Shahjalal University of Science and Technology
Department of Computer Science and Engineering

Diffusion Model for Oversampling Class Imbalanced Tabular Data

Rafat Ashraf Joy

Reg. No.: 2017331074

4^{th} year, 2^{nd} Semester

Iqramul Islam

Reg. No.: 2017331054

4^{th} year, 2^{nd} Semester

Department of Computer Science and Engineering

Supervisor

Mohammad Shahidur Rahman

Professor

Department of Computer Science and Engineering

26th February, 2023

Diffusion Model for Oversampling Class Imbalanced Tabular Data

A Thesis submitted to the Department of Computer Science and Engineering, Shahjalal University of Science and Technology, in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering.

By

Rafat Ashraf Joy

Reg. No.: 2017331074

4^{th} year, 2^{nd} Semester

Iqramul Islam

Reg. No.: 2017331054

4^{th} year, 2^{nd} Semester

Department of Computer Science and Engineering

Supervisor

Mohammad Shahidur Rahman

Professor

Department of Computer Science and Engineering

26^{th} February, 2023

Recommendation Letter from Thesis/Project Supervisor

The thesis/project entitled *Diffusion Model for Oversampling Class Imbalanced Tabular Data* submitted by the students

1. Rafat Ashraf Joy
2. Iqramul Islam

is under my supervision. I, hereby, agree that the thesis/project can be submitted for examination.

Signature of the Supervisor:

Name of the Supervisor: Mohammad Shahidur Rahman

Date: 26th February, 2023

Certificate of Acceptance of the Thesis/Project

The thesis/project entitled *Diffusion Model for Oversampling Class Imbalanced Tabular Data* submitted by the students

1. Rafat Ashraf Joy
2. Iqramul Islam

on today is, hereby, accepted as the partial fulfillment of the requirements for the award of their Bachelor Degrees.

Head of the Dept.	Chairman, Exam. Committee	Supervisor
Mohammad Abdullah Al Mumin	M. Jahirul Islam	Mohammad Shahidur Rahman
Professor and Head	Professor	Professor
Department of Computer Science and Engineering	Department of Computer Science and Engineering	Department of Computer Science and Engineering

Abstract

In this study, a denoising diffusion probabilistic model was employed to oversample class imbalanced tabular data for the purpose of credit card default prediction. The synthetic data is statistically close to the original data and surpasses other deep generative models such as GAN and VAE in terms of classification performance. In addition, we present a pip package named 'keshik' which can work on any similar problem (i.e. oversample the minority category in a class imbalanced tabular data) via calling a simple API.

Keywords: Credit Default Prediction, Diffusion, Class imbalance problem

Acknowledgements

All praise is for Allah, who created us, the Most Compassionate, Most Merciful. And special thanks goes to our thesis supervisor Professor **Mohammad Shahidur Rahman** for giving us guideline, advice and encouragement for the work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This study investigates whether diffusion models can be used to generate synthetic tabular data in order to tackle class imbalance for the task of predicting credit default. Class imbalanced classification problems are prevalent in diverse domains such as networking [1] [2], finance [3] [4] [5], climatology [6] [7] and healthcare [8] [9] [10]. To address class imbalance, deep generative models such as variational autoencoders and generative adversarial networks have been employed to generate synthetic data. Diffusion models are a novel class of generative models and they offer better sample quality compared to VAEs and GANs in image generation. This brings the research question - whether diffusion model's better sample quality in images translates to tabular data.

1.1 Why Synthetic Tabular Data?

Getting actionable insights from data analytics has become an integral part of modern business operations. However, in major corporations in telecommunications, insurance, pharmaceuticals and banks, data sharing is often hindered by legal constraints such as those set by European General Data Protection Regulation (GDPR). As a substitute to allow data sharing while adhering to legal and privacy requirements, synthetic tabular data is emerging.

Below are a few reasons why synthetic data is necessary.

1. **Scarcity of Historical Data:** In some domains and use cases, it is near impossible to get historical data. Obtaining counterfactual data is helpful in many situations like these for testing hypotheses and making inferences.

2. **Handling Class Imbalance:** Class imbalance is a prevalent issue in some machine learning classification problems such as fraud detection, disease diagnosis and network security. In those imbalanced datasets, classical machine learning and anomaly detection algorithms often fail [11]. Through generating synthetic observations of the minority class, the machine learning classifier can generalize better and yield better results as evidenced by prior studies [12] [13] [14] [15].

3. **Data Sharing:** Sharing data is a very common practice among financial, pharmaceutical and manufacturing industries. Often these companies need to send data to third party consultancy firms to gain actionable insights via data analysis. However, due to legal restrictions stemmed from privacy concerns, it had become impossible to continue this practice. For example, the one prohibition set by GDPR regulation states that "Processing personal data is generally prohibited, unless it is expressly allowed by law, or the data subject has consented to the processing" [16]. Synthetic data allows the companies to circumvent this impediment [17] and devolve the task of analytics to specialized third party data analytics firms.

4. **Training Deep Learning Models:** In recent times, deep learning is becoming popular due to their accuracy and capability of finding complex patterns in data. However, one downside of it is that - the requirement of immense computational powers, which is delivered by cloud service providers. Uploading user data to a cloud is hindered by several reasons such as size, bandwidth and privacy concerns. Through utilizing synthetic data, this problem can be avoided.

1.2 Objective and Novelty of this study

Although lately diffusion models have got much traction for tasks like image synthesis [18], beating state-of-the-art GANs in terms of sample quality [19], so far no study has focused on them for the purpose of generating synthetic tabular data. This work

1. Investigates the question whether diffusion models can be used for the same type of over-sampling task that its two popular predecessors - variational autoencoder and generative adversarial network has been doing successfully.

2. In addition, this study presents a pip package named ‘keshik’ which can operate on any tabular data with binary target column.

Chapter 2

Literature Review

2.1 Class Imbalance Problem

So far there has been several research work for the problem of credit card fraud detection. In context of the class imbalance problem, these previous work can be divided into three broad categories.

1. **With Oversampling:** There are various oversampling techniques to handle class imbalance problem like Random Oversampling, SMOTE, Borderline-SMOTE, k-means SMOTE, SVM SMOTE, ADASYN, SMOTE-NC etc.

   Jin et al. [20] proposed a technique that predicts credit card defaulters based on SMOTE-XGBoost model. This study used SMOTE algorithm to oversample the training set samples and for preprocessing the input data. Chen et al. [21] proposed a prediction model that is based on k-means SMOTE and BP neural network. In this work, the data distribution is altered using the k-means SMOTE method, and the significance of the features is determined using random forest before being added to the BP neural network's initial weights for prediction.

   SMOTE is one of the most popular solution to the class imbalance problem. To solve class imbalance problem it creates new synthetic data instances. But a problem is that, it creates these instances at random, which leads to the generation of useless new instances. As a result both time and memory complexity get increased. There are some Borderline SMOTE based techniques, which tried to overcome this problem. For example, Al et al.

[22] proposed a technique titled AB-SMOTE. This is a variation of BSMOTE (Borderline SMOTE), that outperform the traditional BSMOTE. This technique considers the affinity of the borderline occurrences to improve the quality of the generated synthetic data. This paper also proposed another technique named as HCAB-SMOTE[23]. According to it, this technique outperforms AB-SMOTE too. In order to improve the density of the boundary, it combines undersampling for removing the majority of noise instances with oversampling techniques. In order to get better results, it uses k-means clustering on the borderline area to determine which clusters to oversample.

Though the oversampling techniques generally performs better than undersampling, the biggest drawback of those oversampling technique is that it increases the chance of overfitting by using identical replicas of previous cases. It is really extremely normal for a learner to create a classification rule for a single, duplicated case while oversampling.

2. **With Undersampling:** Chi et al. [24] takes a reinforcement learning based approach for undersampling the majority class in credit risk forecasting. Chen et al. [25] employs a combination of radial basis function neural networks, clustering and stochastic sensitivity measure to undersample for the problem of loan default prediction in China.

3. **Without Augmentation/Reduction:** For credit risk assessment, Luo et al. [4] introduced a deep learning strategy. For the prediction purpose in this study, the authors used restricted boltzman machines and deep belief networks. A feature selection methodology for credit risk data mining models was presented by Bach et al [26]. They used data on defaulting customers from Croatian financial institutions to fit certain decision tree models. The most notable variables were filtered out based on the models' accuracy in classifying data. To predict credit default, For the purpose of identifying potential credit defaults, Islam et al. [27] integrated machine learning with heuristics. Sayjadah et al. [28] used the random forest technique in conjunction with logistic regression. Non parametric random forest was used by Tang et al. [29] to assess credit risk in the Chinese energy sector. A general methodology for evaluating credit risks using machine learning was proposed by Kruppa et al [30]. In this study, random forest, k nearest neighbors, and bagged k nearest neighbors were used as non-parametric models. It was concluded that random forest is the best technique.

2.2 Generating Synthetic Tabular Data

2.2.1 Statistical Methods

1. **SMOTE**: Smote stands for *Synthetic Minority Oversampling Technique*. It was proposed by Chawla et al. [11]. It is one of the more sophisticated oversampling methods, that creates artificially new samples rather than adding duplicates of the existing points. The major application of the SMOTE is to address the classification issue of an unbalanced data set. The size of negative samples with low value density tends to have a negative impact on the categorization of positive samples with high value density from the standpoint of training models. Less value is used to train these models. The random oversampling technique is the foundation of the improved SMOTE scheme. A new sample point is created by randomly selecting a point on the line between the original sample point and one of the n sample points mentioned above. The main idea is to randomly select n minority class sample points from the original sample point's K nearest neighbors. The fundamental idea behind SMOTE is that nearby points on the feature space have similar properties. It will be more accurate than the conventional sampling method since it samples in the feature space rather than the data space.

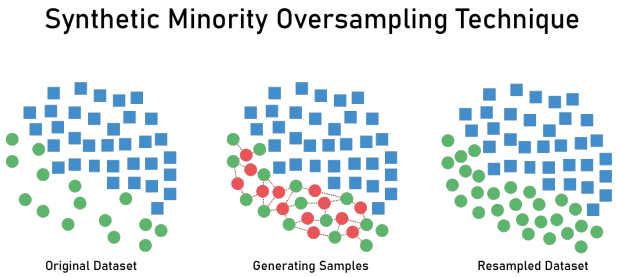

Figure 2.1: SMOTE

The Algorithm for SMOTE is given below [31]:

- Select the k-nearest neighbors that fall into the same category as the observed sample.
- Choose n samples of these k neighbors at random. The number n is determined by the over-sampling requirement; if 100% over-sampling is required, then n = 1.

- Analyze the differences between the observed sample's feature vector and each of its n neighbor feature vectors.
- A random number between 0 and 1 is multiplied by each of the variations.
- To create n additional synthetic observations, add each of these numbers independently to the feature vector of the observed sample.
- Return the freshly created samples.

2. Borderline SMOTE:

   Border-line SMOTE is another technique that is proposed by Han, Wang and Mao[32]. This is an alternative to the SMOTE. It has something to do with the border, just like the name suggests.As a result, Borderline-SMOTE only generates synthetic data along the decision border between the two classes, as opposed to SMOTE, which generates synthetic data at random between the two data. Additionally, Borderline-SMOTE comes in two varieties: Borderline-SMOTE1 and Borderline-SMOTE2. The differences are minimal, whereas Borderline-SMOTE2 only oversampled the minority classes, Borderline-SMOTE1 also oversampled the majority class when the majority data were the cause of misclassification in the decision boundary [33].

   If there are observations in the minority class that are anomalous and occur in the majority class, it creates a line bridge with the majority class, which is problematic for SMOTE. Borderline SMOTE solves this issue by classifying minority observations first. If all of the neighbors are members of the majority class, it labels any minority observation as a noise point and ignores it while producing synthetic data. Additionally, it resamples entirely from a small number of border places where the majority and minority classes are the same (Extreme observations on which a support vector will typically pay attention to) [34].

3. ADASYN:

   Similar to SMOTE, ADASYN (Adaptive Synthetic) is synthetic data generation method. The core idea of ADASYN is to employ a weighted distribution for different minority class observations according to their level of difficulty in learning. In comparison to minority examples that are easier to learn, more synthetic data is produced for minority class examples that are more challenging to learn [35]. The difference between SMOTE and ADASYN is

that for every minority sample, the total amount of mocked data generated in SMOTE will always be equal. However, the ADASYN method uses a density function to decide how many simulated data points should be generated for each minority instance. To produce faked data, this method assigns unique weights to each minority sample. The main purpose of ADASYN is to produce simulated data observations for the minority class observations that were challenging to learn [31]. There are two major weaknesses of ADASYN:

- There may only be one minority example in each neighborhood for sparsely distributed minority cases.
- Due to its adaptability, ADASYN may lose some of its precision.

2.2.2 Generative Model Based Approaches

Lately, a few studies have focused on generating tabular data via deep generative models.

Table GAN [36] is a pioneering study by Park et al. on generating tabular data by generative adversarial networks with a focus on preserving the privacy and preventing information leakage. At first, The original table is converted into multiple square matrices of a particular shape. The generator network generates matrices of similar shape by sampling from a gaussian noise. The discriminator network, on the other hand, tries to differentiate between the real samples and the ones generated by the generator network.

Like its predecessor, TGAN [37] by Xu et al. employes generative adversarial network for the task. However, the difference is - in the generator network, LSTM with attention has been utilized to generate data column by column.

CTGAN [38] is a state-of-the-art tabular data generator which uses conditional generative adversarial network. Previous vanilla GAN architectures were not able to represent multimodal continuous variables. CTGAN handles this issue by mode specific normalization method. Categorical variables are often imbalanced. Owing to this, less frequent categories have the possibility of being omitted from the generated data. CTGAN ensures the fairness of inclusion of all categories via conditional generation and training by sampling.

Wan et al. [39] employed variational autoencoder for data generation with a focus on imbalanced learning. The study delineated how MNIST digits generated by variational autoencoder are better in terms of sample quality compared to SMOTE and ADASYN.

Islam et al. [40] used variational autoencoder for crash data augmentation.

Li et al. [41] investigated the research question whether synthetic data generated by variational autoencoders are differentially private for data sharing in the context of multi market businesses.

Chapter 3

Data

The data used in this study was collected from a prominent Taiwanese bank (a cash and credit card issuer) in October 2005, with the bank's credit card holders as its focus [42] [43]. It contains 30000 instances, each with 23 features. The target attribute 'isDefaulter' is a binary variable. The data is imbalanced, with 6636 instances of default and 23364 instances of non-default, resulting in a ratio of approximately 2:7.

The description of the feature columns are listed below [42] [43]:

3.1 Numerical Features

1. **AGE**: Ages are in years.
2. **LIMIT_BAL**: represents the amount of credit granted in New Taiwan dollars (includes family/supplementary credit as well as individual credit).
3. **BILL_AMT1**: Bill amount report for September 2005 (New Taiwan dollar)
4. **BILL_AMT2**: Bill amount report for August 2005 (New Taiwan dollar)
5. **BILL_AMT3**: Bill amount report for July 2005 (New Taiwan dollar)
6. **BILL_AMT4**: Bill amount report for June 2005 (New Taiwan dollar)
7. **BILL_AMT5**: Bill amount report for May 2005 (New Taiwan dollar)

8. **BILL_AMT6**: Bill amount report for April 2005 (New Taiwan dollar)
9. **PAY_AMT1**: Previously paid amount as of September 2005 (New Taiwan dollar)
10. **PAY_AMT2**: Previously paid amount as of September 2005 (New Taiwan dollar)
11. **PAY_AMT3**: Previously paid amount as of September 2005 (New Taiwan dollar)
12. **PAY_AMT4**: Previously paid amount as of September 2005 (New Taiwan dollar)
13. **PAY_AMT5**: Previously paid amount as of September 2005 (New Taiwan dollar)
14. **PAY_AMT6**: Previously paid amount as of September 2005 (New Taiwan dollar)

The kernel density estimation plot of the numerical variables are as follows:

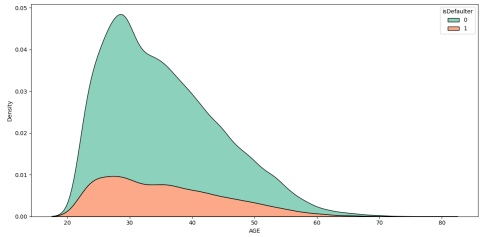

Figure 3.1: Age

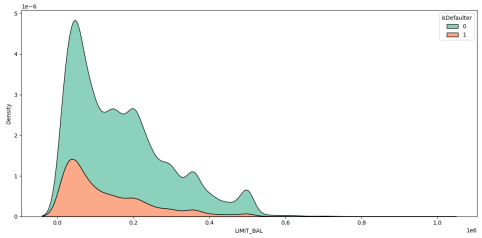

Figure 3.2: LIMIT_BAL

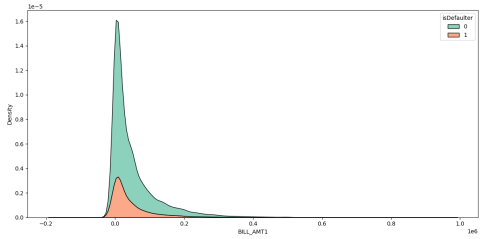

Figure 3.3: BILL_AMT1

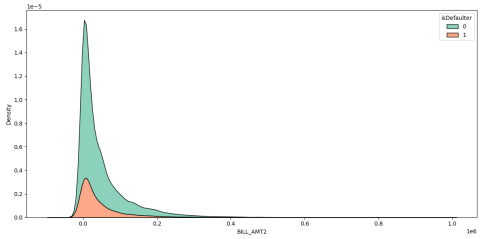

Figure 3.4: BILL_AMT2

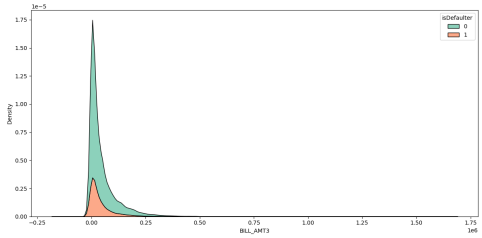

Figure 3.5: BILL_AMT3

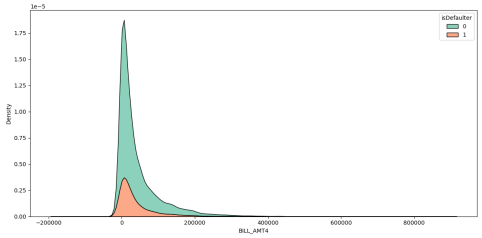

Figure 3.6: BILL_AMT4

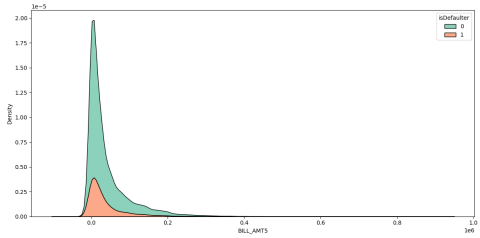

Figure 3.7: BILL_AMT5

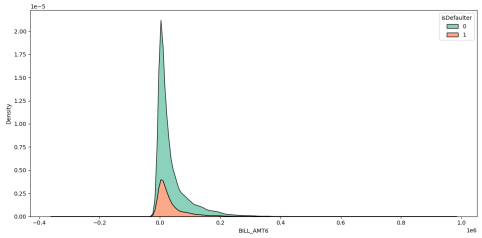

Figure 3.8: BILL_AMT6

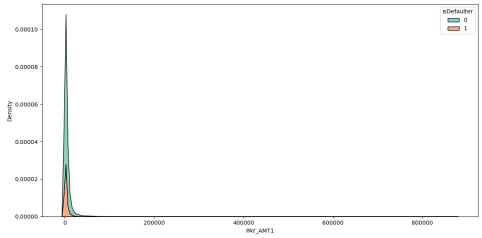

Figure 3.9: PAY_AMT1

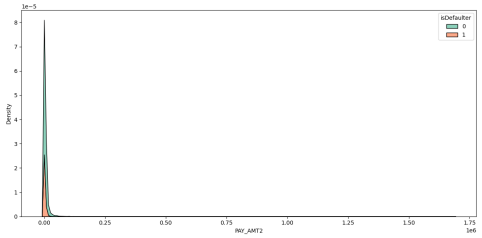

Figure 3.10: PAY_AMT2

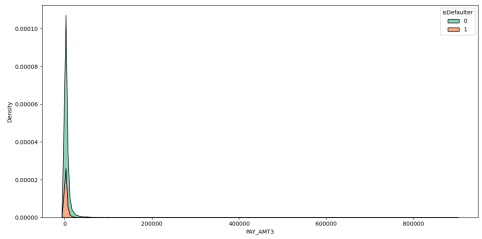

Figure 3.11: PAY_AMT3

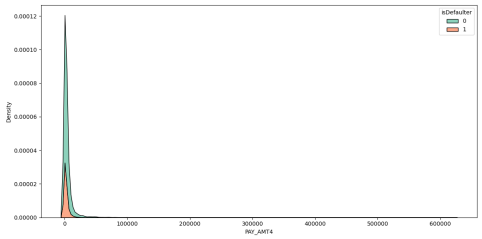

Figure 3.12: PAY_AMT4

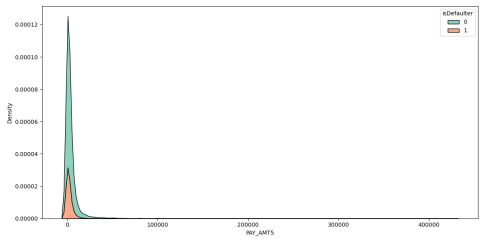

Figure 3.13: PAY_AMT5

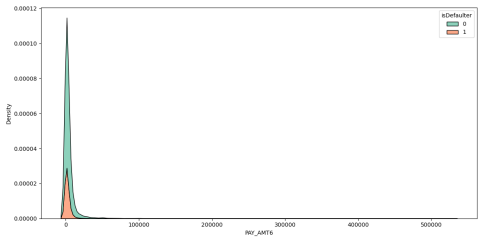

Figure 3.14: PAY_AMT6

Figure 3.15: Continuous Feature Distribution

3.2 Categorical Features

1. **ID**: Each client's unique identification number
2. **SEX**: Gender
3. **EDUCATION**: Education level (1 = graduate school, 2 = university, 3 = high school, 4 = others, 5 = unknown, and 6 = unknown)
4. **MARRIAGE**: Relationship status (1=married, 2=single, 3=others)
5. **PAY_0**: Status of repayment as of September 2005 (-1 = pay on time, 1 = one month late, 2 = two months late,... 8 = eight month late, 9 = nine or more month late)
6. **PAY_2**: Status of repayment as of August 2005 (identical to prior scale)
7. **PAY_3**: Status of repayment as of July 2005 (identical to prior scale)
8. **PAY_4**: Status of repayment as of June 2005 (identical to prior scale)
9. **PAY_5**: Status of repayment as of May 2005 (identical to prior scale)
10. **PAY_6**: Status of repayment as of April 2005 (identical to prior scale)

The Histogram of the categorical variables are as follows:

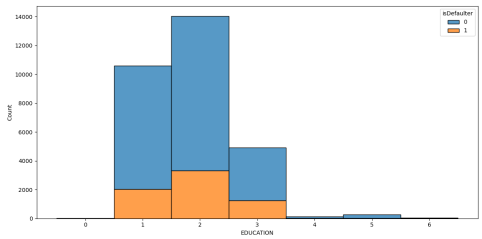

Figure 3.16: EDUCATION

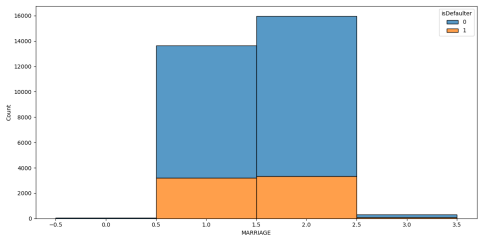

Figure 3.17: MARRIAGE

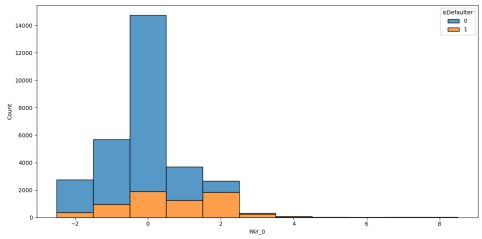

Figure 3.18: PAY_0

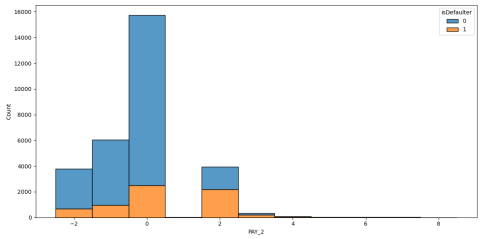

Figure 3.19: PAY_2

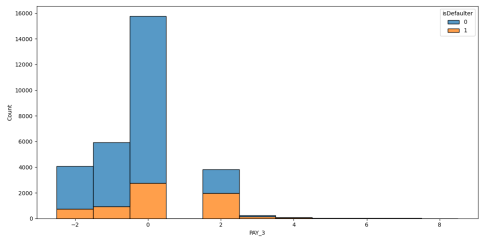

Figure 3.20: PAY_3

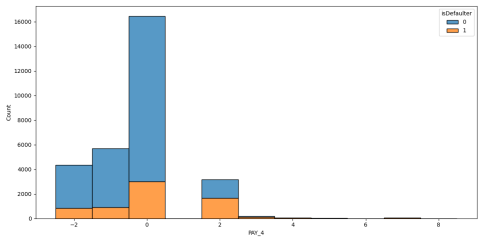

Figure 3.21: PAY_4

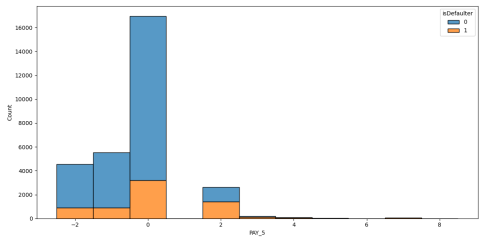

Figure 3.22: PAY_5

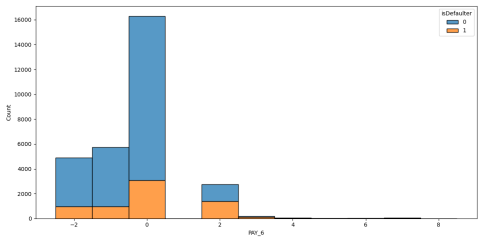

Figure 3.23: PAY_6

Figure 3.24: Discrete Feature Distribution

Chapter 4

Methodology

4.1 Deep Generative Models

Deep generative models are neural networks with multiple hidden layers that have been trained using samples to approximate complex, high-dimensional probability distributions [44] [45]. When properly trained, it can be used to generate high quality samples from the underlying distribution and to calculate the likelihood of each observation. In this study, apart from diffusion, two deep generative models - variational autoencoder (VAE) and Wasserstein Generative Adversarial Network (WGAN) has been used for comparison purpose.

4.2 Diffusion

Diffusion consists of two processes, a forward process where the original data is perturbed by iteratively injecting random noise and a reverse process where noise is successively removed from a pure noise distribution to generate new samples. These processes can be represented by two Markov chains: a forward chain that perturbs the original data to noise and a reverse chain that reverts noise to data - which is the process of sample generation. In the forward chain, the objective is to transform data into a simple probability distribution such as normal distribution. The reverse chain learns transition kernel parameterized by neural network with a view to undoing the forward process. There are three different formulations of diffusion.

1. Denoising Diffusion Probabilistic Models (DDPMs) [46]

2. Score Based Generative Models (SGMs) [47]

3. Stochastic Differential Equation Based Models (Score SDE) [48]

In this study, DDPM has been implemented for synthetic data generation.

In mathematical form, if $x_0 \sim q(x_0)$ is the data distribution, the forward process yields a sequence of random variables $x_1, x_2, x_3, ...x_T$ with transition kernel $q(x_t|x_{t-1})$. Through markov property the join distribution of $x_1, x_2, x_3, ...x_T$ can be conditioned on x_0, as follows[46]:

$$q(x_1, x_2, x_3, ...x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{4.1}$$

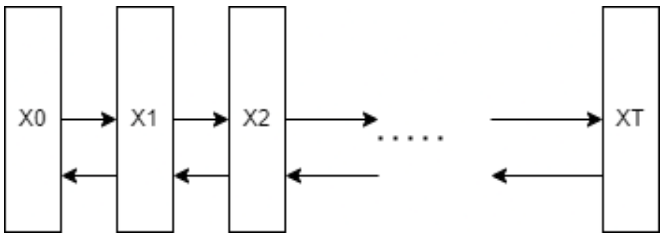

Figure 4.1: Schematic Diagram of Diffusion

Gaussian perturbation is a standard configuration for the transition kernel.Mathematically it can be expressed as follows:

$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \tag{4.2}$$

where, β_t is a hyperparameter chosen before training the model and it ranges from 0 to 1.

After the forward process iteratively injects noise to the data, all structures are lost. The initial step in creating new data samples for DDPMs is to create an unstructured noise vector from the prior distribution, and the noise is then gradually removed by running a learnable Markov chain in the opposite time direction. The reverse markov chain is parameterized by two terms:

1. A prior distribution. Since the forward process is constructed such that $q_T \approx \mathcal{N}(x_T; 0, 1)$, the prior is set as $p(x_T) = \mathcal{N}(x_T; 0, 1)$.

2. A learnable transition kernel, denoted as:

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)=\mathcal{N}\left(\mathbf{x}_{t-1} ; \mu_\theta\left(\mathbf{x}_t, t\right), \Sigma_\theta\left(\mathbf{x}_t, t\right)\right) \tag{4.3}$$

where, θ indicates model parameters and a neural network parameterizes Variance $\Sigma_\theta(x_t, t)$ and mean $\mu_\theta(x_t, t)$.

Given the reverse markov chain, it is possible to generate a data sample x_0 by sampling a noise vector $x_T \sim p(x_T)$ via the transition kernel. Training the reverse Markov chain to match the forward Markov chain's real time reversal is essential for the sampling procedure to be successful. The parameter θ has to updated such that the joint distribution of the reverse markov chain $p_\theta(x_1, x_2, ...x_T)$ closely approximates the joint distribution of the forward markov chain $q(x_1, x_2, ...x_T)$.

4.2.1 Training Algorithm

The steps of the training algorithm are as follows[46]:

I. repeat

II. $x_0 \sim q(x_0)$

III. $t \sim uniform(1, ...T)$

IV. $\epsilon \sim \mathcal{N}(0, 1)$

V. Gradient descent by this formula :

$$\nabla_\theta \left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0+\sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon}, t\right)\right\|^2$$

VI. Until converged.

4.2.2 Sampling Algorithm

The algorithm for sampling is as follows:

I $x_T \sim \mathcal{N}(0, I)$

II for t=T,....,1 do :

$z \sim \mathcal{N}(0, I)$ if $t > 1$, else z=0

$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta \left(\mathbf{x}_t, t \right) \right) + \sigma_t \mathbf{Z}$

III end for

IV return x_0

4.3 Variational AutoEncoder

Variational autoencoders are a combination of variational inference and deep learning.They are probabilistic generative models that have two coupled, but independently parameterized models [49]. They are called encoder and decoder[50]. The input variable is mapped by the first neural network to a latent space that corresponds to a variational distribution's parameters. Through this mechanism, the encoder is able to create a collection of samples that are all derived from the input data distribution. On the contrary, the decoder's purpose is to produce or generate data points by mapping from the latent space to the output space [49]. The difference between an autoencoder and a variational autoencoder is that the encoder generated latent space is regularized to be close to a standard gaussian.

The encoder $q_\phi(z|x)$finds the latent representation z, of the input vector x and it is parameterized by ϕ. The decoder $p_\theta(x'|z)$ tries to reconstruct the latent representation z to x' and it is parameterized by θ.

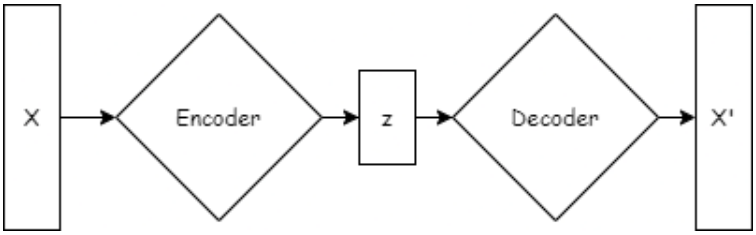

Figure 4.2: Schematic Diagram of Variational AutoEncoder

There are two terms in the loss function, L of a variational autoencoder.

$$L = log p_\theta(x|z) - D_{KL}(q_\phi(z|x)||p_\theta(z)) \tag{4.4}$$

The first term $log p_\theta(x|z)$ minimizes the reconstruction error between the input and output while the second term penalizes the disparity between the encoder's result $q_\phi(z|x)$ and the prior $p_\theta(z)$.

4.4 Wasserstein GAN

The main idea behind Generative Adversarial Networks (GANs) is to train two networks an adversarial fashion, playing a two player minimax game. The generator network maps from a noise vector to the generated data samples. On the contrary, the discriminator network is fed both the genuine and generator yielded fake samples, with a goal to discriminate between these two classes.

If P_{data} is the data distribution and P_{model} is the model distribution, then the generator network's goal is to shift P_{model} to P_{data}. For that purpose it samples a noise vector z from noise prior P_z and generates a sample G(z) of the same dimension of the real data. The discriminator network tries to differentiate between these two classes. Through this minimax game, the generator gets better at creating samples that are close to the real ones and the discriminator gets better at distinguishing the fake and real samples.

The objective function of generative adversarial networks is defined as follows:

$$\min_G \max_D \mathbb{E}_{x\sim P_{\text{data}}\,(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{4.5}$$

Standard generative adversarial networks use Jensen-Shannon divergence to measure the difference between two probability distributions. However, it leads to vanishing gradient problem, since it is not differentiable in every point of space. To overcome this shortcoming, wasserstein distance has been proposed. Regardless of whether the generator is working or not, WGAN learns due to the smoother gradient. As exact computation of wasserstein distance is intractable, it is approximated in Kantorovich-Rubinstein duality form [51] -

$$W\,(p_{\text{data}}\,, p_{\text{model}}\,) = \sup_{\|f\|_L<1} \left[E_{x\sim p_{\text{data}}} f(x) - E_{z\sim p_{\text{model}}} f(x)\right] \tag{4.6}$$

where the where the supremum(the least upper bound) is computed over all 1-Lipschitz continuous functions.

4.5 API Development

We have developed a pip package which allows developers utilize the functionality of our research in a more generalizable manner.

The sourcecode can be found at: https://github.com/rajoy99/keshik.

4.5.1 Scaling and Inverse Scaling

Before applying the diffusion model on the training dataframe, it was scaled to $[-1, 1]$range by robust scaling method. Robust scaler is defined as follows:

$$x_i = \frac{x_i - x_{median}}{x_{75} - x_{25}} \tag{4.7}$$

where, $x_{75} - x_{25}$ is the interquartile range. By removing the median, this Scaler scales the data according to the quartile range. Since the data had some outliers, robustscaler was chosen for scaling.

By doing scaling, the neural network reverse process is guaranteed to operate on consistently scaled inputs beginning with the conventional normal prior $P(X_T)$ [46]. After the generation was done, the data was rescaled to its original range, by applying an inverse transform.

4.5.2 The Auxiliary Neural Network

Since diffusion models require an auxiliary model to be able to predict the added noise, a neural network(or any mathematical function mapping from R^d to R^d) is necessary. The original diffusion paper [46], used a U-Net like architecture for that purpose. In our DDPM implementation, the auxiliary neural network is made of a basic building block named 'CBL layer'. Each CBL layer performs three operations in the following order:

- 2 dimensional convolution
- 2 dimensional batch normalization
- LeakyReLU activation

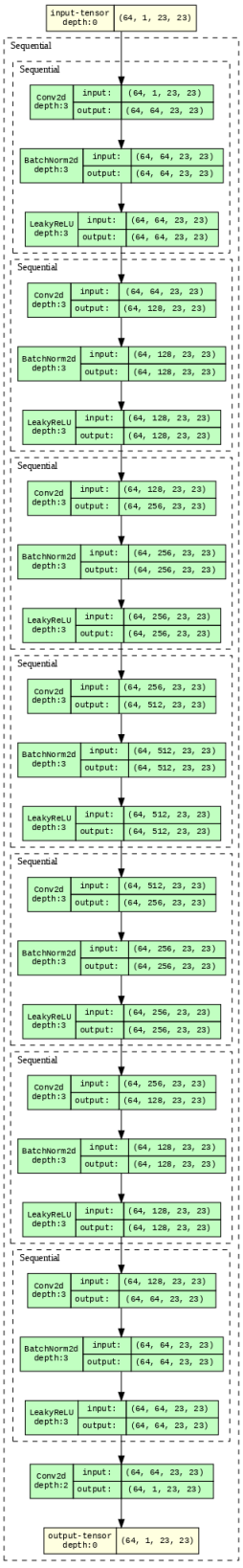

Figure 4.3: Architecture of the noise predictor network

4.5.3 Data Generation

Tabular data's inherent heterogeneity makes proper modeling difficult because the individual features may have entirely different characteristics. There are three choices of feeding the original data into the diffusion model. Each of these has different consequences.

1. **By Column:** Generating one column at a time. Since all the values in a particular column tend to be homogeneous, this approach would be perfect to generate values which are least unperturbed from the influence of other columns.

2. **By Row:** Generating one row at a time. While this preserves the semantic consistency among features, it also leads to adulteration of one column by nearby column.

3. **By Matrix:** Splitting the entire input dataframe into multiple square matrices where (n*n), where n is the size of the column. This approach ensures the semantic consistency among columns. However, the downside of this method is similar to by rows approach.

The sourcecode of the diffusion model and the oversampling utility functions are added in the appendix.

4.5.4 Handling Categorical Columns

Handling categorical values was a challenge in this work because of the intricacies in sampling from a discrete distribution.

For categorical features, two approaches can be followed:

1. From the numerical features train any classifier (such as Logistic Regression, Artificial Neural Network or SVM) to predict the discrete label.

2. Rounding off the generated float values.

The API has an optional argument named 'categorical_columns' in which the list of the name of the categorical columns can be passed. If it is passed, then the generated values in this column will be handled according to either two of the strategies, which can be selected via an optional parameter named 'cat_handle_mode'.

Chapter 5

Results and Discussion

5.1 Classification Performance

To measure the performance of our approach, we have chosen ROC-AUC. There are two factors behind this choice. First, since the data is imbalanced, ROC-AUC is a good metric instead of accuracy. Second, previous research studies on the class imbalance problems mainly used ROC-AUC as the error metric.

ROC stands for Reciever operating characteristic curve which plots the true positive rate (TPR) versus the false positive rate (FPR) rate at different classification thresholds.

TPR is defined as follows:

$$TPR = \frac{TP}{TP + FN} \tag{5.1}$$

FPR is defined as follows:

$$FPR = \frac{FP}{FP + TN} \tag{5.2}$$

To compare the performance of our method with previous approaches, we trained a logistic regression model. The following table shows the ROC-AUC scores of different interpolation based and deep learning based approaches along with the results acheived by using keshik package.

Approach	ROC-AUC	F1 Score
Voting Classifier on oversampled data	0.7185	0.47877
Voting Classifier on original data	0.6988	0.44324
LSTM [52]	–	0.4649

Table 5.2: Comparison with previous work

Approach	ROC-AUC
Imbalanced	0.38
SMOTE	0.58
ADASYN	0.57
VAE	0.55
WGAN	0.50
keshik - by rows	0.54
keshik - by columns	0.51
keshik - by matrices	0.57

Table 5.1: Classification by Logistic Regression

5.1.1 Comparison with previous studies

To benchmark our approach with previous studies, we have trained a voting classifier consisting of five different classifiers and averaged their results. The classifiers are :

1. Multi Layer Perceptron
2. Random Forest Classifier
3. Logistic Regression
4. K Nearest Neighbours
5. Decision Tree Classifier

Table 5.2 shows the comparison of previous studies and our approach's result:

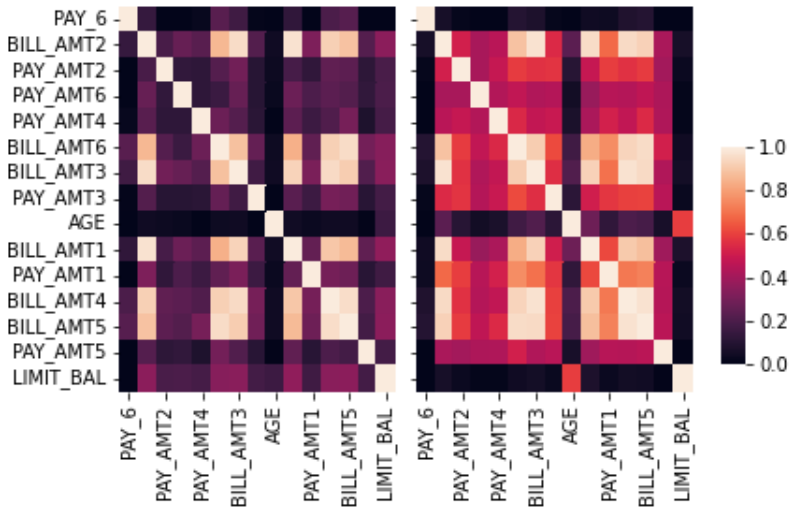

Figure 5.1: Correlation plot among numerical columns; Left: Original, Right: Synthetic

5.2 Quality of the synthetic data

To measure the quality of the generated data, three performance metrics can be used:

1. Checking manually by contrasting real data and generated data distribution plots.
2. Statistical distribution distances such as Hellinger distance, Jensen Shannon divergence and Wasserstein distance.

5.2.1 Manual Contrasting via plots

5.2.1.1 Pairwise Correlation Plot

Pairwise correlation plots reveal how well the semantic consistency was preserved in the synthetic data compared to the original data.

5.2.1.2 Distribution Plot

Figure shows the distribution plots of the original minority data and synthetic data.

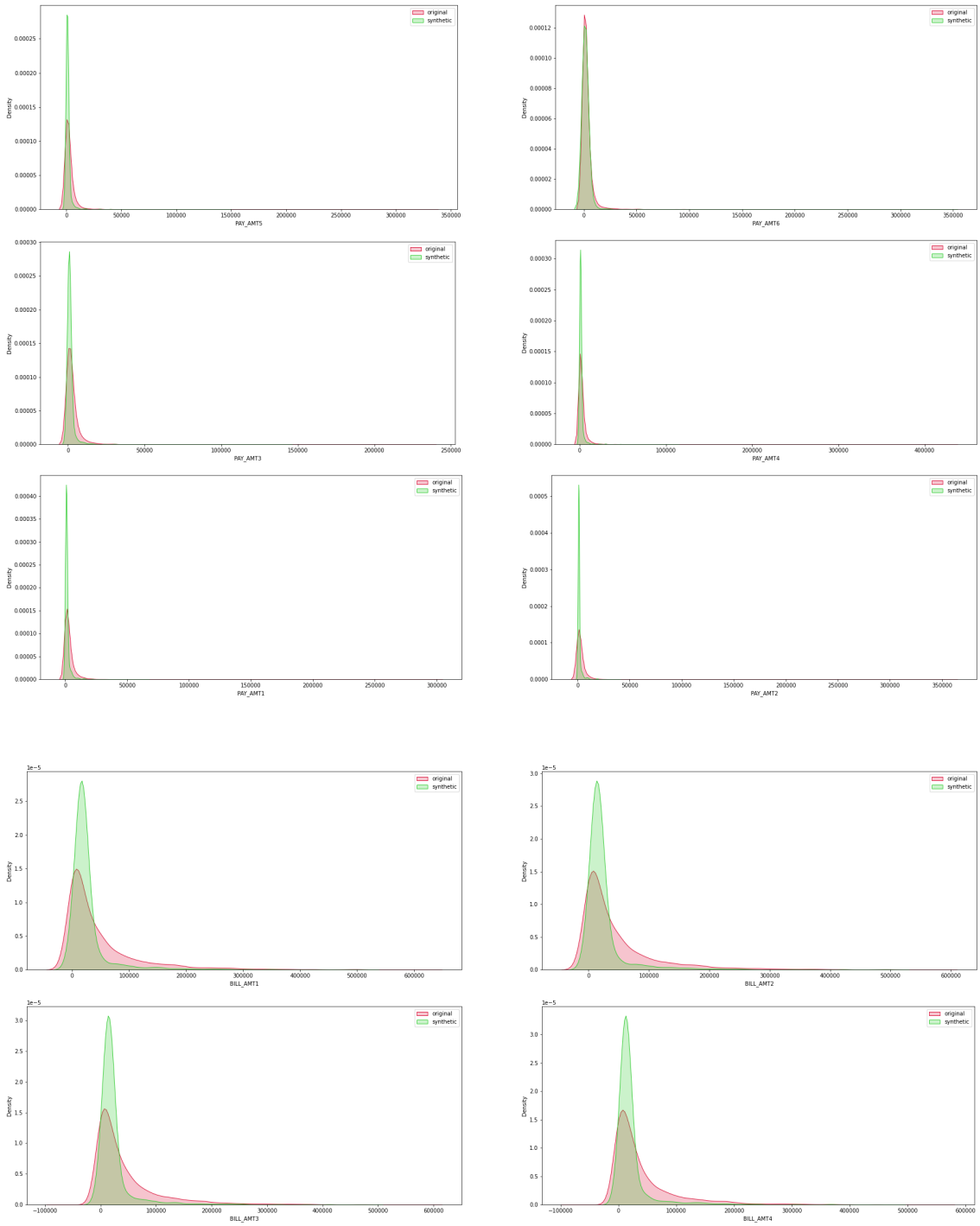

Figure 5.2: Comparison of Real vs Synthetic Continuous variable distribution

Except some skewed distribution such BILL_AMT1, BILL_AMT2, BILL_AMT3, BILL_AMT4, our approach can estimate the original data distribution quite faithfully.

5.2.2 Statistical distribution distance

Jensen-Shannon divergence is a measure for estimating how similar two probability distributions are to one another. In mathematical terms, it is defined as follows:

$$D_{JS}(P\|Q) = \frac{1}{2}D_{KL}(P\|M) + \frac{1}{2}D_{KL}(Q\|M) \tag{5.3}$$

where D_{KL} is the Kullback − Leibler divergence between two probability distributions.

Feature	BILL_AMT6	BILL_AMT3	PAY_AMT4	BILL_AMT4	BILL_AMT5
JS Divergence	0.1109707	0.1121464	0.08725559	0.116793	0.11496

Feature	PAY_AMT3	PAY_AMT6	BILL_AMT2	PAY_AMT5	LIMIT_BAL
JS Divergence	0.0834	0.08086	0.11089	0.08204	0.0848

Feature	AGE	PAY_AMT1	PAY_AMT2	SEX	MARRIAGE	EDUCATION
JS Divergence	0.0633	0.0874	0.0858	0.00098	0.2456	0.1073

Feature	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6
JS Divergence	0.439251	0.204345	0.500444	0.1501895	0.1039861	0.105403

Table 5.3: Jenssen Shannon divergence among Synthetic and Real Features

The distances were calculated using APIs from Scipy's stats submodule [53].

Chapter 6

Conclusion and Future Work

In this study, a denoising diffusion probabilistic model was used to generate synthetic data of the minority class. Although our approach did not outperform the state-of-the-art, this study has initiated a new direction of research. In future, a few areas of improvement can be identified. First, for each individual features taking into account the specific distribution, generate the data conditionally. Second, the categorical features can be handled more intelligently by using a LSTM network. Third, the noise predictor auxiliary network in this study is very basic. In subsequent work, more complex architectures as well as different configurations of neural network can be tested to see whether it improves overall data generation quality. Real world data often have multimodal and long tailed distributions, but in this work no attempts have been made to capture this type of complexity.

References

[1] H. Ding, L. Chen, L. Dong, Z. Fu, and X. Cui, "Imbalanced data classification: A knn and generative adversarial networks-based hybrid approach for intrusion detection," *Future Generation Computer Systems*, vol. 131, pp. 240–254, 2022.

[2] S. Bagui and K. Li, "Resampling imbalanced data for network intrusion detection datasets," *Journal of Big Data*, vol. 8, no. 1, pp. 1–41, 2021.

[3] J. Sun, J. Li, and H. Fujita, "Multi-class imbalanced enterprise credit evaluation based on asymmetric bagging combined with light gradient boosting machine," *Applied Soft Computing*, vol. 130, p. 109637, 2022.

[4] C. Luo, D. Wu, and D. Wu, "A deep learning approach for credit scoring using credit default swaps," *Engineering Applications of Artificial Intelligence*, vol. 65, pp. 465–470, 2017.

[5] Y. Liu, Q. Zeng, B. Li, L. Ma, and J. Ordieres-Meré, "Anticipating financial distress of high-tech startups in the european union: A machine learning approach for imbalanced samples," *Journal of Forecasting*, vol. 41, no. 6, pp. 1131–1155, 2022.

[6] D. Healy, Z. Mohammed, N. Kanwal, M. N. Asghar, and M. S. Ansari, "Deep learning model for thunderstorm prediction with class imbalance data," in *Proceedings of International Conference on Information Technology and Applications: ICITA 2021*. Springer, 2022, pp. 195–205.

[7] S. Ketu and P. K. Mishra, "Scalable kernel-based svm classification algorithm on imbalance air quality data for proficient healthcare," *Complex & Intelligent Systems*, vol. 7, no. 5, pp. 2597–2615, 2021.

[8] L. Li and G. Liu, "In-hospital mortality prediction for icu patients on large healthcare mimic datasets using class imbalance learning," in *2020 5th IEEE International Conference on Big Data Analytics (ICBDA)*. IEEE, 2020, pp. 90–93.

[9] J. Li, L.-s. Liu, S. Fong, R. K. Wong, S. Mohammed, J. Fiaidhi, Y. Sung, and K. K. Wong, "Adaptive swarm balancing algorithms for rare-event prediction in imbalanced healthcare data," *PloS one*, vol. 12, no. 7, p. e0180830, 2017.

[10] V. Kumar, G. S. Lalotra, P. Sasikala, D. S. Rajput, R. Kaluri, K. Lakshmanna, M. Shorfuzzaman, A. Alsufyani, and M. Uddin, "Addressing binary classification over class imbalanced clinical datasets using computationally intelligent techniques," in *Healthcare*, vol. 10, no. 7. MDPI, 2022, p. 1293.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[12] J. Wang, S. Li, B. Han, Z. An, H. Bao, and S. Ji, "Generalization of deep neural networks for imbalanced fault classification of machinery using generative adversarial networks," *Ieee Access*, vol. 7, pp. 111 168–111 180, 2019.

[13] Q. Li, L. Chen, C. Shen, B. Yang, and Z. Zhu, "Enhanced generative adversarial networks for fault diagnosis of rotating machinery with imbalanced data," *Measurement Science and Technology*, vol. 30, no. 11, p. 115005, 2019.

[14] K. Li, W. Ma, H. Duan, H. Xie, J. Zhu, and R. Liu, "Unbalanced network attack traffic detection based on feature extraction and gfda-wgan," *Computer Networks*, vol. 216, p. 109283, 2022.

[15] K. Huang and X. Wang, "Ada-incvae: Improved data generation using variational autoencoder for imbalanced classification," *Applied Intelligence*, vol. 52, no. 3, pp. 2838–2853, 2022.

[16] "Consent - general data protection regulation (gdpr)," https://gdpr-info.eu/issues/consent/, (Accessed on 02/12/2023).

[17] D. Rankin, M. Black, R. Bond, J. Wallace, M. Mulvenna, G. Epelde *et al.*, "Reliability of supervised machine learning using synthetic data in health care: Model to preserve privacy for data sharing," *JMIR medical informatics*, vol. 8, no. 7, p. e18910, 2020.

[18] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2256–2265.

[19] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8780–8794, 2021.

[20] L. Jin, Z. Wu, and J. Zhao, "Prediction of credit card defaulters based on smote-xgboost model," 2022.

[21] Y. Chen and R. Zhang, "Research on credit card default prediction based on k-means smote and bp neural network," *Complexity*, vol. 2021, pp. 1–13, 2021.

[22] H. Al Majzoub and I. Elgedawy, "Ab-smote: An affinitive borderline smote approach for imbalanced data binary classification," *International Journal of Machine Learning and Computing*, vol. 10, no. 1, pp. 31–37, 2020.

[23] H. Al Majzoub, I. Elgedawy, Ö. Akaydın, and M. Köse Ulukök, "Hcab-smote: A hybrid clustered affinitive borderline smote approach for imbalanced data binary classification," *Arabian Journal for Science and Engineering*, vol. 45, no. 4, pp. 3205–3222, 2020.

[24] J. Chi, G. Zeng, Q. Zhong, T. Liang, J. Feng, X. Ao, and J. Tang, "Learning to undersampling for class imbalanced credit risk forecasting," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 72–81.

[25] Y.-Q. Chen, J. Zhang, and W. W. Ng, "Loan default prediction using diversified sensitivity undersampling," in *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1. IEEE, 2018, pp. 240–245.

[26] M. P. Bach, J. Zoroja, B. Jaković, and N. Šarlija, "Selection of variables for credit risk data mining models: preliminary research," in *2017 40th International Convention on Information*

and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2017, pp. 1367–1372.

[27] S. R. Islam, W. Eberle, and S. K. Ghafoor, "Credit default mining using combined machine learning and heuristic approach," *arXiv preprint arXiv:1807.01176*, 2018.

[28] Y. Sayjadah, I. A. T. Hashem, F. Alotaibi, and K. A. Kasmiran, "Credit card default prediction using machine learning techniques," in *2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA)*. IEEE, 2018, pp. 1–4.

[29] L. Tang, F. Cai, and Y. Ouyang, "Applying a nonparametric random forest algorithm to assess the credit risk of the energy industry in china," *Technological Forecasting and Social Change*, vol. 144, pp. 563–572, 2019.

[30] J. Kruppa, A. Schwarz, G. Arminger, and A. Ziegler, "Consumer credit risk: Individual probability estimates using machine learning," *Expert systems with applications*, vol. 40, no. 13, pp. 5125–5131, 2013.

[31] A. Dasgupta, "Credit card default prediction using smote and machine learning algorithms," 2020.

[32] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing: International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I 1*. Springer, 2005, pp. 878–887.

[33] "medium.com," https://medium.com/m/global-identity-2?redirectUrl=https%3A%2F%2Ftowardsdatascience.com%2F5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5, (Accessed on 02/13/2023).

[34] "medium.com," https://medium.com/m/global-identity-2?redirectUrl=https%3A%2F%2Ftowardsdatascience.com%2Fclass-imbalance-smote-borderline-smote-adasyn-6e36c78d804, (Accessed on 02/13/2023).

[35] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 2008, pp. 1322–1328.

[36] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks," *arXiv preprint arXiv:1806.03384*, 2018.

[37] L. Xu and K. Veeramachaneni, "Synthesizing tabular data using generative adversarial networks," *arXiv preprint arXiv:1811.11264*, 2018.

[38] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[39] Z. Wan, Y. Zhang, and H. He, "Variational autoencoder based synthetic data generation for imbalanced learning," in *2017 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2017, pp. 1–7.

[40] Z. Islam, M. Abdel-Aty, Q. Cai, and J. Yuan, "Crash data augmentation using variational autoencoder," *Accident Analysis & Prevention*, vol. 151, p. 105950, 2021.

[41] S.-C. Li, B.-C. Tai, and Y. Huang, "Evaluating variational autoencoder as a private data release mechanism for tabular data," in *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2019, pp. 198–1988.

[42] I.-C. Yeh and C.-h. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert systems with applications*, vol. 36, no. 2, pp. 2473–2480, 2009.

[43] "Uci machine learning repository: default of credit card clients data set," https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients, (Accessed on 11/30/2022).

[44] L. Ruthotto and E. Haber, "An introduction to deep generative modeling," *GAMM-Mitteilungen*, vol. 44, no. 2, p. e202100008, 2021.

[45] R. Salakhutdinov, "Learning deep generative models," *Annual Review of Statistics and Its Application*, vol. 2, pp. 361–385, 2015.

[46] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[47] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," *Advances in neural information processing systems*, vol. 32, 2019.

[48] Y. Song, C. Durkan, I. Murray, and S. Ermon, "Maximum likelihood training of score-based diffusion models," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1415–1428, 2021.

[49] D. P. Kingma, M. Welling *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[50] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. [Online]. Available: http://probml.github.io/book2

[51] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[52] R. Liu, "Machine learning approaches to predict default of credit card clients," *Modern Economy*, vol. 9, no. 11, pp. 1828–1838, 2018.

[53] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

Appendices

Appendix A

SourceCode

DDPM_Model.py

```
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST
from torchvision import transforms
from torchvision.utils import save_image, make_grid
from typing import Dict, Tuple
from tqdm import tqdm


def scheduler(beta1, beta2, T) :

    """

    Computes schedules for training and sampling.
    Inputs:
    beta1 : float, within range(0,1)
    beta2 : float, within range(0,1)
    T : int, number of perturbation iterations
    Returns : Dict[str,torch.Tensor]

    """
```

```
    beta_t = (beta2 - beta1) * torch.arange(0, T + 1, dtype=torch.float32) / T
                                                + beta1
    sqrt_beta_t = torch.sqrt(beta_t)
    alpha_t = 1 - beta_t
    log_alpha_t = torch.log(alpha_t)
    alphabar_t = torch.cumsum(log_alpha_t, dim=0).exp()

    sqrtab = torch.sqrt(alphabar_t)
    oneover_sqrta = 1 / torch.sqrt(alpha_t)

    sqrtmab = torch.sqrt(1 - alphabar_t)
    mab_over_sqrtmab_inv = (1 - alpha_t) / sqrtmab

    return {
        "alpha_t": alpha_t,
        "oneover_sqrta": oneover_sqrta,
        "sqrt_beta_t": sqrt_beta_t,
        "alphabar_t": alphabar_t,
        "sqrtab": sqrtab,
        "sqrtmab": sqrtmab,
        "mab_over_sqrtmab": mab_over_sqrtmab_inv
    }

# Template Layer for use in auxiliary model: Convolution -> BatchNorm ->
                                        LeakyRelU
CBL_layer = lambda ic, oc: nn.Sequential(
    nn.Conv2d(ic, oc, 8, padding=2),
    nn.BatchNorm2d(oc),
    nn.LeakyReLU(),
)

class Auxiliary(nn.Module):

    """
        Predicts the Noise for the reverse markov chain
```

```
        Any Mathematical function (or neural network) that maps from n-
                                                dimensional space
        to n-dimensional space
    """

    def __init__(self, n_channel: int) -> None:
        super(Auxiliary, self).__init__()
        self.conv = nn.Sequential(
            CBL_layer(n_channel, 64),
            CBL_layer(64, 128),
            CBL_layer(128, 256),
            CBL_layer(256, 512),
            CBL_layer(512, 256),
            CBL_layer(256, 128),
            CBL_layer(128, 64),
            nn.Conv2d(64, n_channel, 3, padding=1),
        )

    def forward(self, x, t) -> torch.Tensor:


        return self.conv(x)




class Diffusion(nn.Module):

    def __init__(
        self,
        Epsilon_Predictor: nn.Module,
        betas: Tuple[float, float],
        n_T: int,
        criterion: nn.Module = nn.MSELoss(),
    ) -> None:
        super(Diffusion, self).__init__()
        self.Epsilon_Predictor = Epsilon_Predictor
```

```
        for k, v in scheduler(betas[0], betas[1], n_T).items():
            self.register_buffer(k, v)

        self.n_T = n_T
        self.criterion = criterion


    def forward(self, x: torch.Tensor) -> torch.Tensor:

        """

        Training Process
        1. Implements forward markov chain of Diffusion
        2. Predicts noise value from x_t using Epsilon_Predictor
        3. Returns the loss of the Epsilon Predictor(or the Auxiliary Model)

        """


        # Sample t from Uniform Distribution : u(0, n_T)
        _ts = torch.randint(1, self.n_T, (x.shape[0],)).to(
            x.device
        )


        # Sample epsilon from Normal Distribution : N(0, 1)
        eps = torch.randn_like(x)


        # Compute x_t from Diffusion's forward chain formula
        x_t = (
            self.sqrtab[_ts, None, None, None] * x + self.sqrtmab[_ts, None,
                                                    None, None] * eps
        )
```

```
        # Predict the error term from the auxiliary model and return the loss
        return self.criterion(eps, self.Epsilon_Predictor(x_t, _ts / self.n_T))



    def sample(self, desired_samples: int, size, device):

        """
        Generates new data by the reverse Diffusion process
        Input :
        desired_samples : Type (int)
        size : Type (tuple)
        """

        # Sample x_T from Normal Distribution : N(0, 1)
        x_i = torch.randn(desired_samples, *size).to(device)


        # Sampling Algorithm in DDPM Paper.
        for i in range(self.n_T, 0, -1):

            z = torch.randn(desired_samples, *size).to(device) if i > 1 else 0
            eps = self.Epsilon_Predictor(x_i, i / self.n_T)

            #Reverse Diffusion Process. Will run Until i=1. x_1 is the
                                                    Generated Sample
            x_i = (
                self.oneover_sqrta[i] * (x_i - eps * self.mab_over_sqrtmab[i])
                                                        + self.sqrt_beta_t[i]
                                                       * z
            )

        return x_i
```

oversampler.py

```
import numpy as np
```

```
from sklearn.preprocessing import MinMaxScaler,RobustScaler
from traindiffusion_byMatrix import train_Diffusion_byMatrix
from traindiffusion_byRows import train_Diffusion_byRows
from traindiffusion_byColumns import train_Diffusion_byColumns
import pandas as pd
from sklearn.neural_network import MLPClassifier


def Diffusion_Oversampler(X_train,y_train,n_iteration=2,categorical_columns=[],
                                        n_T=1000,oversampling_mode='byMatrix',
                                        cat_handle_mode='prediction'):


    cols=X_train.shape[1]
    col_names=X_train.columns

    #minority determination
    minority_class=y_train.value_counts().index[-1]
    majority_class=y_train.value_counts().index[0]
    print("Minority and Majority class are: ",minority_class,majority_class)
    desired=y_train.value_counts().iloc[0] - y_train.value_counts().iloc[1]

    original_data_minority=X_train[y_train==minority_class]
    original_data_majority=X_train[y_train==majority_class]

    #Scaling
    scaler=RobustScaler()
    scaled_minority=scaler.fit_transform(original_data_minority)




    # Synthetic Data Generation
    if oversampling_mode=='byMatrix':
        generated=train_Diffusion_byMatrix(data=scaled_minority,n_iteration=
                                                n_iteration,augment_size=
                                                desired,n_T=n_T)
```

```
elif oversampling_mode=='byColumns':
    generated=train_Diffusion_byColumns(data=scaled_minority,n_iteration=
                                        n_iteration,augment_size=
                                        desired,n_T=n_T)
elif oversampling_mode=='byRows':
    generated=train_Diffusion_byRows(data=scaled_minority,n_iteration=
                                        n_iteration,augment_size=
                                        desired,n_T=n_T)


#Inverse Scaling
generated_rescaled=scaler.inverse_transform(generated)
generatedDF = pd.DataFrame(generated_rescaled, columns=col_names)


# Handling Categorical variables
if cat_handle_mode=='prediction':
    numerical_columns=list(set(list(X_train.columns))-set(
                                        categorical_columns))
    if len(categorical_columns)>0:
        Classifier=MLPClassifier()
        TrainX=original_data_minority[numerical_columns]
        TestX=generatedDF[numerical_columns]

        for v in categorical_columns:

            TrainY=original_data_minority[v]
            Classifier.fit(TrainX,TrainY)

            generatedDF[v]=Classifier.predict(TestX)

elif cat_handle_mode=='roundoff':
    if len(categorical_columns)>0:
        for v in categorical_columns:
            generatedDF[v]=generatedDF[v].map(np.round)
```

```
    frames = [generatedDF, original_data_minority, original_data_majority]
    totalX=pd.concat(frames)
    totalY=np.concatenate((np.full(generated_rescaled.shape[0]+
                                           original_data_minority.shape[0],
                                           minority_class),np.full(
                                           original_data_majority.shape[0],
                                           majority_class)),axis=None)


    return totalX,totalY
```

traindiffusion_byMatrix.py

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from typing import Dict, Tuple
from tqdm import tqdm
import numpy as np
from DDPM_model import Diffusion,Auxiliary


def train_Diffusion_byMatrix(data,n_iteration,augment_size,n_T=1000):

    #square matrix generation
    squared=[]
    rows=data.shape[0]
    cols=data.shape[1]

    i=0
    while i+cols<rows:
        squared.append(data[i:i+cols])
        i+=cols

    data=np.array(squared)

    cols=data.shape[1]
    data=torch.Tensor(data)
```

```
un_dataloader=DataLoader(data)



n_epoch = n_iteration
device="cuda:0"
ddpm = Diffusion(Epsilon_Predictor=Auxiliary(1), betas=(1e-4, 0.02), n_T=
                                            n_T)
ddpm.to(device)

generated_size=0

optim = torch.optim.Adam(ddpm.parameters(), lr=2e-4)

for i in range(n_epoch):

    ddpm.train()

    progress_bar = tqdm(un_dataloader)

    loss_ema = None

    for x in progress_bar:
        optim.zero_grad()
        # print("Shape of the fed tensor : ",x.shape)
        x = x.to(device)
        loss = ddpm(x)
        loss.backward()
        if loss_ema is None:
            loss_ema = loss.item()
        else:
            loss_ema = 0.9 * loss_ema + 0.1 * loss.item()
        progress_bar.set_description(f"loss: {loss_ema:.4f}")
        optim.step()


    ddpm.eval()
```

```
    generated=[]
    while generated_size<augment_size:

        with torch.no_grad():
            xh = ddpm.sample(1, (1,cols,cols), device)
            xh=xh.cpu().detach().numpy().reshape(cols,cols)

            generated.append(xh)
            generated_size+=cols

    generated=np.vstack(generated)
    print("Generated Shape : ",generated.shape)

    return generated
```