of the synthetic data to replace real-world data in the training process. Table V compares the accuracy of various classifier models on test sets when trained using real datasets, synthetic datasets generated using different methods, and our approach. It also provides the number of training instances used for each classifier model. The number of training instances for TVAE, CopulaGAN, and CTGAN is equal to the number of training instances in the real-world dataset, while CT-VAE has significantly fewer instances. The value of *t* for CT-VAE is chosen in a way that the number of samples produced is less than the number of samples in the real dataset.

Based on the results presented, our CT-VAE approach performed well in most experiments, often achieving the best or second-best results even with fewer training samples. Out of 16 model experiments, CT-VAE performed best in 10 experiments and was second best in 5 experiments. TVAE performed best in 5 experiments and second best in 9 experiments. CTGAN performed best in 5 experiments and second best in 1 experiment. Copula GAN was the second best in 2 experiments. Here, some of the best and second-best performances were achieved by different approaches simultaneously. Our approach performed better than others in 3 out of 4 experiments on the travel customer dataset, using a smaller sample size of 313 compared to the 763 samples used by competitors. Similarly, in the Adult Income and Credit default datasets, our approach achieved the highest results in 3 out of 4 experiments with 5625 and 3979 samples, which are smaller than the 39073 and 24000 samples used by others.

We found that our approach performed better than real-world datasets in 3 experiments: DT and LR in Credit Default and DT in Heloc. This is likely because the synthetic dataset generated by the t-way approach includes data that may not be present in real-world datasets.

RQ2: We experimented to investigate the impact of *t* on the performance of our approach. We generated various synthetic datasets for various *t* values and then calculated the accuracy of the models trained on these synthetic datasets using test sets. Results in Table VI show that as *t* increases, the number of generated samples also increases.

We then trained classifiers using these synthetic data and evaluated their accuracy on the test sets. Interestingly, we found that even when using lower *t* values, the classifiers performed well on the test sets, as indicated by the high accuracy presented in the same table. For example, in the case of the Adult Income dataset, a *t* value of 4 generated 5625 samples and its performance was comparable to a *t* value of 5 which generated 40390 training samples. Similar results were found for the Credit Default dataset with the *t* values of 4 and 5. This may be because the dataset generated at a lower *t* value already captures all the patterns that are needed for making correct predictions. It also suggests that one could use lower *t* values to reduce computational cost and complexity without compromising the performance of the classifiers.

TABLE V:
MODEL PERFORMANCE OF REAL-WORLD AND SYNTHETIC DATASETS, GENERATED USING BASELINES AND CT-VAE, ON TEST SET

		Real-World		TVAE		CopulaGAN		CTGAN		CT-VAE		
		Number of instances	Accuracy	Number of instances	Accuracy	Number of instances	Accuracy	Number of instances	Accuracy	t	Number of instances	Accuracy
Travel Customer	LR	763	0.85	763	0.79	763	0.76	763	0.76	3	313	**0.82**
	DT		0.90		0.78		0.71		0.62			**0.79**
	RF		0.87		**0.83**		0.72		0.59			0.79
	SVM		0.85		0.80		0.76		0.76			**0.84**
Heloc	LR	8367	0.72	8367	0.68	8367	0.65	8367	**0.70**	4	3733	0.68
	DT		0.61		0.67		0.54		**0.68**			0.65
	RF		0.72		**0.71**		0.66		0.70			**0.71**
	SVM		0.72		**0.71**		0.64		**0.71**			0.68
Adult Income	LR	39073	0.85	39073	0.81	39073	0.82	39073	**0.84**	4	5625	0.83
	DT		0.81		0.78		0.77		0.77			**0.80**
	RF		0.85		0.80		0.81		0.80			**0.82**
	SVM		0.85		**0.82**		0.81		**0.82**			**0.82**
Credit Default	LR	24000	0.80	24000	0.80	24000	0.79	24000	0.73	4	3979	**0.804**
	DT		0.71		0.76		0.74		0.71			**0.79**
	RF		0.81		0.80		0.79		0.75			**0.805**
	SVM		0.81		**0.80**		0.79		0.75			0.797

The best accuracy is marked in bold, and the second-best accuracy is underlined.
LR= Logistic Regression, DT= Decision Tree, RF= Random Forest, SVM= Support Vector Machine Classifier

TABLE VI:
THE NUMBER OF SAMPLES GENERATED FOR DIFFERENT T-VALUES AND A SELECTED LATENT DIMENSION, AS WELL AS THE ACCURACY ON THE TEST SET FOR CLASSIFIERS TRAINED USING THESE GENERATED SAMPLES

Dataset	t	Number of Synthetic Samples	Accuracy for Logistic Regression	Accuracy for Decision Tree	Accuracy for Random Forest	Accuracy for Support Vector Classifier
Travel Customer VAE input dimension= 12 Latent dimension= 7	3	313	0.82	0.79	0.79	0.84
	4	1743	0.80	0.795	0.798	0.84
	5	7874	0.80	0.83	0.83	0.84
	6	7088	0.79	0.83	0.81	0.84
Heloc VAE input dimension = 24 Latent dimension= 7	3	532	0.68	0.63	0.67	0.68
	4	3733	0.68	0.65	0.71	0.68
	5	20676	0.68	0.64	0.70	0.695
	6	100842	0.67	0.65	0.67	0.68
Adult Income VAE input dimension = 64 Latent dimension= 11	3	697	0.82	0.79	0.81	0.815
	4	5625	0.83	0.80	0.82	0.82
	5	40390	0.82	0.79	0.82	0.82
	6	265139	0.81	0.78	0.82	0.82
Credit Default VAE input dimension = 23 Latent dimension= 9	3	567	0.80	0.78	0.80	0.79
	4	3979	0.804	0.79	0.805	0.79
	5	25469	0.804	0.79	0.805	0.79
	6	141119	0.803	0.78	0.79	0.80

V. Related Work

In the related work section, we compare our proposed approach, CT-VAE, with related works in the literature. Traditional techniques for approximating the probability distribution of tabular data include Bayesian networks based on Chow-Liu approximation [1] and statistical tools like copulas [19]. More recent methods use variational autoencoders [5][7][12][18][26] or generative adversarial networks [23][24][28]. We discuss the existing approaches for generating synthetic data in the following categories: VAE and GAN-

based approaches and approaches for exploring latent vector space.

VAE-based approaches include the use of VAEs for producing synthetic data for imbalanced learning problems [25] and SINVAD, a VAE-based technique for producing synthetic images for assessing the safety features of deep neural networks [6]. The approach in [25] generates synthetic samples based on a learned data distribution, with an emphasis on providing examples for the minority class in imbalanced datasets. We both use VAE as a generator, but the difference is that they use random sampling when generating latent vectors from the latent distribution, while we use t-way test generation to create the latent space. SINVAD generates realistic and diverse test images that can cause misclassifications in DNNs by exploring a subset of images that are semantically similar to a particular dataset. Like our approach, SINVAD also uses latent space to generate test images. However, our approach expands upon this by using t-way sampling of latent vectors to produce synthetic tabular data. Tabular data is typically organized into rows and columns, and consist of numerical or textual data, while image data is multi-dimensional and consists of pixels with specific color values.

Both [24] and [28] use GAN networks to create synthetic data, with [24] concentrating on medical data generation and [28] targeting synthetic data generation for privacy reasons. Both papers use random sampling of the latent space as input for the GAN network to generate synthetic samples, while our approach uses t-way testing to produce t-way latent vectors as input for the VAE decoder to generate synthetic samples. GANs are better suited for generating images [11], while VAEs like CT-VAE are more appropriate for generating tabular data.

Existing approaches for generating synthetic data by exploring the latent vector space include The Feature Driven VAE (FD-VAE). It is a novel type of Variational Autoencoder that allows for control over the generation of new data by partitioning the latent space according to specific features. The input data is passed through multiple encoders that each focus on a specific feature, and the output of these encoders is combined and passed through a single decoder to generate new data. An example of this would be using the FD-VAE to generate new images of faces with specific attributes such as glasses and long hair, even if the training data does not contain any examples with that specific combination of features [4]. However, our approach differs from FD-VAE, as it generates new data by covering t-way feature interaction among the latent features through t-way sampling in the latent space.

In summary, our proposed approach, CT-VAE, is a unique approach for generating synthetic tabular data for training ML models that combines VAEs with t-way combinatorial testing. By utilizing t-way testing to generate the t-way latent space of the VAE, our approach has achieved better model accuracies in our experiments. This is likely because our approach generates a more diverse and representative collection of synthetic data. While there are existing approaches for generating synthetic data using VAEs and GANs, our approach is the first to integrate VAEs with t-way combinatorial testing for generating synthetic tabular data for training ML models.

VI. Conclusion And Future Work

In this paper, we presented an approach for generating synthetic data using CT and VAE. Our approach consists of three major steps: learning the distribution of a real-world dataset using VAE, generating a t-way set of latent vectors using CT, and generating a synthetic dataset by feeding the t-way latent vectors to the decoder network of the VAE. Our approach produces a synthetic dataset that resembles the real-world data and can be used to train ML models. Our experimental evaluation shows that the use of synthetic datasets for model training can achieve model accuracy that is very close to the use of real-world datasets and our approach performs better than several state-of-the-art approaches.

Future research could go in several directions. One direction is to conduct a more thorough evaluation by applying our approach to more datasets and models. A second direction is to explore different discretization techniques to discretize the latent dimensions, such as decile-based discretization [14], frequency-based discretization [14], or chi-merge discretization [17]. Discretization identifies representative values for each parameter and could significantly impact the quality of the t-way set of latent vectors. Additionally, we could explore the potential of our approach to generating synthetic image data.

ACKNOWLEDGMENT

This work is supported by (70NANB21H092) from Information Technology Lab of National Institute of Standards and Technology (NIST).

Disclaimer: Certain software products are identified in this document. Such identification does not imply recommendation by the NIST, nor does it imply that the products identified are necessarily the best available for the purpose.

References

[1] Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory, 14(3):462–467, 1968.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive. ics.uci.edu/ml.

[3] Doersch, Carl. "Tutorial on variational autoencoders." *arXiv preprint arXiv:1606.05908* (2016).

[4] Greco, Gianluigi, Antonella Guzzo, and Giuseppe Nardiello. "FD-VAE: A feature driven VAE architecture for flexible synthetic data generation." *International Conference on Database and Expert Systems Applications*. Springer, Cham, 2020.

[5] Islam, Zubayer, and Mohamed Abdel-Aty. "Sensor-Based Transportation Mode Recognition Using Variational Autoencoder." *Journal of Big Data Analytics in Transportation* 3.1 (2021): 15-26.

[6] Kang, Sungmin, Robert Feldt, and Shin Yoo. "Sinvad: Search-based image space navigation for dnn image classifier test input generation." *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 2020.

[7] Kim, Yoon, et al. "Semi-amortized variational autoencoders." *International Conference on Machine Learning*. PMLR, 2018.

[8] Kingma, Diederik P., and Max Welling. "An introduction to variational autoencoders." *Foundations and Trends® in Machine Learning* 12.4 (2019): 307-392.

[9] Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114* (2013).

[10] Kohavi, Ron, and Mehran Sahami. "Error-based and entropy-based discretization of continuous features." *KDD*. 1996.

[11] Kornish, David, Soundararajan Ezekiel, and Maria Cornacchia. "Dcnn augmentation via synthetic data from variational autoencoders and generative adversarial networks." *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE, 2018.

[12] L Vivek Harsha Vardhan and Stanley Kok. Generating privacy-preserving synthetic tabular data using oblivious variational autoencoders. In Proceedings of the Workshop on Economics of Privacy and Data Labor at the 37 th International Conference on Machine Learning (ICML), 2020.

[13] Lei, Yu, et al. "IPOG: A general strategy for t-way software testing." *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. IEEE, 2007.

[14] Lime | discretize.py, https://github.com/marcotcr/lime/blob/fd7eb2e6f760619c29fca0187c 07b82157601b32/lime/discretize.py#L205, Accessed: 2022-01-23

[15] N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 399–410, Oct 2016. doi: 10. 1109/DSAA.2016.49.

[16] Oliabev, Averkiy. "HELOC." *Kaggle*, 31 Mar. 2021, https://www.kaggle.com/datasets/averkiyoliabev/home-equity-line-of-creditheloc.

[17] Randy, Kerber. "Chimerge: Discretization of numeric attributes." *Proceedings of the tenth national conference on Artificial intelligence*. 1992.

[18] Sajad Darabi and Yotam Elor. Synthesizing multi-modal minority samples for tabular data. arXiv preprint arXiv:2105.08204, 2021.

[19] Sanket Kamthe, Samuel Assefa, and Marc Deisenroth. Copula flows for synthetic data generation. arXiv preprint arXiv:2101.00598, 2021.

[20] Sklearn.tree.DecisionTreeClassifier – scikit-learn-1.0.2, https://scikitlearn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifie r.html, Accessed: 2022-01-22

[21] Supervised Binning, https://www.saedsayad.com/supervised_binning.htm, Accessed: 2022- 01-30

[22] Tejashvi. "Tour & Travels Customer Churn Prediction." *Kaggle*, 31 Oct. 2021, https://www.kaggle.com/datasets/tejashvi14/tour-travels-customer-churn-prediction, Accessed: 2022-01-23.

[23] Borisov, Vadim, et al. "Deep neural networks and tabular data: A survey." *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[24] Choi, Edward, et al. "Generating multi-label discrete patient records using generative adversarial networks." *Machine learning for healthcare conference*. PMLR, 2017.

[25] Wan, Zhiqiang, Yazhou Zhang, and Haibo He. "Variational autoencoder based synthetic data generation for imbalanced learning." *2017 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2017.

[26] Xu, Lei, et al. "Modeling tabular data using conditional gan." *Advances in Neural Information Processing Systems* 32 (2019).

[27] Yeh and Lien. UCI machine learning repository, 2016. URL https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

[28] Youngmin, Park, et al. "Data synthesis based on generative adversarial networks." *arXiv preprint arXiv:1806.03384* (2018).

[29] Yu, Linbin, et al. "Acts: A combinatorial test generation tool." 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. IEEE, 2013.

[30] Yu, Borazjany, Mehra N., et al. "Combinatorial testing of ACTS: A case study." *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 2012.

[31] V., Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

[32] Ergün, Burak. "Loan Data Set." *Kaggle*, 8 Nov. 2018, www.kaggle.com/datasets/burak3ergun/loan-data-set, Accessed: 2022-01-23