Clinical Medicine Journal
Vol. 7, No. 2, 2021, pp. 49-59
http://www.aiscience.org/journal/cmj
ISSN: 2381-7631 (Print); ISSN: 2381-764X (Online)

Simulation of Synthetic Diabetes Tabular Data Using Generative Adversarial Networks

Heng Wee Lin Eunice, Carol Anne Hargreaves*

Department of Statistics & Applied Probability, Faculty of Science, National University of Singapore, Singapore

Abstract

Generative Adversarial Networks (GANs) is a relatively new research avenue in the domain of Deep Learning and Artificial Intelligence. Over the past few years, GANs have been extensively researched into due to their ability to generate realistic synthetic data. The creation of synthetic tabular data is especially useful when it is desirable to avoid using the original data set due to privacy reasons. Therefore, the objective of this paper was to review the effectiveness of GANs in simulating synthetic tabular diabetes data. Methodology: Prior to GAN training, we applied min-max normalization on the features. To analyze the similarity between the real data set and each synthetic data set, we conducted exploratory data analysis before employing some statistical methods. We compared the synthesized data with the original data using data visualizations including histograms and boxplots. We also computed confidence intervals for the means of the real data variables and compared them with the confidence intervals for the means of the synthetic data. Results: The results showed that 8 of the 9 confidence intervals overlapped. We also checked whether the mean of a particular variable in the synthetic data set fell into the confidence interval of the same variable in the real data set. For each variable, we had two different probability distributions: the true distribution (from the real data); and an approximation of that distribution (from the synthetic data). To quantify the difference between the two distributions, we computed the Kull-Lieber (KL) divergence score. The KL scores for all 8 predictors were relatively small and close to 0, which is ideal. A model for classifying patients as having diabetes was built using only the real data. Another model for classifying patients as having diabetes was built using the combined real and synthetic data. The model using the combined real and synthetic data achieved a much higher accuracy of 87.0% as compared to 78.7% attained when only using the real data. Conclusion: We built a realistic synthetic data set using generative adversarial networks. The synthetic data set proved to be very similar to the real dataset and could successfully replace the real data for analysis for research purposes. Further, we verified that the availability of more training data for diabetes classification helped to improve the accuracy of the classifier, while achieving a relatively high recall.

Keywords

Generative Adversarial Networks (GANs), Deep Learning, Artificial Intelligence, Tabular Data, Synthetic Data, Generator, Discriminator, Encoder, Decoders

Received: February 4, 2021 / Accepted: March 2, 2021 / Published online: May 31, 2021

1. Introduction

Training data forms the backbone for any machine learning project. If provided with insufficient, erroneous or irrelevant data as input, even the most robust of machine learning models will not be able to deliver good results. The main purposes of creating artificial tabular data are as follows: (1) to increase the size of small training data sets, and (2) to safeguard the privacy of the original data.

A high-quality training data set should be sufficiently large and should also be representative of the entire population. Training on such a data set would allow the model to adequately capture

* Corresponding author
E-mail address: carol.hargreaves@nus.edu.sg (C. A. Hargreaves)

the underlying structure of the data, which will in turn enable the model to generalize well to unseen data. In a real-world setting, however, one often has minimal access to data required to solve a problem or to draw insights [1]. Moreover, data collection can be prohibitively costly in terms of both time and money. When limited data is available, one option would be to artificially inflate the size of the training data set by supplementing the original data set with artificial samples.

The usage of synthetic data in lieu of the original can protect the confidentiality of the latter. Some databases contain a vast amount of sensitive information. It is not advised to work directly on the database due to the risk of data breach, as well as the risk of re-identification. For example, medical records at a hospital are likely to contain a lot of personal information about the patients [2]. Even if the names and identification numbers are excluded before the data is disclosed, we cannot rule out the possibility that certain individuals may still be identified using a combination of other characteristics, such as their age, gender, height and weight. Due to the risk of re-identification, it is understandable that many regulations have been put in place to restrict the use of this kind of database [2]. One possible approach to resolving this problem would be to simply generate a sufficiently realistic synthetic data set based on the original. The former can then be used safely in place of the latter to train the model, thereby achieving a balance between privacy and utility.

In this paper, we will analyze the use of Generative Adversarial Networks (GANs) in simulating synthetic tabular data to deal with these two issues.

2. Methodology

Generating Adversarial Networks (GANs) were first introduced by Ian J. Goodfellow and his colleagues in 2014 [27]. This idea, which incited the possibility of AI-generated content, brought about much excitement in the field of Deep Learning and Artificial Intelligence (AI). Since then, much effort has been invested by researchers into developing GANs. The architecture of GANs is illustrated in Figure 1 below.

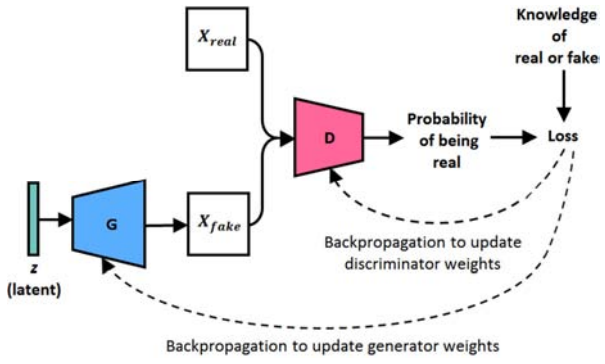

Figure 1. GAN Architecture Image modified from Alver [3].

2.1. Generative Adversarial Network (GAN) Framework

The basic concept behind GANs involves pitting two adversaries against each other in a game. One agent is the generator network. As its name suggests, the generator creates fake samples which are intended to stem from the same distribution as the training data. The other agent is the discriminator-network. Using conventional supervised learning techniques, the discriminator examines samples and learns to classify them into two groups – either real or fake. The goal of the generator is to fool the discriminator into thinking that the fake samples generated are actually real. This process is illustrated in Figure 2.

By definition, the discriminator is a differentiable function D that receives x as input and has $\theta^{(D)}$ as its parameters. In contrast, the generator is a differentiable function G that receives z as input and has $\theta^{(G)}$ as its parameters. To be more precise, each agent has a unique cost function defined in terms of its own parameters. The discriminator seeks to minimize $J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right)$ and can only do so by manipulating $\theta^{(D)}$. Likewise, the generator seeks to minimize $J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right)$ and can only do so by manipulating $\theta^{(G)}$ [4].

Even though $J^{(D)}$ depends on $\theta^{(G)}$ and $J^{(G)}$ depends on $\theta^{(D)}$, neither agent has control over its opponent's parameters. Therefore, it is more accurate to describe this situation as a game instead of an optimization problem. The solution to a game is a Nash equilibrium, which in this case, refers to local differential Nash equilibria [5]. Specifically, the solution to this game corresponds to a tuple $\left(\theta^{(D)}, \theta^{(G)}\right)$, which is a local minimum of $J^{(D)}$ with respect to $\theta^{(D)}$ and a local minimum of $J^{(G)}$ with respect to $\theta^{(G)}$[4].

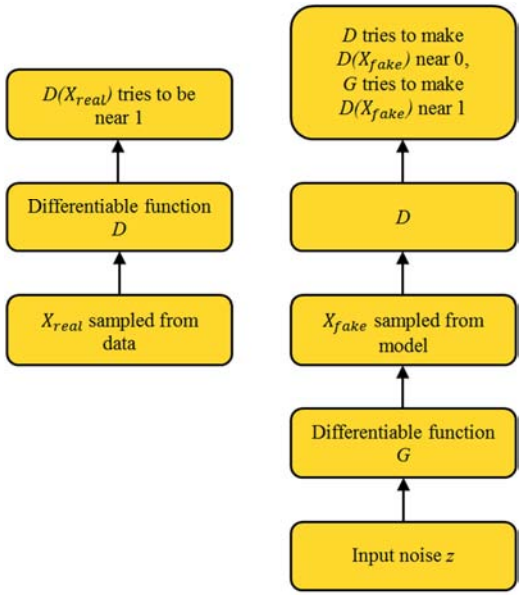

Figure 2. The GAN Framework Image modified from Goodfellow [4].

With reference to Figure 2, the discriminator and generator networks are represented by the functions D and G respectively. Under the assumption that half of the inputs provided are real and the remaining half are fake, the goal of the discriminator is to output the probability that an input is real. The game unfolds in the following two scenarios.

In the first scenario, training examples x_{real} are randomly sampled from the training set and used as input for the discriminator. In this case, the goal of the discriminator is to make $D(x_{real})$ close to 1, signifying that x_{real} is a real sample from the training set.

In the second scenario, random noise z is randomly sampled from some simple prior distribution and used as input for the generator. Here, z can be regarded as just a source of randomness that allows the generator to output many samples instead of only one realistic sample. Note that for p_{model} to have full support on the x space, the dimension of z must be at least as large as the dimension of x [4]. Correspondingly, the discriminator receives input $G(z)$, a fake sample x_{fake} produced by the generator. This time round, both players participate. The discriminator tries to make $D(G(z))$ approach 0 to signify that the input is fake. Simultaneously, the generator tries to make the same quantity approach 1.

Based on game theory, if both models have sufficient capacity, the Nash equilibrium of this game corresponds to the generator producing perfect samples $G(z)$ that come from the same distribution as the training data. At this point, the discriminator cannot actually distinguish between the two sources of data and simply says that every input has a probability one-half of being real and a probability one-half of being fake; that is, $D(x) = \frac{1}{2}$ for all x.

2.2. The Training Process

GAN's training process entails simultaneous application of stochastic gradient descent (SGD). Two mini-batches are sampled per iteration: one mini-batch of x samples from the training data and one minibatch of random noise z from a prior distribution. After which, two gradient steps are made concurrently: one involves updating $\theta^{(D)}$ to decrease $J^{(D)}$ and the other involves updating $\theta^{(G)}$ to decrease $J^{(G)}$. For either case, it is feasible to use any gradient-based optimization algorithm. Adam is often recommended as a good choice [4].

In some early GAN papers, it is not uncommon to read that the authors implemented a fixed number of updates of the discriminator for each update of the generator [6]. Heuristically, overfitting can be alleviated by limiting the number of discriminator updates per generator update [7]. As of late 2016, it is of Goodfellow et al.'s opinion that the procedure of simultaneous gradient descent works best in practice [4]. This necessitates updating the discriminator once every generator update.

Algorithm
Minibatch stochastic gradient descent training of GANs.

for number of training iterations **do**

1. *Train the discriminator.*
   - Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
   - Sample minibatch of m noise samples $\{x^{(1)}, \dots, x^{(m)}\}$ from training data distribution $p_{data}(x)$.
   - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[log\, D(x^{(i)}) + log\left(1 - D\left(G(z^{(i)})\right)\right)\right]$$

2. *Train the generator.*
   - Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
   - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} log\left(1 - D\left(G(z^{(i)})\right)\right)$$

end for
The gradient-based updates can use any standard gradient-based learning rule.

Figure 3. Algorithm for Training GANs Image modified from Goodfellow [8].

2.3. The Discriminator's Cost, $J^{(D)}$

The cost function used for the discriminator is simply the standard cross-entropy cost that is minimized while training a binary classifier: $J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -E_{x\sim p_{data}} log(D(x)) - E_{z\sim p_g} log\left(1 - D(G(z))\right)$. The sole difference is that the discriminator (or classifier) is trained on two minibatches of data instead of one. One minibatch originates from the training data, where the label is usually 1 for all samples and the other minibatch originates from the generator, where the label is 0 for all samples [4].

To derive the optimal discriminator strategy, we start by rewriting $J^{(D)}$ in its integral form: $J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\int_{-\infty}^{\infty} [p_{data}(x)\, log(D(x)) + p_{model}(x)\, log(1 - D(x))]\, dx$. Note that $p_{data}(x)$ represents the original distribution; and $p_{model}(x)$ represents the generator distribution. Our goal is to minimize $J^{(D)}$ with respect to D, assuming, that both $p_{data}(x)$ and $p_{model}(x)$ are non-zero everywhere. This is equivalent to maximizing the integrand with respect to D. Notice that the integrand can be written in the form $f(y) = a\, log(y) + b\, log(1 - y)$.

To find critical point(s), we differentiate $f(y)$ with respect to y and equate to 0: $f'(y) = \frac{a}{y} - \frac{b}{1-y} = 0 \rightarrow y^* = \frac{a}{a+b}$ where $a + b \neq 0$. We differentiate once more to get the second derivative: $f''(y^*) = -(a+b)^2 \cdot \left(\frac{1}{a} + \frac{1}{b}\right) < 0$ where $a, b \in (0, 1)$, which ascertains that $f(y)$ has a maximum point at $y = \frac{a}{a+b}$. Substituting $a = p_{data}(x)$,

$b = p_{model}(x)$ and $y = D(x)$ into y^*, we obtain the optimal discriminator strategy: $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$.

2.4. Minimax Game

A full specification of the game requires specifying cost functions for both agents. Since both agents are rivals, it seems reasonable to set the generator's cost function as the negative of the discriminator's cost function: $J^{(G)} = -J^{(D)}$. This translates to playing a zero-sum game in which the two agents' costs always sum up to zero. Since $J^{(G)}$ is directly associated with $J^{(D)}$, we can condense the entire game into a value function specified solely in terms of $J^{(D)}$; that is: $V(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$. The solution to a zero-sum game involves minimization in an outer loop and maximization in an inner loop, which gives rise to its alternative name - a minimax game: $\theta^{(G)*} = arg\min_{\theta^{(G)}} \max_{\theta^{(D)}} V(\theta^{(D)}, \theta^{(G)})$ [4].

To understand the above, consider $J^{(D)}$ from the perspective of the generator. Given a fixed generator, the discriminator learns to minimize its cross-entropy $J^{(D)}$, which is equivalent to maximizing V. In contrast, the generator strives to maximize the minimum cross-entropy attainable by the discriminator, which is equivalent to minimizing V.

2.5. Non-saturating Generator Loss

However, putting a minimax game into play is not ideal. In the scenario where the discriminator learns well and confidently rejects the generated samples, this causes the generator's gradient to vanish, which impedes further training of the generator network [4].

One method to resolve this problem is to employ cross-entropy minimization for the generator as well. Instead of simply negating the discriminator's cost, we replace the target for constructing the cross-entropy cost for the generator. [4] The generator's new cost is then: $J^{(G)} = -E_{z\sim p_g} \log D(G(z))$. Instead of minimizing the log-probability of the discriminator being correct, the generator now maximizes the log-probability of the discriminator being wrong.

Unlike the minimax game, which is theoretically motivated, this game is heuristically driven. The sole intention of this modification is to ensure that each agent (or network) will continue to have a sufficiently large gradient even when it is suffering defeat. Since the cost functions of both agents are no longer directly linked to each other, the game is no longer zero-sum and cannot be defined by a single value function.

3. Problems in Training GANs

3.1. Vanishing Gradients

Backpropagation is an extensively used algorithm in training neural networks. It works by computing the gradient of the loss function with respect to each weight using the chain rule one layer at a time. As gradient flows backward from the last layer to the first layer, it gets progressively smaller. If it becomes too small, the initial layers either learn very slowly or stop learning completely. This is termed as the vanishing gradients problem. This can be attributed to the use of "S"-shaped activation functions like *tanh* and *sigmoid* [9]. For example, when the input becomes large (either negative or positive), the *sigmoid* function saturates at 0 or 1, with the derivative becoming extremely close to zero. In other words, the gradient vanishes.

3.2. Mode Collapse

Data distributions can be highly complex and multimodal. Multimodal simply means that the data distribution has a lot of "peaks" or "modes". Each mode denotes a concentration of similar data samples, which are distinct from those of other modes. The Helvetica scenario or mode collapse is a problem that arises when the generator learns to map different input z values to the same output point [4]. In other words, in attempting to fool the discriminator, the generator produces samples that belong to a limited set of modes.

The severity of mode collapse varies from complete to partial collapse. In the case of complete collapse, all generated samples are identical, whereas in partial collapse, most of the generated samples share common properties. In both scenarios, the generator will not produce a wide variety of samples. As a result, the generated data will not be representative of the data distribution in reality, which implies that the learnt GAN is not useful at all.

4. Generative Adversarial Network (GAN) Architecture

The GANs were constructed using the *pytorch* library in Python.

Our GAN architecture has the following parameters:

LeakyReLU as the activation function with a negative slope of 0.3

Batch size = 5, 10 or 15

Learning rate = 0.0002, 0.0003, 0.0005

Batch normalization layers

Use of dropout in the discriminator with a probability of 0.2

Minibatch discrimination in the discriminator

Binary cross-entropy as the loss function

Adam as the optimizer algorithm

Two hidden layers with size 256 and 512

In the generator, the layers are ordered in ascending size

In the discriminator, the layers are ordered in descending size

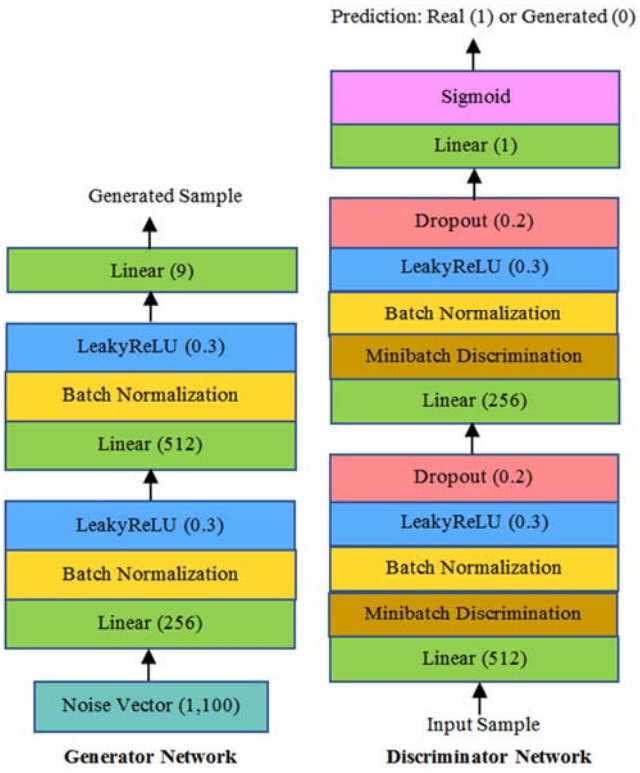

Figure 4. GAN Architecture.

Binary cross-entropy loss is used because it is most suitable to measure the performance of a model whose output is a probability between 0 and 1 [10]. There are no convolutional layers because the input is not an image. Dropout was included to prevent overfitting [11]. One-sided labelling was implemented where positive labels were smoothed to 0.9. LeakyReLU was selected as the activation function since its gradient does not saturate during backpropagation. LeakyReLU is also preferred over ReLU since it solves the "dying ReLU" problem and facilitates gradient flow throughout the architecture.

We tested nine different configurations varying the batch size and the learning rate. In this experiment, we intentionally use a simple network to focus on the basic proposal of generating synthetic numerical data sets.

4.1. Batch Normalization

Batch normalization was included to stabilize the training process [12]. It standardizes outputs from the preceding layer to have a zero mean and unit variance. This prevents the normalized input $|x|$ from approaching the outer limits of the *sigmoid* function. This ensures that the derivative is not too small and thereby mitigates against vanishing gradients. It is recommended for batch normalization layers to be included in both the discriminator and generator networks, except the output of the generator and input to the discriminator [13]. Batch normalization should occur after "S"-shaped activations (*tanh*, *sigmoid*) and before non-Gaussian activations (*ReLU*, *LeakyReLU*) [14].

4.2. Minibatch Discrimination

The concept behind 'minibatch discrimination' is to ensure that the discriminator examines multiple samples in combination instead of in isolation [15]. By doing so, mode collapse becomes easily detectable, since the discriminator will recognize that the minibatch of generated samples are fake whenever all the samples are very close to each other. The generator is then penalized and forced to output a variety of samples per minibatch generated.

Salimans et al. proposed modelling the *closeness* between samples in a minibatch by calculating the L_1-norm between the rows of matrices associated with different inputs [15]. The smaller the L_1-norm, the *closer* the inputs are. After which, this additional information is concatenated with the original input and fed into the subsequent layer of the discriminator. Effectually, the classification task of the discriminator remains the same but it is now able to use this extra material as side information to change its behavior accordingly [15].

4.3. One-sided Label Smoothing

One-sided label smoothing is a form of regularization which increases the complexity of discriminator training by providing smoothed labels to the discriminator network [4]. This means that we can assign decimal values, such as 0.9 or 0.8 to real samples, and 0.1 or 0.2 to fake samples, instead of simply labelling every example as either 1 or 0. This serves to discourage the discriminator from being overconfident about its classification.

In practice, however, it is important to only smooth the positive labels [15]. Suppose we use a target of α (in place of 1) for the real samples and a target of β (in place of 0) for the fake samples. Then the optimal discriminator function is $D^*(x) = \frac{\alpha \cdot p_{data}(x) + \beta \cdot p_{model}(x)}{p_{data}(x) + p_{model}(x)}$. When $\beta = 0$, smoothing by α simply scales down the optimal value of $D^*(x)$ without modifying the shape of the function. In contrast, when $\beta \neq 0$, the shape of the function $D^*(x)$ changes. Consider a region where $p_{data}(x)$ is small and $p_{model}(x)$ is significantly larger. $D^*(x)$ will peak near the spurious mode of $p_{model}(x)$. The

presence of $p_{model}(x)$ in the numerator is problematic because, in areas where $p_{data}(x)$ is approximately zero and $p_{model}(x)$ is large, erroneous samples from $p_{model}(x)$ have no incentive to move nearer to the data. We therefore smooth only the positive labels to α, leaving negative labels set to 0 [15].

5. Hyper Parameter Tuning

Given that the GAN is trained using SGD, one challenge involves carefully selecting the learning rate and the batch size. Using training examples, SGD estimates the error gradient for the current state of the model and updates the weights of the model via back propagation.

Learning rate refers to the amount by which the weights are updated during training. If limited time is available, it is of Bengio & Yoshua's opinion, that this is the hyper parameter worth tuning [16]. A too large value could cause premature convergence to a suboptimal solution. In contrast, a too small value could result in painfully slow convergence or cause the process to get trapped in an undesirable local minimum. For a neural network with inputs mapped to the [0,1]-interval, typical learning rate values are between 10^{-6} and 1 [16].

A large batch size provides a more accurate error gradient estimate. It is more probable that adjustment of the model weights will lead to improved performance. However, this comes at the cost of a slow rate of convergence [17]. Alternately, using a small batch size provides a less accurate estimate that is highly dependent on the specific training examples used. This noisy estimate of the error gradient would result in noisy updates to the model weights. Nonetheless, this noisy update could help prevent premature convergence and the increased model update frequency could also result in faster learning [18].

Varying the learning rate together with the batch size affects the learning process in different ways. Hence, we will experiment using nine combinations of hyper parameter values to train the GANs.

6. Data Preparation

The Pima Indians Diabetes data set was retrieved from the UCI Machine Learning Repository and can also be found on the Kaggle website [19]. All patients are females of Pima Indian heritage and are at least 21 years old. The data set consists of 9 variables and has a total of 768 observations/rows.

The definition of the variables are as follows:

Pregnancies: Number of times pregnant.

Glucose: The blood plasma glucose concentration after a 2-hour oral glucose tolerance test.

Blood Pressure: Diastolic blood pressure (mm/Hg).

Skin Thickness: Triceps skin fold thickness (mm).

Insulin: 2-Hour serum insulin (mu U/ml).

BMI: Body mass index (kg/m squared).

Diabetes Pedigree Function: A function that determines the risk of type 2 diabetes based on family history; the larger the function, the higher the risk of type 2 diabetes.

Age: Age (years).

Outcome: Whether the person is diagnosed with type 2 diabetes ($0 = no, 1 = yes$).

6.1. Dealing with Missing Values

The original data set contains missing values denoted by 0. Variables with zeros include Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin and BMI. Except for Pregnancies, it is not normal for the other variables to take on the value 0. SkinThickness (29.6%) and Insulin (48.7%), in particular, contain large numbers of missing values.

Since patients with the same diagnosis are likely to be more similar to each other, our original idea was to split the data set based on Outcome before dealing separately with the missing values. We realised that the median Insulin level for diabetic patients is 0 because there are more missing values (51.5%) than non-missing values (48.5%). To resolve this, we considered imputing missing values with the median of non-missing values. However, simply replacing more than half of the variable with the same value may drastically alter its distribution. To ensure that the training data is reliable, we decided to drop observations with missing values. Among the 374 observations with missing Insulin values, most contain missing values for other variable(s) as well. The cleaned data set has 392 observations.

6.2. Accuracy Checks

We divided the cleaned data set based on Outcome and obtained the range of values for each variable. We then verified the medically acceptable minimum and maximum values with a licensed medical doctor. Two variables - BloodPressure and Insulin, stand out.

5 patients with unusually low diastolic blood pressure of 40 mm/Hg and below were removed. High insulin levels are indicative of insulin resistance, which is associated with Type 2 diabetes. While there is no medical literature on the cut-off for the maximum possible insulin value, the normal medically acceptable range for a non-diabetic is between 16 – 166 mu U/ml. We noticed 27 non-diabetic patients with insulin levels

more than 250 mu U/ml. In our opinion, the data seems contradictory so we decided to remove these 27 observations. The resulting data set has 360 observations. There are 232 patients with no diabetes and 128 with diabetes.

6.3. Feature Scaling

Prior to GAN training, we applied min-max normalization on the features. This linear transformation technique preserves the relationship among the original data values while mapping each feature to fall within the [0,1]-interval [20]. Homogenising the range for all the features prevents those with large variances from dominating others [20]. It also helps to cut down the range of values that the generator network generates as well [10].

7. Selection of the Best Artificial Data Set

The real data set referred to in this section is the data set used for training the GANs. The artificial data sets refer to those generated by the learnt GANs. For fair comparison, all artificial data sets generated contain the same number of 360 observations as the real data set. To analyze the similarity between the real data set and each artificial data set, we conducted exploratory data analysis before employing some statistical methods. The procedure is as follows:

Histograms and boxplots were first plotted to visualize the distribution of the variables.

The 95% Confidence Interval (CI) for the mean of each variable in both the real and artificial data sets was computed.

Hypothesis Testing was conducted using a two-sample Z-test. Our Null Hypothesis H_0 is the mean of a particular variable in the real data set is equal to the mean of the same variable in the artificial data set. A total of 9 hypothesis tests were conducted.

The best artificial data set was chosen based on: (1) the number of variables in which the mean of the variable in the artificial data set falls into the CI of the same variable in the real data set; and (2) the number of variables in which we do not reject H_0.

8. Performance of Best Artificial Data Set

This data set was generated by a GAN trained with learning rate - 0.0003 and batch size - 10.

8.1. Exploratory Data Analysis

Histograms belonging to the original data set are shaded green, while those belonging to the synthetic data set are shaded blue. For each variable, the greater the degree of overlap between the two histograms, the closer the real and synthetic variables are in their distributions.

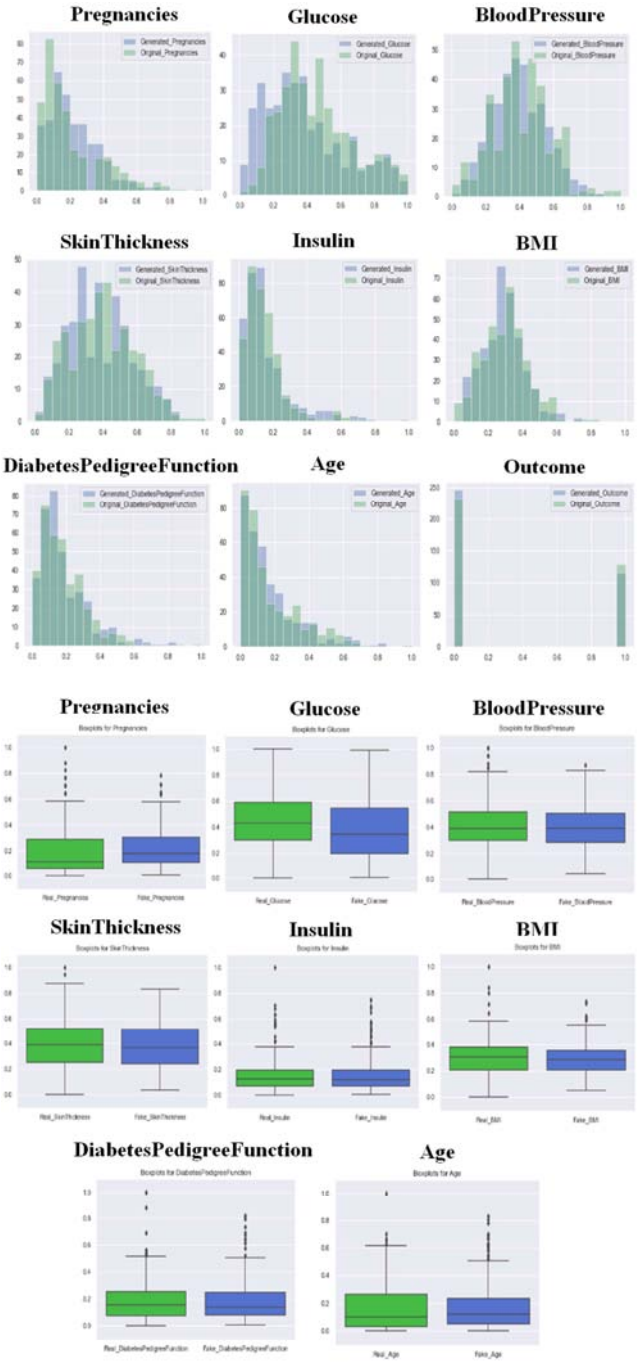

Figure 5. Visual Comparison of Real and Synthetic Variables.

8.2. Statistical Methods

8.2.1. Hypothesis Testing

$$H_0: \mu_1 = \mu_2 \text{ vs } H_1: \mu_1 \neq \mu_2$$

Our Null Hypothesis H_0 is that the mean of a particular variable in the real data set, μ_1, is equal to the mean of the

same variable in the artificial data set, μ_2. If p-value < 0.05, we reject H_0 and conclude that the two means are different.

Ideally, we want the two means to be the same; hence, we do not want to reject H_0 for as many variables as possible.

Table 1. Statistical Significance Test Results.

Variable	p-value	Do not reject Null Hypothesis
Pregnancies	0.152257	TRUE
Glucose	0.000062	FALSE
BloodPressure	0.26678	TRUE
SkinThickness	0.280832	TRUE
Insulin	0.560453	TRUE
BMI	0.259814	TRUE
DiabetesPedigreeFunction	0.788961	TRUE
Age	0.78002	TRUE
Outcome	0.269629	TRUE

Our significance tests support that our synthetic data generated are similar to the real data, as 9 out of the 10 synthetic variables (90%) are similar to the real variables.

8.2.2. Confidence Intervals

According to the Central Limit Theorem (CLT), given a sufficiently large sample size n, the distribution of the sample mean for a variable, $\bar{x}$, will approximate a normal distribution regardless of that variable's distribution in the population. Formal statement of the CLT: If $\bar{x}$ is the mean of a random sample $\{X_1, X_2, \ldots, X_n\}$ of size n from a distribution with a finite mean μ and a finite positive variance σ^2, then the distribution of $Z = \frac{\bar{x}-\mu}{\frac{\sigma}{\sqrt{n}}} \sim N(0,1)$ as $n \to \infty$. This means that the variable $\bar{x}$ follows a $N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$ distribution [21].

For a population with unknown mean μ and known standard deviation σ, a 95% CI for μ, based on a simple random sample of size n, is: $\bar{x} \pm \left(1.96 \times \frac{\sigma}{\sqrt{n}}\right)$. Since σ is unknown, we estimate it using the sample standard deviation $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$. [22] The 95% CI estimate for the population mean is then: $\bar{x} \pm \left(1.96 \times \frac{s}{\sqrt{n}}\right)$.

We computed the 95% CI for the mean of variables in the real data set and the 95% CI for the mean of variables in the synthetic data set. For each variable, we check whether the two CIs overlap. All variables CI for the Real data and synthetic data overlap and perform well except for the Glucose variable.

Table 2. Confidence Intervals for Real and Synthetic Variables.

	Real Lower Bound	Real Upper Bound	Synthetic Lower Bound	Synthetic Upper Bound	Overlap of CIs
Pregnancies	0.179988	0.219685	0.202973	0.2331	TRUE
Glucose	0.437466	0.48237	0.365505	0.416082	FALSE
Blood Pressure	0.392797	0.429594	0.38099	0.413559	TRUE
Skin Thickness	0.377259	0.415995	0.362789	0.400658	TRUE
Insulin	0.139785	0.16527	0.144053	0.172303	TRUE
BMI	0.285334	0.314109	0.27652	0.301148	TRUE
Diabetes Pedigree Function	0.169487	0.197846	0.171524	0.201438	TRUE
Age	0.150553	0.185836	0.154974	0.188334	TRUE
Outcome	0.306038	0.405073	0.268547	0.364787	TRUE

Next, we checked whether the mean of a particular variable in the synthetic data set falls into the CI of the same variable in the real data set. Unsurprisingly, the means of all synthetic variables fall into the CI for the mean of the real variable, except for the Glucose variable.

Table 3. Real Lower Bound and Upper Bound Comparison for the Synthetic Mean.

	Real Lower Bound	Real Upper Bound	Synthetic Mean	Synthetic Mean in Real CI
Pregnancies	0.179988	0.219685	0.218036	TRUE
Glucose	0.437466	0.48237	0.390794	FALSE
BloodPressure	0.392797	0.429594	0.397274	TRUE
SkinThickness	0.377259	0.415995	0.381724	TRUE
Insulin	0.139785	0.16527	0.158178	TRUE
BMI	0.285334	0.314109	0.288834	TRUE
DiabetesPedigreeFunction	0.169487	0.197846	0.186481	TRUE
Age	0.150553	0.185836	0.171654	TRUE
Outcome	0.306038	0.405073	0.316667	TRUE

8.2.3. Kullback-Leibler (KL) Divergence

For each variable, we have two different probability distributions: the true distribution (from the real data); and an approximation of that distribution (from the synthetic data). To quantify the difference between the two, we compute the KL divergence score. Mathematically, the KL divergence is the expectation of the logarithmic difference between the probability of data in the real data distribution P with the approximating synthetic data distribution Q: $KL(P\|Q) =$

$E[\log p(x) - \log q(x)]$ [23].

Table 4. KL Divergence Score.

Variable	KL Divergence Score
Pregnancies	0.73907
Glucose	0.37357
BloodPressure	0.201723
SkinThickness	0.24717
Insulin	0.62436
BMI	0.20132
DiabetesPedigreeFunction	0.58388
Age	1.08929

The intuition behind this score is simple. Consider P's divergence from Q. When the probability of an event from P is large and the probability of the same event in Q is small, the divergence is large. When the probability from P is small and the probability from Q is large, the divergence is also large, but is smaller as compared to the first case. The score takes on values between 0 and $+\infty$. A score of 0 indicates that P and Q match perfectly. For each predictor variable, we compute the KL divergence score between its true and approximated distribution. The KL divergence scores for all 8 predictors are relatively small and are close to 0, which is ideal.

8.3. Synthetic Minority Oversampling Technique (SMOTE)

Traditional machine learning classification algorithms often exhibit unsatisfactory performance on imbalanced data sets. This is because the minority class has minimal effect on the overall accuracy. For example, if a data set has an extremely imbalanced class distribution of 95:5, the useless classifier will be 95% accurate simply by predicting all samples as the majority class. The predictive performance of such a classifier is especially deceiving since samples from the minority class are completely missed. This poses a significant problem since the minority class is usually the class of interest and therefore, the more important class.

SMOTE balances the classes by synthesizing artificial samples [24]. It randomly selects an existing point from the minority class and locates its k nearest neighbours of the same class. For each of these pairs, a new point is generated in the vector between them. In this study, we use the implementation from the *imbalanced-learn* python library [25].

8.4. Random Forest (RF) Classifier

The real data set was combined with the artificial data set and shuffled to yield a third data set.

For each of the 3 data sets, the following process was conducted:

The data was first split into 70% for training and 30% for testing.

SMOTE was used to balance the classes in the training set.

A RF classifier was fitted to the balanced training set and used to predict on the test set.

8.4.1. Metrics

Our primary goal is to achieve at least 75% prediction accuracy. Apart from achieving a high accuracy rate, we also hope to reduce false negatives in our diagnosis so as to pick up potential patients as soon as possible. This is extremely important as we are dealing with patients' lives, and we should aim to reach out to them in the shortest amount of time possible. Hence, we hope to obtain high sensitivity, without compromising on specificity.

If we were to use a test with low specificity for screening, many patients without diabetes will be classified as having diabetes. These patients would potentially receive unnecessary diagnostic procedures or further treatment. By raising specificity there will be fewer of such false positive results.

Table 5. Accuracy Measures for Imbalanced and Balanced Synthetic, Real and Combined Data.

	Imbalanced Synthetic Data	Balanced Synthetic Data (using SMOTE)
Accuracy	0.85185	0.85185
Sensitivity	0.62857	0.80000
Specificity	0.95890	0.87671
Precision	0.88000	0.75676
Recall	0.62857	0.80000
F1-score	0.73333	0.77778
AUC	0.79374	0.83836

	Imbalanced Real Data	Balanced Real Data (using SMOTE)
Accuracy	0.80556	0.78704
Sensitivity	0.74359	0.84615
Specificity	0.84058	0.75362
Precision	0.72500	0.66000
Recall	0.74359	0.84615
F1-score	0.73418	0.74157
AUC	0.79208	0.79989

	Imbalanced Combined Data	Balanced Combined Data (using SMOTE)
Accuracy	0.84722	0.87037
Sensitivity	0.71831	0.81690
Specificity	0.91034	0.89655
Precision	0.79688	0.79452
Recall	0.71831	0.81690
F1-score	0.75556	0.80556
AUC	0.81433	0.85673

For the RF classifiers fitted to all three data sets, we notice an increase in sensitivity and Area Under Curve (AUC) score after balancing the training data using SMOTE. To verify whether the availability of more training data helps improve the performance of classification algorithms, we will focus specifically on the results of the RF classifiers fitted to the

balanced real training data and the balanced combined training data. The latter achieves a much higher accuracy of 87.0% as compared to 78.7% attained by the former. The latter also achieves a higher AUC score of 0.857, which implies that the model is now better at distinguishing between the positive class and negative class. Even though the latter has slightly lower sensitivity of 81.7%, this value is relatively high and is still comparable to the 84.6% attained by the former.

8.4.2. Feature Importance

We compare the feature importance for the RF classifiers fitted to all 3 data sets using the implementation from the *scikit-learn* python library. In this case, feature importance is calculated as the reduction in node impurity (Gini index or entropy) weighted by the probability of reaching that node [26]. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value, the more important the feature.

With reference to the table below, the features in each data set are ordered in descending order of feature importance. Ideally, the top few features picked out for the combined data set should be the same few identified in the real data set. We can see from Table 6, that the 5 most important features for the Real and Combined data is the same, while the order of importance is slightly different.

Table 6. Features of Importance for Synthetic, Real and Combined Data.

Synthetic Data	Real Data	Combined Data
Insulin	Glucose	Glucose
BMI	Insulin	Insulin
Glucose	Age	BMI
Age	BMI	DiabetesPedigreeFunction
SkinThickness	DiabetesPedigreeFunction	Age
DiabetesPedigreeFunction	SkinThickness	Pregnancies
BloodPressure	Pregnancies	SkinThickness
Pregnancies	BloodPressure	BloodPressure

9. Conclusion

In this paper, we studied common problems experienced in training GANs. To address these issues, we incorporated various techniques; namely, one-sided label smoothing, minibatch discrimination and batch normalization into the network architecture of the Vanilla GAN model. Different combinations of learning rates and batch sizes were used for GAN training. To determine which combination works best, we evaluated the similarity between the real and the synthetic data generated. This meets our first objective - to generate a sufficiently realistic synthetic data set that can be used in place of the original real data.

Our next step involved using SMOTE to balance the classes in the real data set and the combined data set. We analyzed the performance of a RF classifier trained separately on the balanced real data set and on the balanced combined data set. We verified that the availability of more training data helps improve the accuracy of the classifier, while achieving relatively high sensitivity. This meets our second objective - to inflate the size of the training data so as to enhance the performance of machine learning classifiers.

As an initial inquiry, this research limits itself to one GAN architecture and one data set. For future work, we will explore different network architectures, as well as data sets with diverse characteristics. We are particularly interested in investigating the effect of the number of classes and features in the data set on the final results.

References

[1] 7 Effective Ways to Deal with a Small Dataset | Hacker Noon. https://hackernoon.com/7-effective-ways-to-deal-with-a-small-dataset-2gyl407s. Accessed 10 Jan. 2021.

[2] Maynard-Atem, L (2019). The Data Series – Solving the Data Privacy Problem Using Synthetic Data, Impact, 2019:2, 11-13, DOI: 10.1080/2058802X.2019.1668192

[3] Alver, S. (2018, September 04). Connections Between GANs and AC Methods in Reinforcement Learning. Retrieved February 02, 2021, from https://alversafa.github.io/blog/2018/09/04/gan-ac.html

[4] Goodfellow, Ian. "NIPS 2016 Tutorial: Generative Adversarial Networks." ArXiv: 1701.00160 [Cs], Apr. 2017. arXiv.org, http://arxiv.org/abs/1701.00160.

[5] Ratliff, L. J., Burden, S. A., and Sastry, S. S. (2013). Characterization and computation of local nash equilibria in continuous games. In Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on, pages 917–924. IEEE.

[6] Arjovsky, M., Chintala, S. & Bottou, L. (2017). Wasserstein generative adversarial networks. In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of Proceedings of Machine Learning Research, pages 214–223. PMLR, 2017.

[7] Thanh-Tung, Hoang, et al. “Improving Generalization and Stability of Generative Adversarial Networks.” ArXiv: 1902.03984 [Cs, Stat], Feb. 2019. arXiv.org, http://arxiv.org/abs/1902.03984.

[8] Goodfellow, Ian & Pouget-Abadie, Jean & Mirza, Mehdi & Xu, Bing & Warde-Farley, David & Ozair, Sherjil & Courville, Aaron & Bengio, Y.. (2014). Generative Adversarial Networks. Advances in Neural Information Processing Systems. 3. 10.1145/3422622.

[9] Szandała, T. (2020). Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. Bio-inspired Neurocomputing Studies in Computational Intelligence, 203-224. doi:10.1007/978-981-15-5495-7_11

[10] Tanaka, F. H., & Aranha, C. (2019). Data Augmentation Using GANs. arXiv: 1904.09135.

[11] Srivastava, Nitish & Hinton, Geoffrey & Krizhevsky, Alex & Sutskever, Ilya & Salakhutdinov, Ruslan. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 15. 1929-1958.

[12] Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning, in PMLR 37:448-456

[13] Radford, Alec, et al. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.” ArXiv:1511.06434 [Cs], Jan. 2016. arXiv.org, http://arxiv.org/abs/1511.06434.

[14] J. Brownlee, A Gentle Introduction to Batch Normalization for Deep Neural Networks (2019), Maching Learning Mastery

[15] Salimans, Tim, et al. “Improved Techniques for Training GANs.” ArXiv: 1606.03498 [Cs], June 2016. arXiv.org, http://arxiv.org/abs/1606.03498.

[16] Bengio, Yoshua. “Practical Recommendations for Gradient-Based Training of Deep Architectures.” ArXiv: 1206.5533 [Cs], Sept. 2012. arXiv.org, http://arxiv.org/abs/1206.5533.

[17] Brownlee, Jason. “How to Control the Stability of Training Neural Networks with the Batch Size.” Machine Learning Mastery, 20 Jan. 2019, https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/.

[18] Brownlee, Jason. “A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size.” Machine Learning Mastery, 20 July 2017, https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[19] Pima Indians Diabetes Database. https://kaggle.com/uciml/pima-indians-diabetes-database. Accessed 11 Jan. 2021.

[20] Han, Jiawei, et al. “Data Preprocessing.” Data Mining, Elsevier, 2012, pp. 83–124. DOI.org (Crossref), doi:10.1016/B978-0-12-381479-1.00003-4.

[21] Sample Means. (n.d.). Retrieved January 18, 2021, from http://www.stat.yale.edu/Courses/1997-98/101/sampmn.htm#clt

[22] Estimation of a population mean. (n.d.). Retrieved February 17, 2021, from https://www.britannica.com/science/statistics/Estimation-of-a-population-mean

[23] Kurt, W. (2017, May 10). Kullback-Leibler Divergence Explained. Retrieved January 16, 2021, from https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained

[24] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, Volume 16, pages 321-357, 2002, 2011. doi: 10.1613/jair.953.

[25] Guillaume Lemaˆıtre, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. Journal of Machine Learning Research, 18(17):1–5, 2017. URL http://jmlr.org/papers/v18/ 16-365.html.

[26] Ronaghan, S. (2019, November 01). The Mathematics of Decision Trees, Random Forest and Feature Importance in Scikit-learn and Spark. Retrieved January 24, 2021, from https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3

[27] Ian Goodfellow et al. “Generative adversarial nets”. In: Advances in neural information processing systems. 2014, pp. 2672–2680.