Report

Summer Internship – 2022

Synthetic Tabular Data Generation using Generative Adversarial Network

PREPARED BY:

PARTH KANERIYA (202112084)

RENISH GADHIYA (202112037)

GUIDED BY:

PROF. AMIT MANKODI

INDEX

1. Introduction

Tabular data is one of the most common and important data in today's world. Big amounts of data, including census results, health care records, and web logs are all stored in tabular format. Such data is valuable because it contains useful patterns that can help in decision-making. As more and more companies and research institutes rely on data to make decisions, people have recognized the need to enable good decision-making and ensure privacy protection, as well as manage other issues. It is in this context that the demand for synthetic data arises.

Tabular data is widely used in different fields and has become an integral part of predicting potential needs. Data can be used to predict the risk of disease and provide people with life and medical advice. Data can also help governments and companies make decisions. The growth of the field is exponential, as the availability of massive data inspires people to explore different applications.

However, due to the quality, quantity, and privacy issues associated with using real data, people usually do not stick to the original data when creating and exploring these applications in various domains.

Using tabular data to train a recommender system involves data quality, data imbalance, and data privacy issues. Data quality can be improved by applying various filtering criteria to the data. Data imbalance can be addressed by partitioning the data and learning of local models. Addressing the privacy issue is more challenging.

1.1 Project Objective :

Our project objective is to generate synthetic data which resembles original dataset using different GAN models and to compare and analyzed different GAN models.

1.2 Project Scope :

Our project aims to generate synthetic data which can be used to train machine learning algorithms with bigger dataset. It is more accurate and scalable replacement of real world data with assured privacy.

2. Preliminaries

2.1 Synthetic Data

Synthetic data is information that's artificially manufactured rather than generated by real world events. Synthetic data can be created in three ways. One, by using some perturbation of real data. Two, by combining attributes from the real data and three, by generating samples from some distribution. All three methods have advantages and disadvantages. With the rise of big data and deep learning, private data has become a one of the most valuable types of data, and the promise of realistic synthetic data has so far been a holy grail. This is often because the utility and privacy of data are inversely proportional when it comes to machine learning models. The field of data synthesis with deep learning techniques is still in its infancy, and we expect to see promising results in the near future, as well as standardization of methods and evaluations.

2.2 Applications of Synthetic Data :

High-quality synthetic data has important applications.

Data understanding: Learning the distribution of tabular data can help us understand the underlying structure and association between columns.

Data compression: Synthetic data generators can be used to store tabular data efficiently and compactly. A small generative neural network can be easily stored on portable devices to generate an infinite number of rows.

Data augmentation: A generative model can generate (more) training data, or reasonably perturb the original data, which can improve the performance of downstream predictive models.

Data disclosure: Data privacy is an important issue today. Using synthetic data instead of real data can avoid the disclosure of private information while still allowing data to be used for various applications.

Synthetic data serve various purposes, including machine learning. Neural network can replicate the data distribution that you have feed. From that neural network model learn the statistical properties of real data to produce synthetic data that mimic the original data.

2.3 Machine Learning

Machine learning is the capacity of machines to learn a specific task without explicit instructions, relying on patterns instead. Machine learning is considered a subset of the term Artificial Intelligence. There are two types of machine learning:

1) Supervised learning
2) Unsupervised learning
3) Reinforcement learning

Supervised Learning

Supervised learning is done with labeled data, meaning that for each data point it is known to which class that data point belongs. This is used, for example, when performing classification and regression. Training is done by feeding the model the image and telling it how wrong the predicted class was. To achieve this ability, the data and classes need to be transformed to a format interpretable by a machine learning model. Often, this means the data and classes are transformed to numbers. For example, to find whether the picture has cat or dog, the class cat is transforming into a 0 while the class dog become a 1.

Unsupervised Learning

Unsupervised learning is a type of algorithm that learns patterns from untagged data. The biggest difference between the supervise and unsupervised learning is the absence of classes. This means that this method is not suitable for classification or regression, but rather has applications in dimensionality reduction and clustering. In some pattern recognition problems, the training data consists of a set of input vectors x without any corresponding target values. The goal in such unsupervised learning problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine how the data is distributed in the space, known as density estimation. To put forward in simpler terms, for a n-sampled space x_1 to x_n, true class labels are not provided for each sample, hence known as learning without teacher.

Reinforcement learning

Reinforcement learning can be considered a self learning model, where the model performs actions in a certain environment and receives feedback from the environment. To oversimplify it, this is similar to how humans learn. When a child gets a reward for doing the right things but is being punished for doing the wrong things, it slowly learns. The same principle is applied in reinforcement learning. Reinforcement learning has seen a lot of usage in playing games with machine learning, like chess and go. In both cases, reinforcement learning was one of the fundamental blocks that allowed these models to perform so well.

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. And neural network is mostly used in generative adversarial network which is used for generating synthetic data.

2.4 Deep Learning

The term deep learning commonly refers to neural networks with several or many hidden layers. These neural networks attempt to simulate the behavior of the human brain far from matching its ability allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.

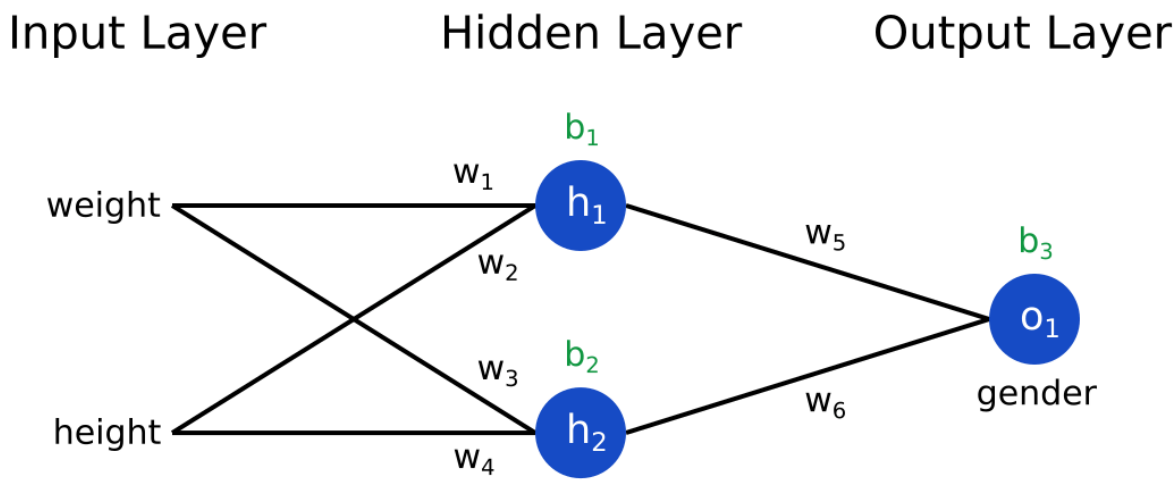
Deep learning neural networks, or artificial neural networks, attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data.

Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called *visible* layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

2.5 Neural Network

Neural networks are inspired by the workings of biological neurons from the brain. These neurons get a certain input and produce a certain output depending on the input. A neural network combines many layers of these neurons to create complex structures, which is able to learn advanced non-linear functions. The weights of the neuron's input and the threshold of the neuron are learned during training and are also called the parameters of the model.

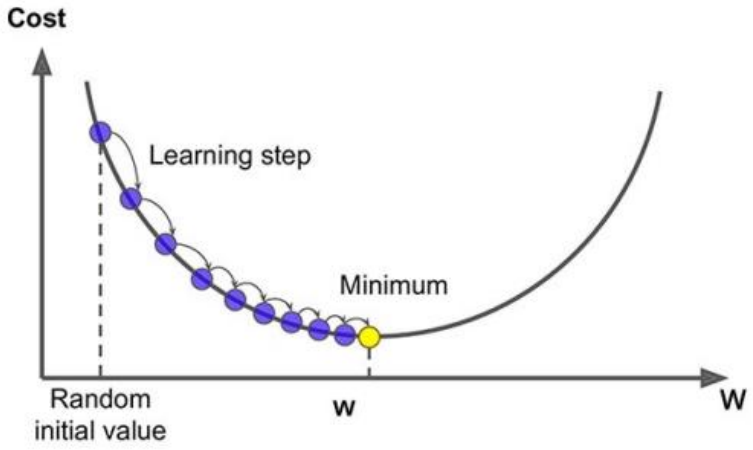
Another process called back propagation uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and back propagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate.



Simple Neural Network

Back propagation and Gradient Descent

One of the major building blocks of neural networks is their capacity to learn many parameters. This capability is enabled by back propagation and gradient descent. A neural network is trained in two steps. First, some data is put into the network and produces a result. Second, it is measured how wrong the answer was (know as loss or error) and the weights of the network are updated accordingly. The weight updates based on the loss or error. Gradient descent enables a network to slowly converge to a minimum of the loss by iteratively calculating the gradients of the weights with respect to the loss and updating the network parameters. When the loss reaches its minimum, the network cannot improve with training anymore. The gradients are calculated with back propagation, which is essentially the chain rule, applied from the output to the input of the network. By doing this many times of data, the network learns how to perform the task at hand.



Above figure shows example of the gradient descent algorithm. The parameter w is updated by calculating the gradient with respect to the loss and multiplying it with a learning step. This is done until convergence, i.e., when w is at the minimum.

There are different types of neural networks to address specific problems or datasets. For example,

Convolutional neural network (CNNs) used primarily in computer vision and image classification applications, can detect features and patterns within an image, enabling tasks, like object detection or recognition.

Recurrent neural networks (RNNs) are identified by their feedback loops. These learning algorithms are primarily leveraged when using time-series data to make predictions about future outcomes, such as stock market predictions or sales forecasting.

When a neural network with many hidden layers trained to approximate complicated and high dimensional probability distribution using large number of sample then it is known as Deep Generative Model.

2.6 Deep Generative Model

The field of deep generative models is even younger than deep learning, with the inception of the Variational Autoencoder (VAE) and Generative Adversarial Network (GAN) in 2013 and 2014, respectively. Deep generative models are characterized by multi-layer neural network that are able to generate samples following the distribution of the data.

Existing generation-based methods

Statistical methods and deep learning methods are used to learn the distribution of real data so that synthetic data can be generated by sampling. Deep learning methods make up another major category of data synthesizers. The motivation for building a high-quality deep neural network for tabular data comes from the success of such models on computer vision and natural language processing. Deep generative models like variational autoencoders (VAEs) and generative adversarial networks (GANs) have two new capabilities: First, the capacity to learn a complicated high-dimensional probability distribution, and second, the ability to draw high-quality samples from images or text. These capabilities have enabled various important applications in vision and language. It is also possible to build similarly high-quality models to generate synthetic tabular data - the implicit joint distribution of columns can be learned from real data, and synthetic rows can then be sampled from that distribution. A few models have been proposed that work by directly applying fully connected networks or convolutional neural networks on tabular data without considering the specific case of modeling tabular data. These models can perform well on datasets.

Generative Adversarial Networks

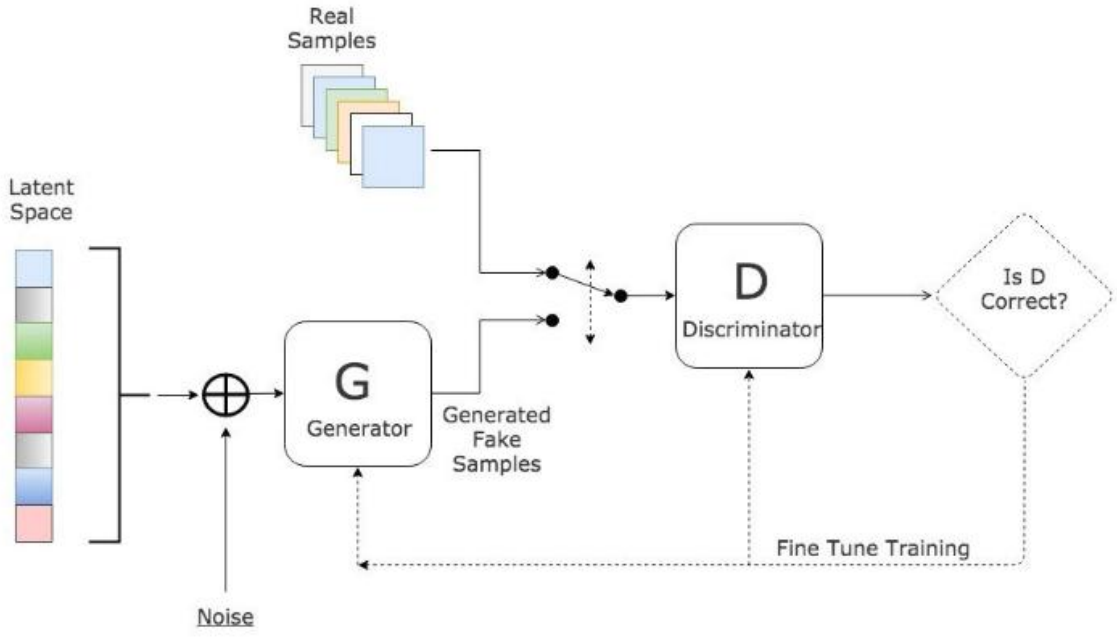
Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyse, capture and copy the variations within a dataset.

GANs can be broken down into three parts:

1) Generative: To learn a generative model, which describe how data is generated in terms of probabilistic model.
2) Adversarial: The training of a model is done in an adversarial setting.
3) Networks: Use deep neural networks as the artificial intelligence algorithms for training purpose.

In GANs, there is a **generator** and a **discriminator**. The Generator generates fake samples of data (be it an image, audio, tabular data etc.) and tries to fool the Discriminator. The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both Neural Networks and they both run in competition with each other in the training phase. The steps are repeated several times and in this, the Generator and Discriminator get better and better in their respective jobs after each repetition.

The working can be visualized by the diagram given below:



Here, the generative model captures the distribution of data and is trained in such a manner that it tries to maximize the probability of the Discriminator in making a mistake. The

Discriminator, on the other hand, is based on a model that estimates the probability that the sample that it got is received from the training data and not from the Generator. The GANs are formulated as a minimax game, where the Discriminator is trying to minimize its reward **V(D, G)** and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss. It can be mathematically described by the formula below:

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

where,
G = Generator
D = Discriminator
pdata(x) = distribution of real data
p(z) = distribution of generator
x = sample from pdata(x)
z = sample from p(z)
D(x) = Discriminator network
G(z) = Generator network

So, basically, training a GAN has two parts:

- **Part 1:** The Discriminator is trained while the Generator is idle. In this phase, the network is only forward propagated and no back-propagation is done. The Discriminator is trained on real data for n epochs, and see if it can correctly predict them as real. Also, in this phase, the Discriminator is also trained on the fake generated data from the Generator and see if it can correctly predict them as fake.
- **Part 2:** The Generator is trained while the Discriminator is idle. After the Discriminator is trained by the generated fake data of the Generator, we can get its predictions and use the results for training the Generator and get better from the previous state to try and fool the Discriminator.

The above method is repeated for a few epochs and then manually check the fake data if it seems genuine. If it seems acceptable, then the training is stopped, otherwise, it's allowed to continue for few more epochs.

In short, A GAN is a type of neural network that is able to generate new data from scratch. You can feed it a little bit of random noise as input, and it can produce realistic Data.

3. Experimental Setup

3.1 Datasets :

We have a datasets containing information of different systems specifications. Dataset have information like CPU clock speed, cache memory size (l1, l2 and l3 size), memory size and types etc. We removed some features which are unnecessary for our experiments. Our dataset consist total nine number of features after pre processing on datasets. And datasets have total 1987 rows of system data. It contains some features as numerical data and some as a categorical data.

3.2 Types of Data :

Categorical Data :

Categorical Data is data that can't be measured or counted in the form of numbers. These types of data are sorted by category, not by number. That's why it is also known as Categorical Data. These data consist of audio, images, symbols, or text. The gender of a person, i.e., male, female, or others, is categorical data. Categorical data can take numerical values. For example, maybe we would use 1 for the colour red and 2 for blue. But these numbers don't have a mathematical meaning. That is, we can't add them together or take the average.

Numerical Data :

Numerical data is any data where data points are exact numbers. Statisticians also might call numerical data, quantitative data. This data has meaning as a measurement such as house prices, salary etc. Numerical data can be characterized by continuous or discrete data. Continuous data can assume any value within a range whereas discrete data has distinct values.

Some features like CPU clock we can say that it is continuous data. Features like memory types, memory speed is example of categorical data.

3.3 Pre-processing of categorical Data :

Encoding of categorical Data :

Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the different models. Encoding categorical data is one of such tasks which is considered crucial. As we know, most of the data in real life come with categorical string values and most of the machine learning models work with integer values only and some with other different values which can be understandable

for the model. All models basically perform mathematical operations which can be performed using different tools and techniques. But the harsh truth is that mathematics is totally dependent on numbers. So in short we can say most of the models require numbers as the data, not strings or not anything else and these numbers can be float or integer.

Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the models to give and improve the predictions.

3.4 Types of Encoding :

Label Encoding :

In this encoding, each category is assigned a value from 1 through N (where N is the number of categories for the feature). One major issue with this approach is there is no relation or order between these classes, but the algorithm might consider them as some order or some relationship.

One Hot Encoding :

In this method, we map each category to a vector that contains 1 and 0, denoting the presence or absence of the feature. The number of vectors depends on the number of categories for features. This method produces many columns that slow down the learning significantly if the number of the category is very high for the feature.

We have used label encoding on our dataset because in our dataset we have large number of categories so that one hot encoding leads to high memory consumption as it produce more columns for each categories.

4. Implementation Details

4.1 Tools and Technologies :

Tools :

- Google Colab
- Python 3.7.13

Libraries :

- Tensorflow 2
- Scikit-learn
- Numpy
- Pandas
- Matplotlib
- Seaborn
- SDV (Synthetic Data Vault)
- ydata-synthetic
- Keras
- table_evalutor

We have implemented following GAN models on datasets.

(1) Vanilla GAN

(2) CTGAN

(3) DATGAN

(4) TGAN

(5) WGAN with Gradient Penalty

4.2 Vanilla GAN

We have implemented Vanilla GAN using Tensorflow and Keras APIs. Before feeding data into GAN we have preprocessed data. We have dropped some features because they have unique values so it will not make sense to generate. After removing one more string type columns we have only features having numerical data.

After features selection we scaled data between 0 and 1. So that, neural network can perform better. After that we have build generator with four layers with LeakyReLu(α = 0.2) activation layers and batch normalizations(momentum=0.8). Last Output layer have sigmoid as activation function. Same way we have build discriminator model with five layers. We have use ADAM as optimizer. We feed random noise with normal distribution into generator and train GAN.

We have generated synthetic data with random noise and inverse transform data into real format so that we can evaluate data. For evaluation of performance of GAN we have compared normal distribution, standard deviations and mean of both real and fake data. We have also compared correlation matrix and first two components of PCA of both data.

```
[ ]  def build_generator(n_columns, latent_dim):
         model = Sequential()
         model.add(Dense(32, kernel_initializer = "he_uniform", input_dim=latent_dim))
         model.add(LeakyReLU(0.2))
         model.add(BatchNormalization(momentum=0.8))
         model.add(Dense(64,  kernel_initializer = "he_uniform"))
         model.add(LeakyReLU(0.2))
         model.add(BatchNormalization(momentum=0.8))
         model.add(Dense(128,  kernel_initializer = "he_uniform"))
         model.add(LeakyReLU(0.2))
         model.add(BatchNormalization(momentum=0.8))
         model.add(Dense(n_columns, activation = "sigmoid"))
         return model

[ ]  latent_dim = 100
     generator = build_generator(data_rescaled.shape[1], latent_dim)

[ ]  optimizer = Adam(lr=0.0002, beta_1=0.5)

[ ]  def build_discriminator(inputs_n):
       model = Sequential()
       model.add(Dense(128,  kernel_initializer = "he_uniform", input_dim = inputs_n))
       model.add(LeakyReLU(0.2))
       model.add(Dense(64,  kernel_initializer = "he_uniform"))
       model.add(LeakyReLU(0.2))
       model.add(Dense(32,  kernel_initializer = "he_uniform"))
       model.add(LeakyReLU(0.2))
       model.add(Dense(16,  kernel_initializer = "he_uniform"))
       model.add(LeakyReLU(0.2))
       model.add(Dense(1, activation = "sigmoid"))
       model.compile(loss = "binary_crossentropy", optimizer = optimizer, metrics = ["accuracy"])
       return model

[ ]  discriminator = build_discriminator(data_rescaled.shape[1])
```

Code Snapshot

4.3 CTGAN

We have used SDV(Synthetic Data Vault) library for implementation of CTGAN. We have done preprocessing of data on datasets. Two columns have String data type in our dataset. So that, we have applied label encoding on that two columns. After label encoding we have all data into numeric format.

We have fit dataset on CTGAN Synthesizer and generated 1000 sample from it. We have done visual evaluation on that generated synthetic data.

4.4 DATGAN

The DATGAN is a synthesizer for tabular data. It uses LSTM cells to generate synthetic data for continuous and categorical variable types. For training DATGAN we have to pass data information to model. So that we have set 'CPU_clock' as continuous features and rest of all features as a categorical features.

We have feeding data to model with batch size of 100 and epochs of 200. We have generated samples from trained model and have done visual evaluation of it.

4.5 TGAN

We have experimented here while implementation of TGAN. We have thought that 'CPU_clock' as categorical features. Because 'CPU_clock' may be continuous but it has some fixed values so we have experimented with that.

We fitted dataset into TGAN model with 10 epochs and 10000 steps per epoch. We have generated some synthetic sample from trained model and have done visual evaluations.

4.6 WGAN with Gradient Penalty

We have used ydata-synthetic library for implementation of WGAN with GP. Here, we have keep 'cpu_clock' as a numerical features and rest of features as categorical features. We have set up batch size of 200, noise latent dimensions as 256 and 1001 epochs. We have generated some samples and have done visual evaluations.

5. Results

We have evaluated our GAN performance based on following evaluation methods.

Column wise mean and standard deviation :

This is not an advanced metric and does not reveal any hidden relations, but functions as a quick sanity check. The means and standard deviations of each column are plotted on a log scale. If the plotted values follow the diagonal, the data has comparable means and standard deviations. Furthermore, the general results do appear to have a correlation with how well these values follow the diagonal, meaning that if many points deviate from the diagonal, it is likely the cumulative distributions and quantitative measures will follow a similar pattern of deviations.

Cumulative Sum :

To visually inspect the similarity between the distributions per column, we plot the cumulative sum of each column for both the real and the fake data on top of each other. This gives one a quite thorough understanding of a column with just one plot, and works for both categorical and continuous columns. Note that this plot does not give any insights into the relations between columns, giving it limited representational power for the whole table. It does allow one to determine which kind of values and columns are easier or more difficult than others.

Column Correlations :

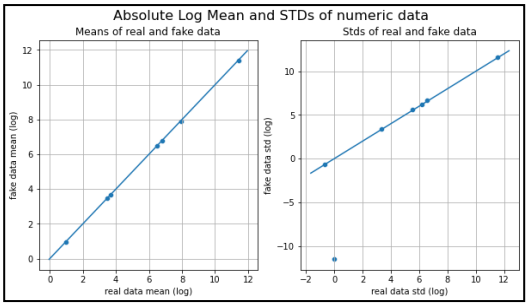
The third visualization shows one an association table for the both the real and synthetic data. It gives a clear understanding of what columns have associations with each other, and shows where the synthetic data diverges, indicating struggles that the model had with modeling this relationship.

Principle Component Analysis :

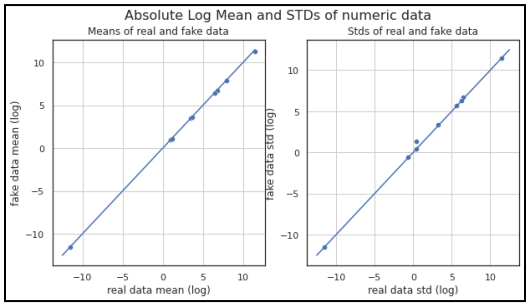
Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process. It tries to convert the data onto principal components, which are linearly uncorrelated vectors. These are then sorted by the amount of variance captured in that component, of which we keep the top k values. If k is smaller than the original number of dimensions, the dimensionality is reduced.

Column wise mean and standard deviation :

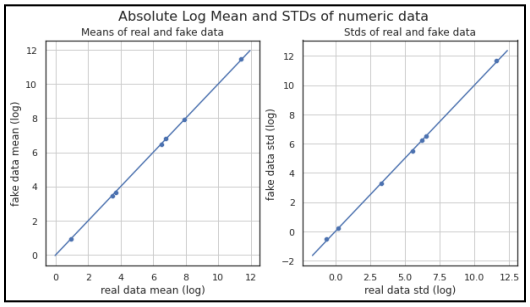
Starting with the first evaluation, we take a look at the mean and standard deviations of the real and fake dataset. We plot the log transformed values of all the numeric columns. We observe that four of the five synthesizers capture these properties with relative ease. WGAN with GP (fig. d), however, already has a hard time reproducing these values, which is not a great sign. Most models are able to capture the mean and standard deviations, but have quite a hard time capturing the correct mean and standard deviation for WGAN with GP (fig. d).



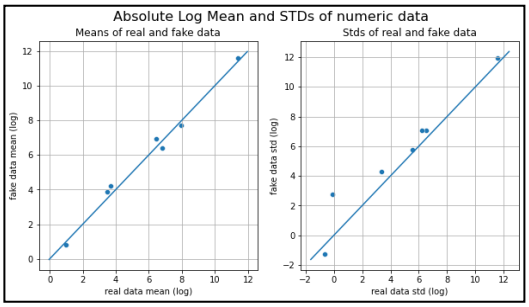
(a) TGAN



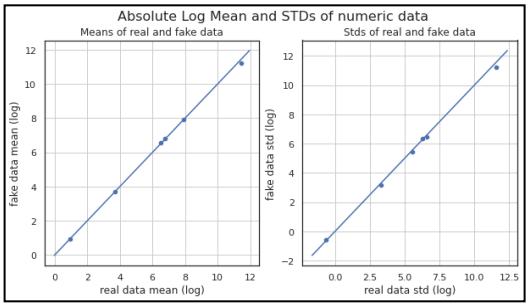
(b) CTGAN



(c) DATGAN



(d) WGAN with GP

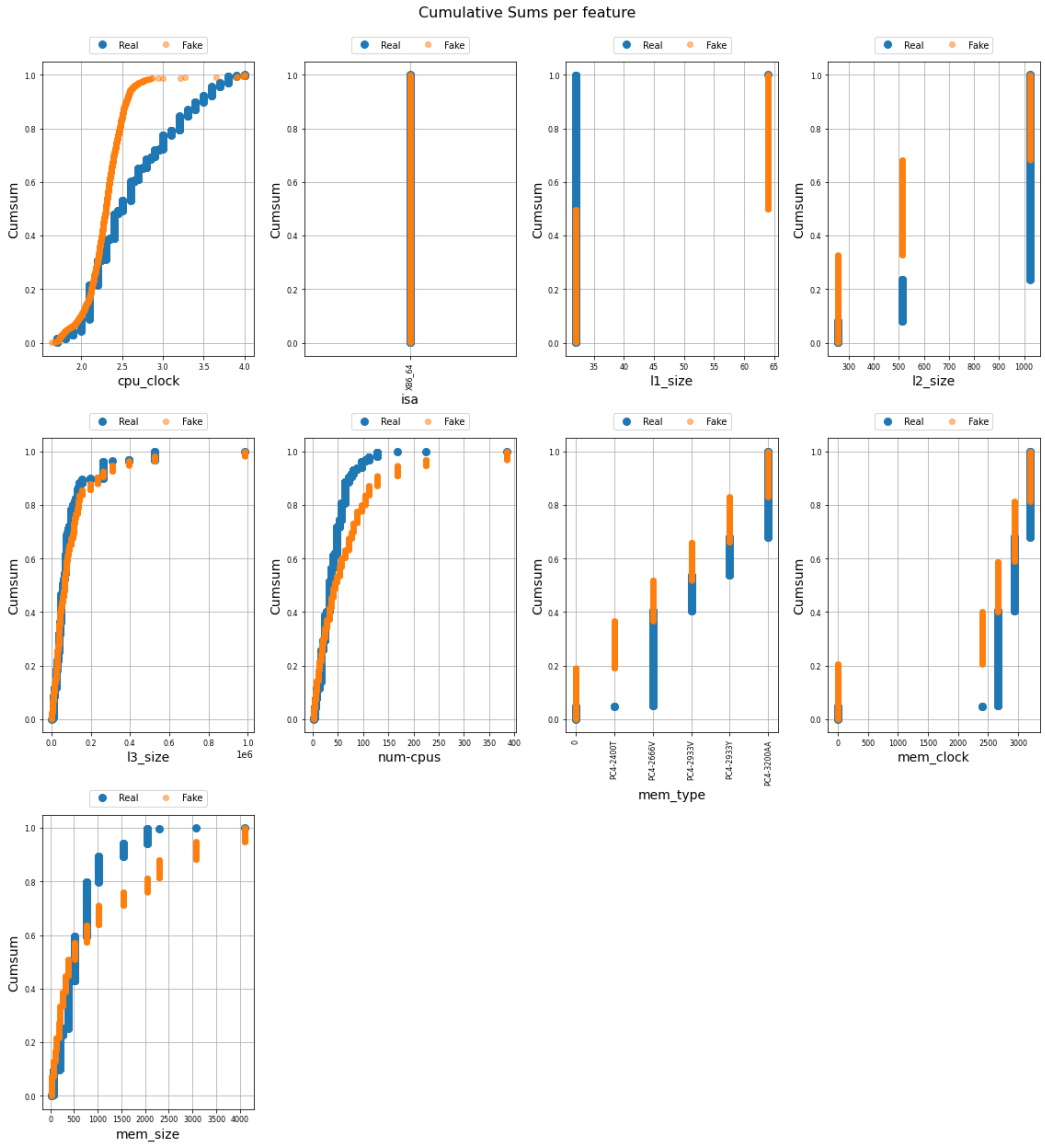


(e) GAN

Mean and Standard deviations of each column of the real and synthetic dataset. All values are log transformed.

Cumulative Sum

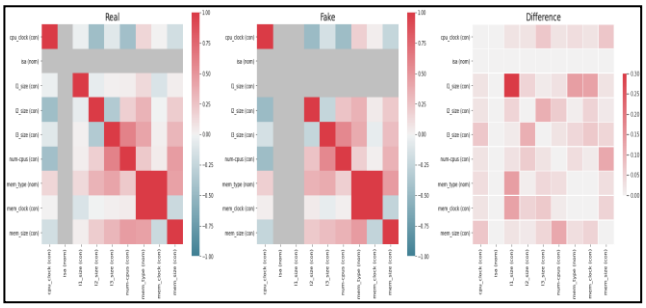
We have a look on cumulative sums per features for all the GAN models and find out that WGAN with GP have comparatively bad distributions per features then other GANs. Other GAN models have almost similar distributions. We can also observe that our custom GAN has very smooth curve for cumulative sums per features which indicates that generated fake data produced with vide range of variety. In some GANs like TGAN and WGAN with GP 'cpu_clock' features have values in small ranges which may be due to problem of mode collapse.



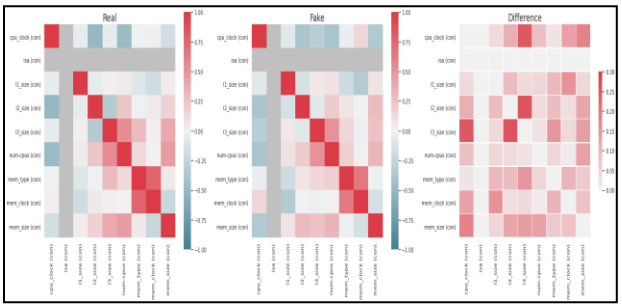
Cumulative sums per features (WGAN with GP)

Column Correlations

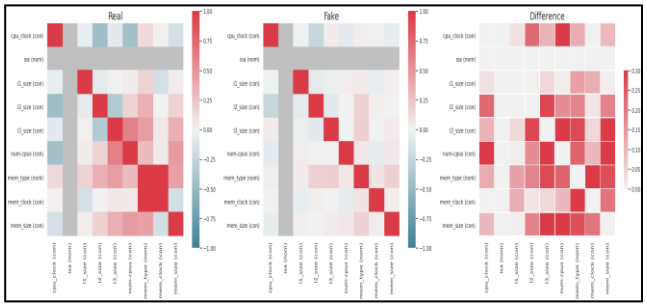
Figure shows the column wise correlations of the dataset for different GANs. First matrix indicates correlations of real data and second for generated data. Third matrix represents absolute difference between correlations of real and generated dataset. It becomes clear that TGAN (fig. a) and CTGAN (fig. b) have better column correlations among all GANs and perform best. Also, we see the performance of WGAN with GP (fig. d) and DATGAN (fig. c) are very poor, being essentially random. So that, we can say WGAN with GP and DATGAN failed to model relationship between features while TGAN and CTGAN mostly able to do it nearly.



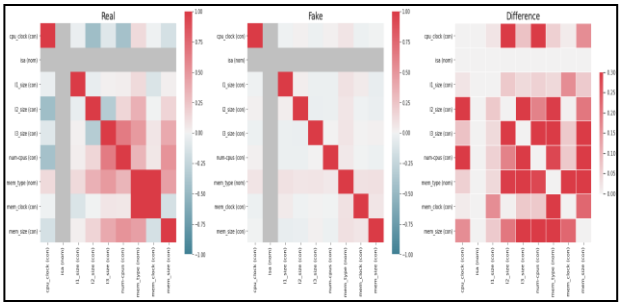
(a) TGAN



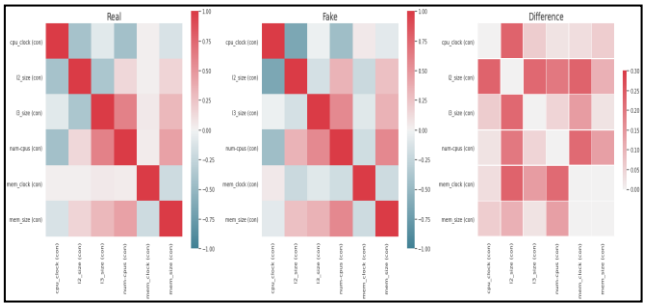
(b) CTGAN



(c) DATGAN



(d) WGAN with GP

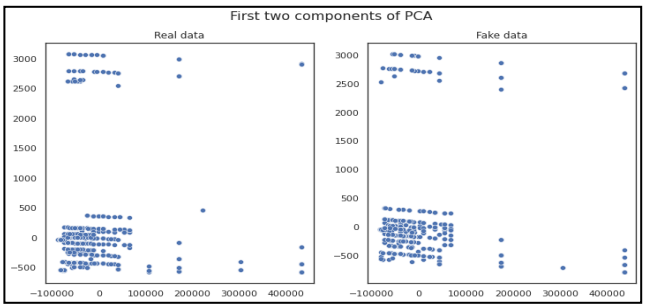


(e) GAN

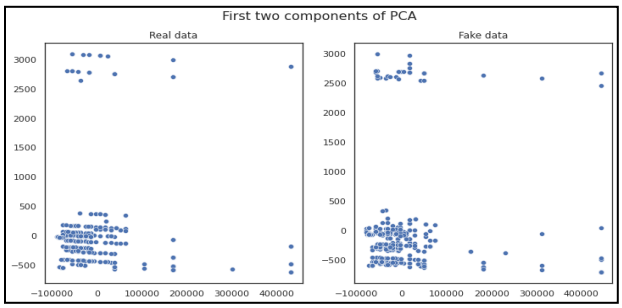
Column Correlations of each column for all GAN models

Principle Component Analysis

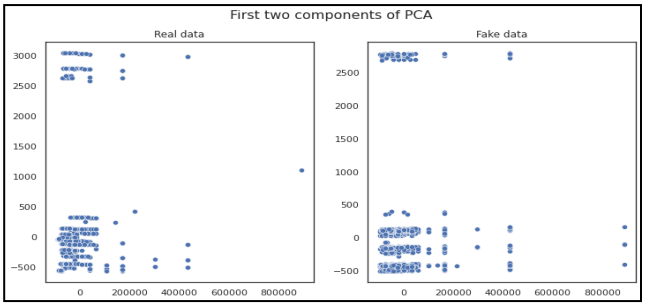
From the first two components of PCA analysis of all GANs models, we can say that TGAN (fig. a) and CTGAN (fig. b) perform better than any other GANs. WGAN with GP (fig. d) and our Vanilla GAN (fig. e) have very poor results of first two components of PCA.



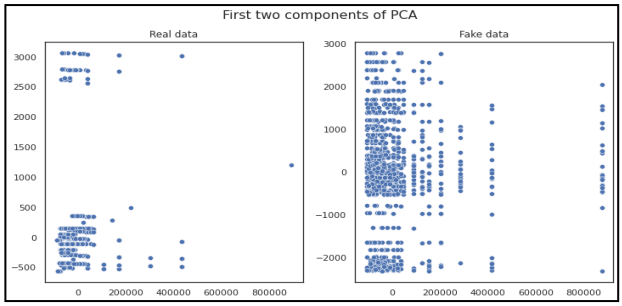
(a) TGAN



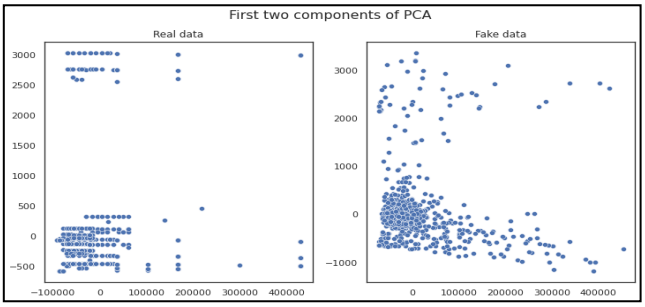
(b) CTGAN



(c) DATGAN



(d) WGAN with GP



(e) GAN

First two components of PCA for Real and Generated Dataset of all GANs

6. Conclusion

From our work on generation of synthetic tabular data with help of different GANs, We conclude that TGAN and CTGAN perform better compare to other GANs in all aspects. Our custom implemented Vanilla GAN has better results than WGAN with GP and DATGAN. During implementation, we have only considered numerical features so in future work performance of Vanilla GAN can be improved with categorical features. We can also improve performance of other GANs with optimization of hyper parameters and train GANs with larger dataset if possible. We can also explore other possibilities to generate synthetic data with Variational autoencoder and other possible algorithms.