Towards Creating Synthetic Data Testbeds for Research

by

Nassim Oufattole

S.B., Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2023

Authored by: Nassim Oufattole
Department of Electrical Engineering and Computer Science
May 19, 2023

Certified by: Kalyan Veeramachaneni
Principal Research Scientist, Laboratory for Information and Decision Systems
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards Creating Synthetic Data Testbeds for Research

by

Nassim Oufattole

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Insurance datasets are generally private in order to protect user information, making it difficult for the ML research community to access and experiment with this data. To increase accessibility and innovation on private insurance data, we compile and share publicly available insurance datasets, analyze challenges inherent in these datasets, and propose, motivate, and evaluate a Synthetic Data sharing framework called Synthetic Insurance Data (`SID`) Testbed that can be used to improve ML performance on tabular datasets by allowing collaborators to generate Synthetic Data for Data Augmentation. In addition to this framework, we recognize that tabular data augmentation is not a well understood phenomenon, and we run controlled experiments to better understand how and when data augmentation improves machine learning performance in the setting of tabular data.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist, Laboratory for Information and Decision Systems

Acknowledgments

I am thankful for the vast support and guidance I received working on this thesis. First and foremost, I thank my advisor, Dr. Kalyan Veeramachaneni, for his insightful advice for designing experiments as well as for his advice on how to communicate well in papers. I can't express how thankful I am for Kalyan's mentorship and direction throughout my Masters. I also want to thank Dongyu, who I started this project working very closely with. He taught me the open source software development workflow, and I learned a lot working with him. I greatly appreciate all of the extremely bright students in my lab and friends who helped me and pushed me forward too. I finally want to thank my parents and my sister, who have always been my rock, through thick and thin.

Contents

List of Figures

List of Tables

Variable	Definition
$\mathbf{D}$	m dimensional random variable for samples of real data. All random variables are bold, and we use the notation that the non-bold, D in this case, is an $m \times n$ matrix of n samples of the random variable. D represents a real dataset. $\mathbf{D}$ has PDF $f(\cdot)$. All PDFs are lower case, and the captial case, $\mathbf{F}(\cdot)$, is the CDF. Marginal PDFs and CDFs use the notation $f_j(D_j)$. Inverse CDFs use the notation $\mathbf{F^{-1}}(\cdot)$. We use the following notation interchangably: $f(\mathbf{D}) = f(\mathbf{D}_1, \ldots, \mathbf{D}_m)$ for all PDFs.
$\mathbf{U}$	is $\mathbf{D}$ after a marginal CDF transform.
$\mathbf{\Omega}$	$\mathbf{\Omega} = \Phi_j^{-1}(\mathbf{U})$. Ω is chosen such that it has a tractable sampling distribution.
$\mathbf{\Omega'}$	represents samples of the random variable $\mathbf{\Omega}$ from the PDF $q(\mathbf{\Omega})$.
$\mathbf{U'}$	is the marginal CDF transform of $\mathbf{\Omega'}$ such that $\mathbf{U}'_j = \Phi_j(\mathbf{\Omega}'_j) \forall j$.
$\mathbf{D'}$	represents Synthetic Data. It is the marginal inverse CDF transform of $\mathbf{U'}$ such that $\mathbf{D}'_j = F_j^{-1}(\mathbf{U}'_j) \forall j$. D' is an $m \times n'$ matrix of samples of the random variable.
p	represents an empirical probability.
r	is the number of mixtures for a mixture model.
x, y	represents input and target samples for a dataset, where $x = (D_1, \ldots, D_{m-1})$ and $y = D_m$. x_i and y_i represent the i^{th} sample's features and target.
r	is the number of mixtures for a mixture model.
M	Machine learning model.

Table 1: Notation Table

Chapter 1

Introduction

More data is being produced and recorded today than ever before, largely thanks to the ever-increasing proliferation of data collection and storage technologies. At the same time, academics who study and work with data, whether to build new analysis methods, identify challenging problems, or simply test their methods, are able to access only a very small percentage of the many rich and unique datasets that might help them move their fields forward. This problem is especially profound for researchers working with tabular data, because privacy concerns keep many datasets in this format from being publicly released. In contrast, there are a number of large open datasets that can be used for image or natural language processing, and more can be compiled through scraping[1]. As a result of this lack of availability, a few types of tabular datasets have become disproportionately popular in academic communities using data for their research. These include:

Benchmarking datasets. These datasets are used by the academic community to benchmark their methods. They can be task-driven (like OpenML[2]) or general-purpose, like those used in a variety of datasets available through UCI[3]. Often, these datasets are less complex than their real-world counterparts.

[1] In computer vision and natural language processing, although labeling can be a difficult problem, the sheer availability of data is generally not, except in certain cases where very specific datasets are required (for example, x-rays for fractures).

[2] https://www.openml.org/

[3] https://archive.ics.uci.edu/ml/datasets.php

Competition datasets. These datasets are released by data science competition platforms like KAGGLE[4]. While widely available and sometimes reflective of industrial-world complexities, these are not generally widely adopted, potentially due to restrictions on redistribution rights.

Open datasets. A third category of datasets are those collected through a publicly available `API`, such as `github api`[5] or the `isodata` [6]. These datasets have interesting real world properties and come from real-world open systems; however, there are not very many of them.

While previously unacceptable in research publications, proprietary datasets are becoming more acceptable at research conferences, provided the methods are also applied to publicly available data. Proprietary datasets are rich, complex and can lead to fruitful research endeavors; including proprietary datasets can give papers additional credibility and bragging rights even though their results may not be verifiable. However, there has been no systematic effort to make such datasets available for research.

A possible solution to this *accessibility* problem is Synthetic Data. Over the past few years, the ability to generate high-quality synthetic tabular data has increased tremendously. Methods and open source software are available for producing Synthetic Data for tabular[50, 55], time series[23, 54, 32], and multi-table datasets. Synthetic datasets are also becoming more realistic, as progress is continually made in generating synthetic phone numbers and other unique data types. Such datasets may be appealing to entities that have a responsibility to protect users' private data but still want to work with outside parties on collaborative data science efforts. Such an entity could train a synthetic data (SD) generator on their private data, sample Synthetic Data, and share the newly-generated Synthetic Data with collaborators. Synthetic Data modeling depends on the prediction task or use case one expects to use it in.

In this thesis we focus on the insurance industry, which is interested in a number

[4]https://www.kaggle.com/
[5]https://docs.github.com/en/rest?apiVersion=2022-11-28
[6]https://pypi.org/project/isodata

of prediction problems. Here we briefly define two common problems we identified:

Loss Prediction the task of predicting how much money a particular insurance client will lose for the entity. This task is business critical for the industry; improvements in loss prediction can help us better understand what factors put people and companies at higher risk for loss, identify interventions, and take appropriate preventative measures [56]. Examples of past work for this task is predicting high loss claimants for auto-insurance[8, 46] or health insurance[34].

Fraud Prediction this is a binary classification task, where given information about an insurance loss, you must classify whether the loss is fraudulent. An example of past work for this task is predicting fraud for auto-insurance[4].

Within the scope of these prediction challenges, the ML task itself can be split into three types: model building (M), feature engineering + model building (FE+M), and Data Augmentation + model building (DA+M). An entity sharing data may be interested in any of these three specific challenges and may craft the task to augment internal efforts to develop the model. Datasets in this domain also present some domain-specific challenges for creating synthetic data generators, requiring domain-specific modeling solutions.

The majority of datasets, including the examples of past work given in the prediction task descriptions [8, 46, 34, 4], are not shared publicly or broadly with the research community. This can be due to privacy (or accessibility regulations) and also a lack of a motive for an entity to put in the effort to share this data broadly. However, we argue that both the privacy constraint and motivation hurdle can be overcome through a a **synthetic data-driven sharing**, and we define this solution in this thesis.

In creating a **synthetic data-driven sharing** solution, an entity may face several questions and challenges. First, how close to reality should the synthetic data be? Often the real data will be multi-table and may include a variety of data types, such as phone numbers and emails. Should the entity apply specific generators to create synthetic data for all tables and data types? In fact, many data science competitions have adopted synthetic data generators – for instance, the popular platform Kaggle

shared a variety of synthetic datasets under their Tabular Data Playground series in 2022[7]. A review of these datasets shows that none contained data types other than categorical/numeric, and almost all were single-table datasets.

Often times the `metadata`, like the column names, could reveal what the entity is collecting. Should these names be released as well? At present, many sites that release datasets also adopt *column-name* anonymization. Even if one decides to create a dataset that is as realistic as possible, how can this aforementioned realism be meaningfully evaluated while simultaneously measuring any possible privacy leakage? This also creates a second challenge: what motivates the entity in question to put in the effort required to share this dataset with the research team, and what are its motivations?

To address the challenges that come with creating and sharing synthetic data, we propose the development of **Synthetic data testbeds**. These testbeds have the following properties:

- **Task driven**: Synthetic data testbeds are driven by a machine learning task: DA+M, FE+M, or M. Being task-driven allows for the development of software frameworks around the synthetic data itself. This enables entities to take the most advantage of the research performed on the data they share, which incentivizes the releasing entity.
- **Domain specific**: These testbeds are designed for a specific domain, and consider the common or unique properties which datasets in that domain typically feature. It also allows for the creation of domain-specific rules for different prediction challenges and different tasks. This enables customization of the synthetic data generation methods and invocation of the appropriate evaluation methods.
- **Framework oriented**: Along with being task driven, we propose these testbeds have an `API` between entities and collaborators, making it easy for the releasing entity to adopt models created by collaborators and provide feedback to the collaborators.

[7]https://www.kaggle.com/competitions/tabular-playground-series-jan-2022

While these testbeds could be useful across many industries, here we detail their specific applications in the insurance domain. We call this the `SID` (Synthetic Insurance Data) testbed and propose that the insurance industry adopt this framework and utilize it to share datasets publicly and/or to share data with collaborators in a private setting.

In order to test the SID testbed, we need insurance data which is most of the time private. However, we were able to find a limited number of publicly available insurance datasets detailed in section 2 which we use to test a prototype of `SID`. We create synthetic data generators trained on these datasets, and provide sampling access and task-specific APIs that are useful for both collaborators and the releasing entity. Our testbed enables releasing entities to follow a systematic procedure to generate, evaluate, and share synthetic data, and subsequently assess and adopt solutions created by researchers working on the synthetic data. We specifically focus on the Data Augmentation(DA+M) task for the aforementioned prediction problems.

1.1 Research Questions

For creating the SID testbed, our first question is how we should model real data for Data Augmentation. Our Research Questions(RQ) for tabular Data Augmentation and for SID are:

RQ1 Is accurate univariate column modeling important for tabular Data Augmentation?

RQ2 Is accurate column dependency modeling important for tabular Data Augmentation?

RQ3 Given a generator, how should we sample for data augmentation?

RQ4 How many Synthetic Data samples should we generate for data augmentation?

RQ5 How could Synthetic Data be used in an ML workflow between an entity and collaborators such that both entities and collaborators benefit. Specifically, a

workflow must benefit entities by allowing them to receive models from collaborators they can easily integrate for the problem they are interested, all without sharing their private data. Additionally, a workflow must benefit collaborators by allowing them to receive feedback (in the form of ML metrics) on their ML model's performance on real data.

1.2 Contributions of this thesis

In this thesis, we attempt to develop synthetic data for insurance datasets via the following contributions:

- **Tabular Data Augmentation Intuition** We run controlled experiments to better understand when and how to apply Data Augmentation on tabular datasets. We restrict our exploration to several Copula methods for modeling the tabular data and generating synthetic data. We experiment with different sampling methods paired with these generators and evaluate these methods in terms of the predictive performance when augmenting with the generated data.
- **Synthetic Data generation for insurance dataset.** We develop a synthetic data generation method for the insurance datasets we work with. We discuss how to handle high dimensions, high cardinality categorical variables, non-Gaussian data, and other complexities.
- **Workflow for Data Augmentation.** We present a workflow for Tabular Data Augmentation for private datasets. We provide solutions for a variety of the nuances and challenges which exist in this process.

The thesis is organized as follows: Section 2 introduces publicly available insurance datasets, discusses the challenges and properties prevalent in insurance datasets, and introduces a simulated dataset with these properties. Section 3 introduces the machine learning pipeline we use for our experiments. In Section 4 we cover related work on Synthetic Data generation using Gaussian Copula models, Gaussian Copula failure modes, and models that can over come these failures. In Section 5, we cover related

work on Data Augmentation. Section 6 proposes the `SID` Testbed for collaborating on Synthetic Data augmentation to improve ML performance on private tabular insurance datasets. In section 7, we design controlled experiments that leverage the SID API to answer our research questions on Data Augmentation. Section 8 details our experimental results and analysis. Section 9 covers future works and limitations. Section 10 gives our conclusion and future works.

Chapter 2

Insurance Datasets

In this chapter, we describe the publically available insurance datasets we compiled to evaluate our data augmentation methods and identify synthetic data generator modeling challenges inherent in these datasets. We are assuming an insurance entity is interested in the loss prediction tasks and thus we attempt to use a subset of these datasets for that task.

The datasets we found are each created for a specific inference purpose: fraud prediction (the task of predicting whether a loss claim is real or is fraud), loss prediction (the task of predicting the monetary loss for an insurance client), or binary loss prediction (the task of predicting whether an insurance company will have a monetary loss greater than zero). Below we list which task each dataset is created for:

1. **Loss Prediction** allstate, emicen auto, emicen rail, prudential life [1],
2. **Binary Loss Prediction** home travel car portos
3. **Fraud Prediction** Fraud 2, databricks, Fraud 4.

In the following sections of this chapter, we describe these datasets, identify statistical and qualitative properties describing each dataset, and note the common challenges found with these datasets.

[1]This dataset is an edge case, as the task for this dataset is to classify the domain expert labeled ordinal risk associated with an insurance client.

2.1 Dataset Descriptions

Here we provide descriptions of each dataset, such as its origins, whether it is anonymized, and why it was created.

allstate [2]

This dataset was posted by Allstate as a part of a data-science competition on Kaggle. The goal of the competition is to create an ML model that can predict the loss for a client. It is unclear how exactly the dataset was generated and what type of insurance the dataset covers as it is not specified and Allstate offers a range of types of insurance coverage. All column names are fully-anonymized and have non-descriptive column names and values.

home [7]

This dataset was created for an R programming language class at Université de Technologie de Troyes. The course is a part of the university's big data masters program. The instructor published the data to allow others exploring data-science to experiment on the dataset and get feedback on improving the dataset. The columns names and values are not anonymized. Furthermore, descriptions of each column's meaning are provided. The dataset has a binary loss variable representing if a home had a claim in the past three years which could allow for training a binary loss classifier. It is unclear if there is an insurance company involved with this dataset, based on the given information.

travel [37]

This data is derived from a Singaporean travel insurance entity. It is was created for the task of Binary Loss Prediction. It has a relatively low number of columns, but the columns have clear descriptions and names. The column values are not anonymized.

car portos [43] This dataset was released by Porto Seguro, one of the largest insurance providers in Brazil. It was released in a Kaggle competition to predict binary loss for automobile insurance claims. The categorical column values are changed as they are all binary (0, or 1), and column names are fully anonymized. The column

names have a naming convention to indicate groups which have a relationship.

emicen auto & **emicen rail** [16, 17] These datasets were provided by Emicen to demo their EmicenPattern data-analysis tool. Both datasets include a loss column and provide column names and column values that are not anonymized.

prudential life [25] This dataset was released by Prudential Life Insurance for a Kaggle competition. The goal for this dataset is to classify a client's risk automatically, rather than manually have employees do so. The target column is ordinal, has eight levels, and represents the risk category that the insurance company assigns to a client. The column values are normalized and column names are grouped into interpretable high-level groups with provided descriptions. For example, there are 41 columns labeled *Medical_History_1-41* which represent normalized variables related to a client's medical history.

Fraud 2 [18] This dataset was released on Kaggle for the task of fraud prediction. There are three tables in this dataset:

1. Employee Data - Data pertaining to employees working on the insurance claim.

2. Vendor Data - Data pertaining to the fraud investigation vendor that helped investigate the claim.

3. Claims Data - The data pertaining to the insurance client and their claim.

The dataset has interpretable column names. Although this dataset is released for fraud prediction, we are using it for loss prediction, and only using the Claims Data table.

databricks [13] This dataset was published by Databricks for a fraud-detection demo of their data-analysis product. The data has interpretable column names. Although this dataset is released for fraud prediction, we are using it for loss prediction.

Fraud 4 [44] This dataset was created for the task of fraud prediction. The source of the data is not specified. Although this dataset is released for fraud prediction, we are using it for loss prediction.

2.2 Dataset Statistics

We present properties and quantitative statistics describing each insurance dataset in Table 2.1. Explanations on what each column and column value represent are given in Table 2.2.

2.3 Dataset Challenges

In this section, we detail the challenging properties prevalent in insurance datasets.

- **Non-Gaussian distributions**: Several insurance columns, especially loss columns, generally have a non-Gaussian distribution. We ran a one-sample Kolmogorov-Smirnov test(kstest), with a p-value threshold of .005, over all of our continuous variable columns (columns that are not categorical or date time). For each column we took the kstest pvalue between the column sample data and a Gaussian distribution fitted to the column sample data, and found that 98.4% of the columns had a pvalue below the threshold, and thus follow a non-Gaussian distribution. This shows that most columns in our dataset are non-Gaussian.

- **Mixed Continuous and Discrete columns**: Some columns are mixed continuous and discrete random variables. We give an example in figure 2-1 where the column cannot be modeled as a common continuous random variable that assumes zero probability of repeating values (such as a Gaussian, Uniform, or Beta distribution). A better approach is to model this as a mixed continuous and discrete variable where values with very low frequencies are modeled with a common continuous distribution, while values with high probability mass are modeled discretely.

- **High dimensional categorical columns**: There are two types of high dimensional categorical columns:

  - Unique Valued Categorical columns. For these columns, every row (in the training and test data) has a unique value. Examples of these would be

name	url	Anon	MT	LE	PE	Rows	Cols	Cat	Cont	LZ	LU	LDtype	|Cat|	max (Cat)
allstate	url	FA	✗	✓	✗	314k	133	116	14	0	158224	float	10	349
home	url	R	✗	✓	✓	256k	66	20	42	168k	3	cat	545	5475
travel	url	R	✗	✓	✗	63k	11	4	7	62k	2	cat	680	1139
car portos	url	FA	✗	✓	✗	1488k	60	32	27	574k	3	float	7	104
emicen auto	url	R	✗	✓	✓	9k	26	17	8	0	8041	float	7	59
emicen rail	url	R	✗	✓	✗	10k	19	13	6	0	10321	float	74	377
prudential life	url	R	✗	✗	✗	79k	129	109	19	6k	9	float	554	59381
Fraud 2	url	R	✗	✓	✓	10k	38	20	10	0	107	int	2206	10000
databricks	url	R	✗	✓	✓	1k	39	21	18	0	763	int	103	1000
Fraud 4	url	R	✗	✓	✗	34k	29	14	15	0	17389	int	292	2552

Table 2.1: Statistics for public datasets. See Table 2.2 for a description of each column and its values.

Column Name	Definition
Anonymity (Anon)	FA - Fully Anonymous dataset R - Raw Interpretable Data SD - Interpretable Synthetic Data
Multi-Table (MT)	Whether the dataset has multiple tables.
Loss Exists (LE)	Whether the dataset have a Loss column.
Premium Exists (PE)	Whether the dataset has a Premium column.
Rows	The number of rows.
Cols	The number of columns.
Cat	The number of categorical columns.
Cont	The number of continuous columns.
Loss Zero (LZ)	The number of rows with Zero Loss (or minimum ordinal risk for the prudential life dataset).
Loss Unique (LU)	The number of unique values in the Loss column.
Loss Data Type (LDtype)	Datatype of loss column. Value of cat means column is categorical.
Average Number of Categories (|Cat|)	The average number of categories in each categorical column.
Max Categories (max (Cat))	The maximum number of categories in all categorical columns.

Table 2.2: Column name and meaning dictionary for table 2.1

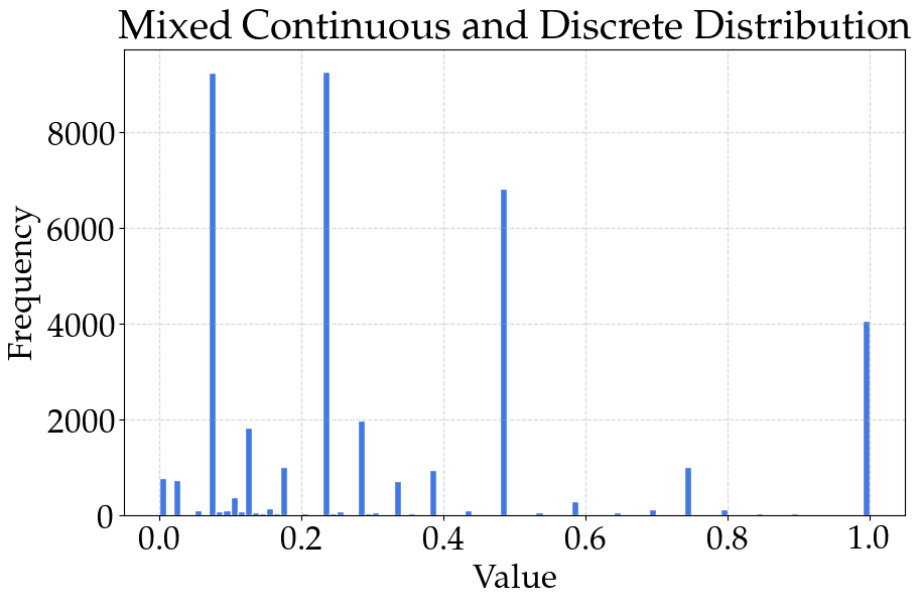


Figure 2-1: This figure is a 100 bin histogram of the PRODUCT_INFO_4 column in the `Prudential_Life` dataset. This dataset has $41,566$ samples however this column only has $1,085$. The most frequent value in this column is 0.076923077 which occurs with a frequency of 9234, which means it is around 22.5% of the values in this column.

street addresses, a full name, or a bank account number.

  - Non-Unique Categorical columns. These are categorical columns. These columns have inference benefits as they can help separate data, however in insurance datasets, these tend to have high carnality (a large number of unique values). This means that the typical one-hot encoding can be intractable for some modeling tasks such as training Gaussian Copula Synthetic Data generators.

- **Complex Column Dependencies** Column dependencies between features in

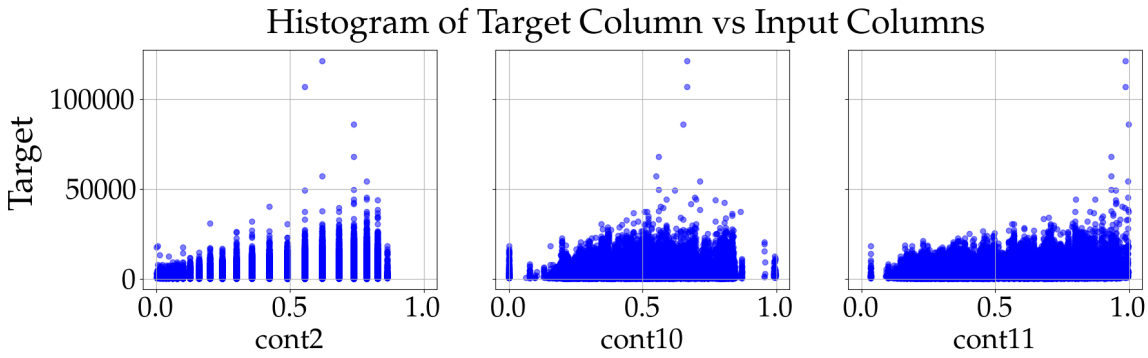


Figure 2-2: As examples of complex column dependencies, we plot bivariate scatter plots where the Y-axis is the target column "loss" from the `Allstate` dataset and the X-axes are the columns "cont2", "cont10", and "cont11" respectively from the `Allstate` dataset.

Insurance data are complex. For example in figure 2-2 we see there is heteroskedasticity (changing target variance as we change a predictive feature value) and nonlinear relationships in the `Allstate` dataset between the target column "loss" and the columns "cont2", "cont10", and "cont11". Models that make strong assumptions on the column relationships will likely fail to capture the complex interactions between variables.

Chapter 3

Insurance Prediction Problems and Our Machine Learning Pipeline

In this chapter we define:

- The metrics we use to measure prediction performance.
- Our Machine Learning pipeline.

3.1 Measuring Prediction Performance with Gini Score

To measure prediction performance for Loss prediction, we use the Gini score. We now give a clear description of the Gini score as this metric is not commonly used in ML literature and is a unique measure of model performance.

Gini Score

The Gini Score is a metric that indicates the model's ability to preserve overall rank, and thus have discriminatory power. It is often used for loss prediction. In the context of loss prediction, we are judging how effective the model is at differentiating between "bad" samples (with high loss) and "good" samples (with low loss). In this section we give intuition for the Gini Score, and provide some beneficial modifications to

Variable	Definition
UA	The unachievable area. It is cacluated as the area under the "Perfect Model".
A	This is the area between the "Lorenz Curve" and the "Random Model".
B	This is the area between the "Perfect Model" and the "Lorenz Curve".
Gini Score	is $\frac{A}{A+B}$.
y	Represents the actual loss for all clients. y_i represents the actual loss for the i^{th} client.
$\hat{y}$	Represents the predicted loss for all clients. $\hat{y}_i$ represents the predicted loss for the i^{th} client.
i	indexes insurance clients.
n	Is the number of insurance clients.
cum_y_i	Is the cumulative sum of the actual losses from client 1 to client i when ordering clients by $\hat{y}$ from low to high.

Table 3.1: Gini Score Notation Table

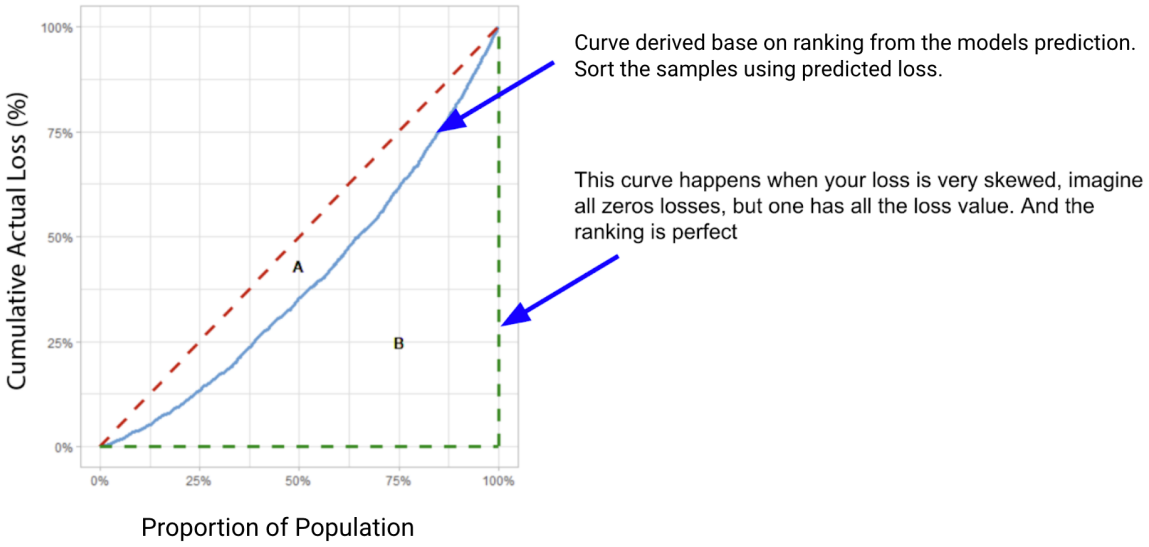


Figure 3-1: Diagram of the Lorenz Curve.

calculating it. Note that we provide notation table 3.1 for quick reference of variables used in this section.

Somers' Delta (Somers' D) is used to measure the ordinal relationship between two variables. In the context of loss prediction, let's say that we have two pairs (y_i, $\hat{y}_i$) and (y_j, $\hat{y}_j$), we are expecting either:

1. $y_i > y_j$ and $\hat{y}_i > \hat{y}_j$ or

2. $y_i < y_j$ and $\hat{y}_i < \hat{y}_j$

where y_i is the loss of sample i and $\hat{y}_j$ is the predicted loss of sample i.

Put simply, say a model predicts the samples' loss and orders them from lowest to highest. If this ranking matches the real-life ranking of samples by loss, the model is deemed perfect.

The way the Gini score is calculated is using the "Lorenze curve". We plotted the "Lorenze curve" in figure 3-1[1] and describe it below.

The blue line is the "Lorenz curve" that is constructed using the predicted loss ($\hat{y}$). Basically: all the samples are ordered by the $\hat{y}$ from low to high. Then you are able to plot every data point ($\frac{i}{n}$, cum_y_i) in the diagram with:

X-Axis $\frac{i}{n}$ denoting the cumulative population proportion of i clients. Since there are n clients, the proportion is $\frac{i}{n}$.

Y-Axis cum_y_i denoting the cumulative actual loss.

The intuition here is just to put every client that is deemed by the model to have a low loss (i.e., less risky) on the left of the x-axis and the high risk clients on the right.

The red line is known as "the Line of Equality" which is expected to be produced by a "random model". To explain it, let's imagine a scenario in which an insurance company has 100 clients and lost $300. That suggests for each client the insurance company has, it lost $\frac{300}{100} = 3$ dollars on average. So if we use a random model, i.e., a model which randomly assigns $\hat{y}$ to every client, the cumulative actual loss in percentage will always equal the proportion of population. In the example, let's say when the proportion of the population is 40%, the cumulative

[1]Note that some past work [20] use proportion of the insurance premium (where premium is the amount a client payed the insurance company for coverage) the as the X-axis, however not all datasets have premium columns so we use proportion of population instead. This is equivalent to assuming all clients have an equal premium.

actual loss should be expected to be 40% of the total actual loss, which is $120. The intuition here is that the random ranking of samples will lead to individual having an averaged loss of $3.

The green line is for a situation in which all the cumulative actual loss is in one sample (the last individual) and the model was also able to rank it perfectly - a.k.a when the actual samples were sorted based on the ranking of the model's predictions, the ranking was accurate. For example, an insurance company has 100 clients in total and lost $300 overall; now the green line means that for the first 99 clients, the insurance company lost $0, but for the last client, the insurance company lost all $300. This situation may not happen in reality, but modeling it provides a way to calibrate/normalize the upper bound Gini score values: That is, it is the hardest data for a model to have created a perfect ranking for, so it is useful to know whether the model could succeed in predicting this ranking.

The Gini Score is calculated as the ratio of:

- the area (A) between the red line (random model) and the blue line (Lorenz curve) to
- the area (A+B) between the red line and the green line. As a note, (A+B) is always equal to 0.5.

There are two problems when calculating Gini score that we address here. First is the *normalization factor* (aka $A + B$): The green line is not reachable given the data. To this end, we need to deduct the unreachable area from the denominator (A+B). To determine the unreachable area based on the given data, we first need to order all the samples by the ground truth loss and re-plot the Lorenz curve in this order, as done in figure 3-2. The area (UA) behind this Lorenz curve is the unreachable area that we should deduct from the original denominator. So the final denominator should be (0.5 - UA).

Here are our thoughts on using $A + B = 0.5$ as the normalization factor.

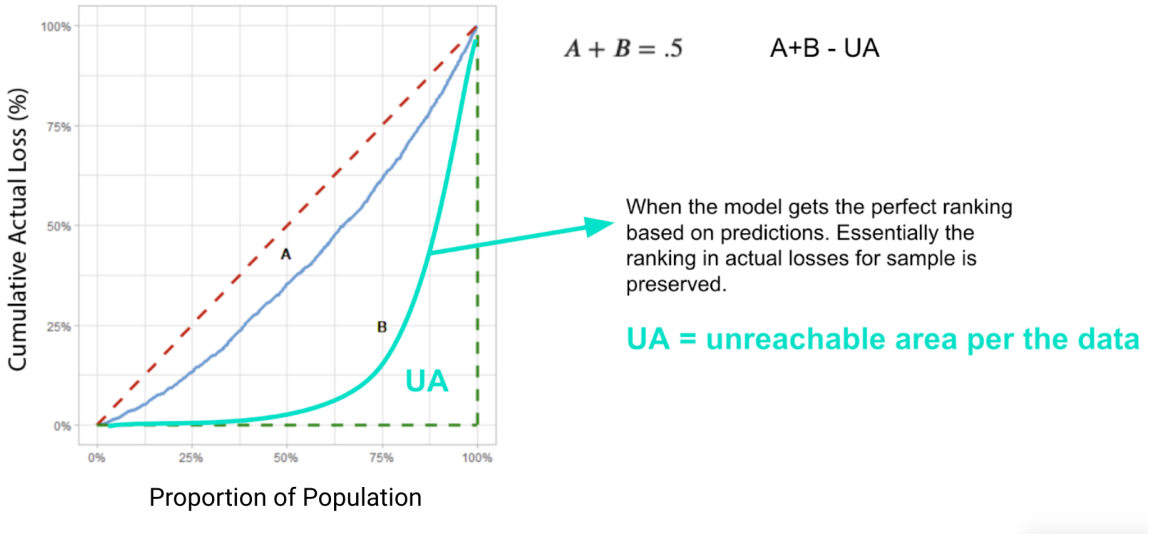


Figure 3-2: Diagram of Lorenz Curve using the corrected Perfect Model Curve.

When it is not a problem Comparison of two models, M_1 and M_2. Both either get divided by 0.5 or area calculated by using 0.5-UA. So the comparison still holds regardless of the denominator used as it is constant in both cases.

When it is a problem This has an effect on the pursuit of the best model. In a scenario where the predicted model is given the exact ranking of the loss, depending on how the actual ranking is (if the perfect model's Lorenze curve is closer to the random model line), we could give a score that is far less than 1 - implying that it may be possible to improve the prediction model, when actually it is not possible. This creates a problem by giving the impression that some predictive model could get a better Gini score when this is not true.

In this thesis we use $A + B = 0.5 - \text{UA}$ as the normalization factor to get a gauge of how close we are to the best ordinal ranking we can get on a given dataset.

The other challenge is actual calculation of the areas A and $A + B$. The area A can naively be computed using the rectangle rule[2]. It is calculated by the sum of the area of n rectangles, where the i^{th} rectangle has:

[2]The same procedure can be done to calculate the area $A + B$, but instead of sorting by the predicted loss, $\hat{y}$, you sort by the actual loss, y.

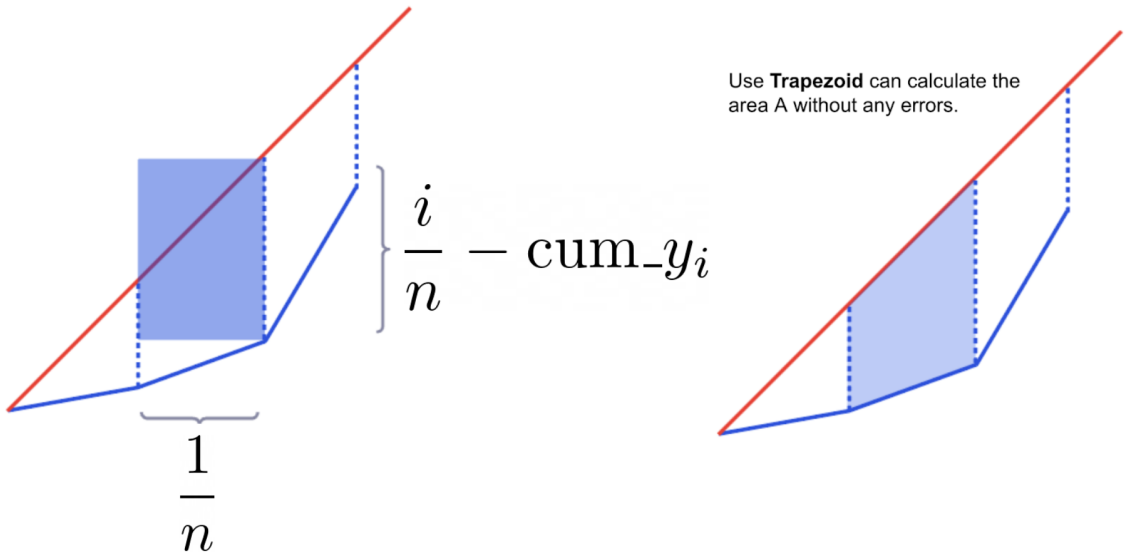


Figure 3-3: A naive implementation of the Gini Score would be to use the rectangle rule to approximate the area between curves, as shown on the left. However, this will lead to errors as you can see the shaded region, representing the rectangle rule approximation is not very accurate. Instead, one should use the trapezoid rule. In this problem it gives the exact area.

Height proportion of the population for i^{th} ranked sample (sorting with $\hat{y}$) minus the cumulative Actual loss $\text{cum\_}y_i$. The height is $\frac{i}{n} - \text{cum\_}y_i$.

Width Proportion of a single individual, which is $\frac{1}{n}$.

We propose using the Trapezoidal rule to calculate the area A (and $A + B$), as it is more accurate. In figure 3-3, we compare how the rectangle rule and trapezoid rule compute area. Theoretically, the Gini score score using the Somers' D measurement should take on a value from -1 to 1, with -1 being a perfect negative ordinal relationship and 1 being a perfect ordinal relationship (meaning the predicted order is exactly the same as the ground-truth order). However, using rectangle integration can lead to the resulting score out of the range [-1, 1], while using a trapezoid can obtain the most accurate value and guarantee the value range from -1 to 1. See the full algorithm for Gini score, using the trapezoid rule, in Appendix Section D.

3.2 Our Machine Learning Pipeline

We perform two stages of preprocessing in our pipeline. The first stage is general preproccessing. General preprocessing converts raw input data into processed data on which we can train Synthetic Data generators. SD generators generate synthetic Processed Data. We call the second stage ML Model Preprocessing. This stage takes in processed data and converts it into machine learning model input data that we feed direcly to our ML model. See figure 3-4 for a visual of the preprocessing Pipeline we described.

Figure 3-4: Highlevel Preprocessing Pipeline

We describe our General Preprocessing and ML Model Preprocessing pipelines in detail below:

General Preprocessing

We perform General Preprocessing on the raw insurance data as follows:

1. Parse date-time variables into python datetime format
2. Drop ID columns
3. Drop rows with missing target column values
4. Imputing missing values in non-target columns
5. Data is shuffled before splitting into 30% test data and 70% training data.
6. Of the training data, withhold 10% as validation set data (used for XGBoost early stopping).

ML Model Preprocessing

For our ML Model Preprocessing we do the following:

1. Encode datetime variables as (year, month, date)

2. Encode categorical variables using one-hot-encoding

3. Normalize continuous variables (including the date time variable encodings) by subtracting mean and dividing by standard deviation.

We use Xgboost regressor and classifier [11] for regression and classification tasks respectively. We tune the hyperparameters[3], documented in table 3.2, using tree-structured Parzen Estimator [5] in the Optuna library [1]. We use 10-fold cross validation in tuning. The XGBoost model is chosen for this task as gradient tree methods like it are the most common predictive models used in the insurance industry for tabular data.

Hyperparameter	XGBoost Parameter	Distribution (Range)
Tree method to use	`tree_method`	hist
Number of gradient boosted trees	`n_estimators`	100
Validation metric early stopping	`early_stopping_rounds`	100
L_1 Regularization term	`alpha`	`log-float`$(1e{-}6, 2)$
L_2 Regularization term	`lambda`	`log-float`$(1e{-}6, 2)$
Minimum required split loss	`gamma`	`log-float`$(1e{-}6, 64)$
Subsample ratio	`subsample`	`float`$(0.5, 1)$
Max tree depth	`max_depth`	`integer`$(2, 20)$
Learning rate	`learning_rate`	`log-float`$(1e{-}3, 0.1)$
Subsample ratio of columns	`colsample_bytree`	`float`$(0.3, 1)$

Table 3.2: XGBoost Hyperparameters

For all experiments involving the generation of synthetic data, we repeat the experiment ten times (generating ten different synthetic data batches) and report the average and standard deviation of the Gini Scores returned from each trial.

[3]See this guide to learn more about Xgboost hyperparameters

We only run our experiments on a subset of the datasets from section 2. In table 3.3, we present the baseline Gini Scores for loss prediction (with no Data Augmentation) on these datasets.

Dataset	Databricks	Emicen Auto	Emicen Rail	Fraud 2
ML Score	0.626	0.953	0.866	0.908

Table 3.3: Baseline Results

Chapter 4

Evaluating and Enhancing Copula Based Synthetic Data Generation

In this thesis, we experiment with using a Data Augmentation workflow to leverage Synthetic Data and improve ML performance. There are several Synthetic Data generators that learn the probability distribution of columns of tabular data and allow the sampling of new rows from the model. GAN models ([50] [57], [39]) are very popular in the space of tabular generator models; however, they are not interpretable and thus offer little help in pushing forward the understanding of data augmentation for tabular data. The Copula models can be interpreted more easily as they separate univariate column distribution modeling from modeling the joint distribution of variables.

Gaussian Copulas are popular, but make assumptions on the joint distribution of the data. Gaussian Copulas can be made more flexible, with the goal of handling complex column dependencies, by Segmenting or using a generalization of them such as the Gaussian Mixture Copula model. With these generalizations in mind, Copulas are a good option for developing Synthetic Data generators for Data Augmentation and analyzing why they do (or do not) help. In this chapter, we review general Copula theory, define the Gaussian Copula model, analyze it's failure modes, and then introduce two models to overcome these failures: the Segmented Gaussian Copula and the Gaussian Mixture Copula model.

4.1 Background

We provide brief background on the theory of Copulas, see [15] for a deeper dive into Copula theory.

4.1.1 Copula Theory

Variable	Definition
$f(\mathbf{D}$	Real data PDF. the CDF for the real data is $F(\mathbf{D})$.
$c(\mathbf{U})$	Copula PDF for GMCM. The CDF is $C(\mathbf{U})$.
$\mathbf{\Omega}$	Is defined as $\mathbf{\Omega}_j = \Phi_j^{-1}(\mathbf{U}_j)$. $\mathbf{\Omega}$ is chosen to be a random variable we can easily sample from.
Σ	Is a covariance matrix.

Table 4.1: Gaussian Copula Notation Table

In this section, we review some results in Copula Theory. We provide summary notation table 4.4 for this section. A Copula C is a joint cumulative distribution function defined on random variables $\mathbf{U}_i$ with uniform marginals, denoted:

$$C(\mathbf{U}_1, \ldots, \mathbf{U}_2).$$

Sklar's theorem [45] tells us that any arbitrary cumulative distribution $F(\mathbf{D}_1, \ldots, \mathbf{D}_m)$ can be uniquely represented by a Copula after performing a CDF transform $F_j(\mathbf{D}_j)$ on each of the marginals $\mathbf{D}_j$:

$$F(\mathbf{D}_1, \ldots, \mathbf{D}_m) = C(F_1(\mathbf{D}_1), \ldots, F_m(\mathbf{D}_m)).$$

Deriving the PDF, $f(\mathbf{D})$ in terms of the Copula PDF $c(\mathbf{U})$ corresponding to the Copula CDF $C(\mathbf{U})$ we get the following relationship between $f(\mathbf{D})$ and $c(\mathbf{U})$:

$$f(\mathbf{D}_1, \ldots, \mathbf{D}_m) = c(\mathbf{U}_1, \ldots, \mathbf{U}_m) \prod_{j=1}^{m} f_j(\mathbf{D}_j). \tag{4.1}$$

Copulas are attractive for modeling the probability density function, $f(\mathbf{D})$, of tabular data because, as shown in equation 4.1, they separate modeling the marginal distributions of data $f_j(\mathbf{D}_j)$ from estimating the column dependencies of the data $c(\mathbf{U})$. Copulas are also popular for generating synthetic data as Copula densities, $c(\mathbf{U})$, are generally chosen such that is easy to sample data from them.

Forward Transform

$$D \rightarrow U \rightarrow \Omega$$

Reverse Transform

$$\Omega' \rightarrow U' \rightarrow D'$$

Figure 4-1: Visual depiction of the **forward transform** and **reverse transform**.

Most Copula models, like the Gaussian Copula model, are defined in terms of a variable, $\mathbf{\Omega}$, that is easy to sample. $\mathbf{\Omega}$ is assumed to have a tractable invertible transformation to a uniform distribution such that $\mathbf{U}'_j = \Phi_j(\mathbf{\Omega}'_j)$. We can then transform this data to the original marginal distributions of $\mathbf{D}$. We do this by applying the inverse marginal CDF function for $\mathbf{D}$ such that $\mathbf{D}'_j = F_j^{-1}(\mathbf{U}'_j)$. Thus we can can sample $\mathbf{\Omega}'$ and transform it into Synthetic Data samples $\mathbf{D}'$. We refer to the transformation from $\mathbf{\Omega}'$ to $\mathbf{D}'$ as the **reverse transform**.

We additionally define the **forward transform** from $\mathbf{D}$ (real data) to $\mathbf{\Omega}$. This is performed as follows: starting with $\mathbf{D}$, we get that $\mathbf{U}_j = F_j(\mathbf{D}_j)$ and $\mathbf{\Omega}_j = \Phi_j^{-1}(\mathbf{U}_j)$. Notice that $\mathbf{U}$ and $\mathbf{\Omega}$ refer to variables transformed from samples of $\mathbf{D}$ in the **forward transform**. $\mathbf{D}'$ and $\mathbf{U}'$ refer to variables transformed from samples of $\mathbf{\Omega}'$ in the **reverse transform**. We give a visual of these transforms in figure 4-1.

Two common Copula models in the Copula literature are the Gaussian Copula

and various forms of vine Copulas [41, 35]. In this thesis, we focus on using Gaussian Copulas and the mixture Gaussian Copula model [6, 27, 47] to model the probability distribution of our data and experiment with various tabular data augmentation techniques. Next, we define the Gaussian Copula.

4.2 The Gaussian Copula (GC)

Here we define the Gaussian Copula [15, 41]. For a Gaussian Copula, we assume $\mathbf{\Omega}$ follows a multivariate Gaussian distribution with 0 mean and covariance matrix Σ, such that $\Sigma_{j,j} = 1 \forall j$. We denote the CDF and PDF as $\Phi_\Sigma(\mathbf{\Omega})$ and $\phi_\Sigma(\mathbf{\Omega})$. The marginal CDFs are defined as $\Phi_j(\mathbf{\Omega}_j) = \Phi(\mathbf{\Omega}_j) \forall j$, where Φ is the CDF of a univariate standard normal distribution. The Gaussian Copula CDF is

$$C_\Sigma(\mathbf{U}_1, \ldots, \mathbf{U}_m) = \Phi_\Sigma(\Phi^{-1}(\mathbf{U}_1), \ldots, \Phi^{-1}(\mathbf{U}_m)),$$

and PDF is

$$c_\Sigma(\mathbf{U}_1, \ldots, \mathbf{U}_m) = \phi_\Sigma(\Phi^{-1}(\mathbf{U}_1), \ldots, \Phi^{-1}(\mathbf{U}_m)).$$

We can fit the Gaussian Copula given real data $\mathbf{D}$. The **forward transform** is used to transform the $\mathbf{D}$ to $\mathbf{\Omega}$ by fitting a univariate CDF $F_j(\mathbf{D}_j)$ to $\mathbf{D}$, and get $\mathbf{U}_j = F_j(\mathbf{D}_j)$. Then we get $\mathbf{\Omega}_j$:

$$\mathbf{\Omega}_j = \Phi^{-1}(\mathbf{U}_j).$$

we can calculate the parameters for the sampling distribution, Σ, as follows:

$$\Sigma_{i,j} = \text{covariance}(\mathbf{\Omega}_i, \mathbf{\Omega}_j).$$

By sampling from the multivariate normal distribution with covariance Σ, we get samples $\mathbf{\Omega}'$ which we can **reverse transform** to have SD $\mathbf{D}'$. Next, we analyze the failure modes of the Gaussian Copula

Dataset	Properties
Positive Covariance Gaussian Copula Data	$\Omega_1, \Omega_2 \sim$ Gaussian Copula with covariance $\Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$
Zero Covariance Gaussian Copula Data	$\Omega_1, \Omega_2 \sim$ Gaussian Copula with covariance $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Negative Covariance Gaussian Copula Data	$\Omega_1, \Omega_2 \sim$ Gaussian Copula with covariance $\Sigma = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}$

Table 4.2: This table summarizes bivariate example datasets that are used to show what kinds of data the Gaussian Copula can successfully model.

4.3 Gaussian Copula Failure Modes

In this section, we discuss the failure modes of the Gaussian Copula, as well as the types of data that it can correctly model. For this purpose, we introduce several simulated datasets that allow us to analyze what kinds of distributions a Gaussian Copula can and cannot model. We then show how the Gaussian Copula's failure models can be overcome with Gaussian Mixture Copula Models. First, we define example datasets in table 4.2 and 4.3. We then provide plots of the data. Finally we analyze the failure modes of the Gaussian Copula on the example datasets, and motivate the use of Gaussian Mixture Copula Models and Segmented Gaussian Copula Models to overcome them.

Dataset	Properties
Cross Data	$U_1, U_2 \sim$ bivariate uniform distribution with covariance $\Sigma = \left[\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right]$
Three Mixture Data	$D_1, D_2 \sim$ bivariate Gaussian Mixture Model with covariance $\Sigma = \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix}\right]$

Table 4.3: This table summarizes bivariate example datasets that are used to show the Gaussian Copula failure modes.

4.3.1 Gaussian Copula Failure Mode Analysis Datasets

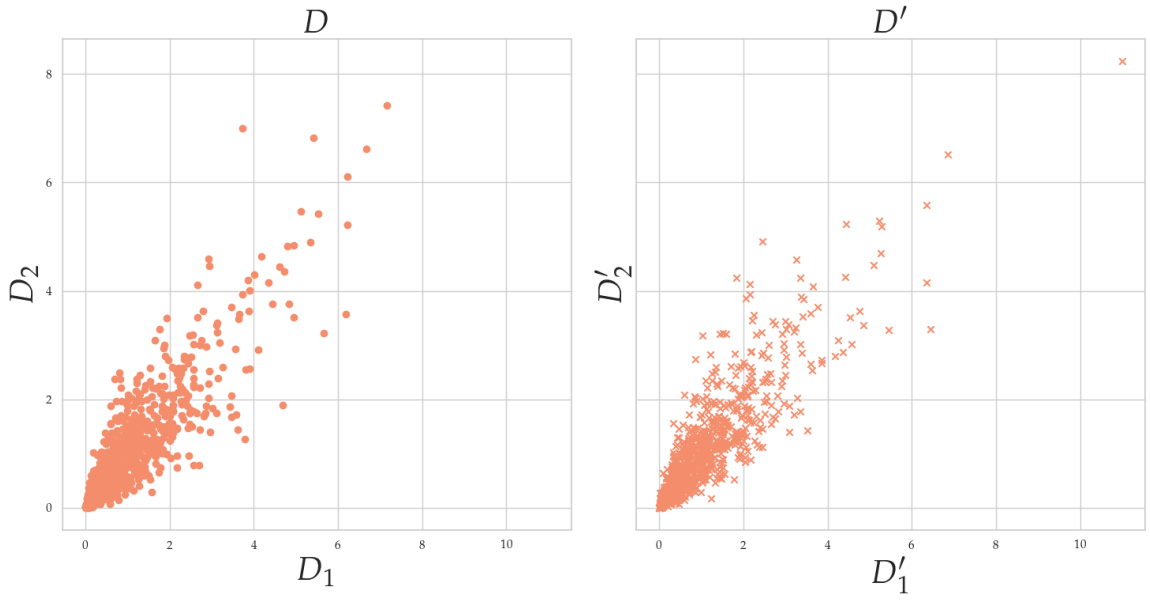


Figure 4-2: Plot of Positive Covariance Gaussian Copula Data. D is real data and D' is data sampled from a fitted Gaussian Copula.

Positive Covariance Gaussian Copula Data This dataset (plotted in figure 4-2) is modeled well by a Gaussian Copula. The data has exponential distribution

marginals, and the learned covariance for this Gaussian Copula is:

$$\Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}.$$

Thus the covariance between Ω_1' and Ω_2' is a high positive value, 0.9.

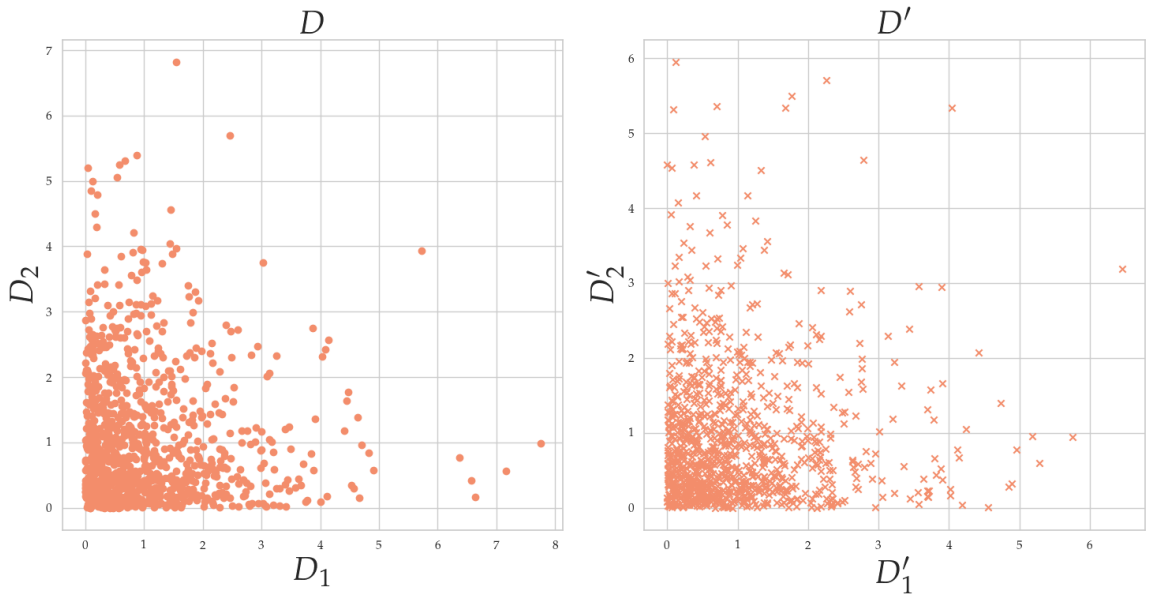


Figure 4-3: Plot of Zero Covariance Gaussian Copula Data. D is real data and D' is data sampled from a fitted Gaussian Copula.

Zero Covariance Gaussian Copula Data This dataset (plotted in figure 4-3) is modeled well by a Gaussian Copula. The data has exponential distribution marginals, and the learned covariance for this Gaussian Copula is:

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Thus the covariance between Ω'_1 and Ω'_2 is zero.

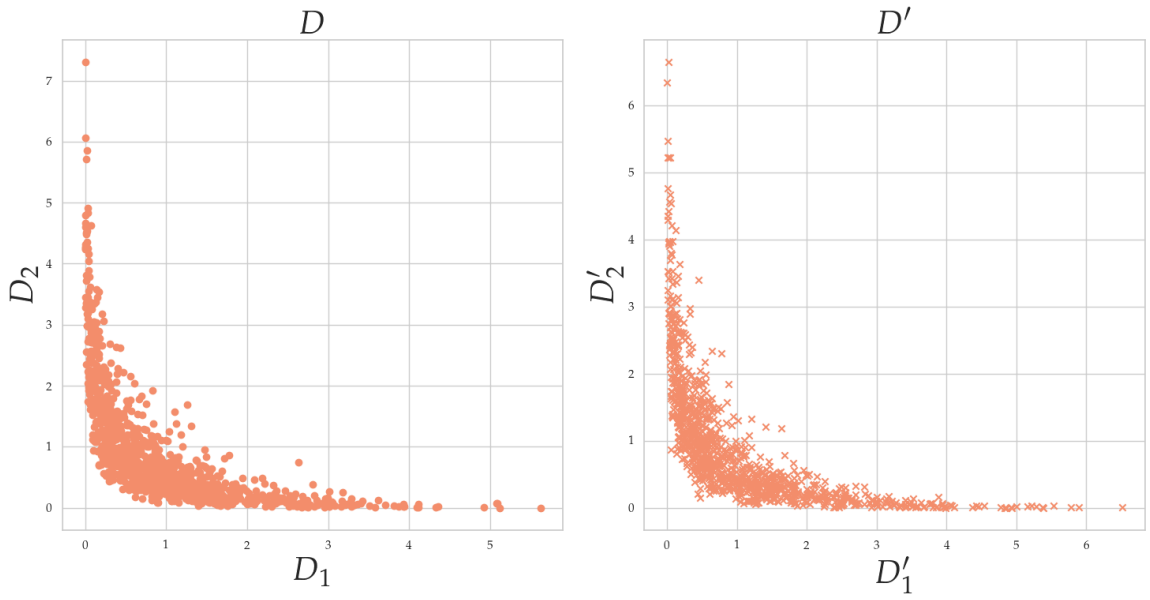


Figure 4-4: Plot of Negative Covariance Gaussian Copula Data. D is real data and D' is data sampled from a fitted Gaussian Copula.

Negative Covariance Gaussian Copula Data This dataset (plotted in figure 4-4) is modeled well by a Gaussian Copula. The data has exponential distribution marginals, and the learned covariance for this Gaussian Copula is:

$$\Sigma = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}.$$

Thus the covariance between Ω'_1 and Ω'_2 is -0.9.

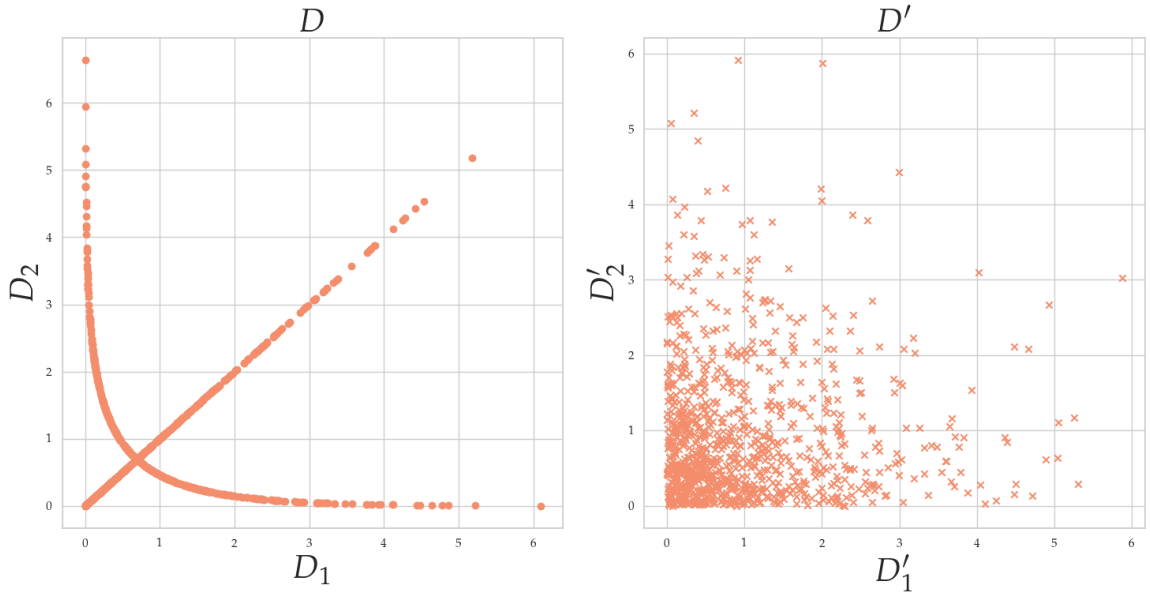


Figure 4-5: Plot of Cross Data. D is real data sampled from the Cross Data and D' is data sampled from a fitted Gaussian Copula.

Cross Data We create a cross shaped distribution and plot D and D' in figure 4-5.

To create the data, we sample U_1 from a uniform distribution and define $\forall i$

$$U_{i,2} = \begin{cases} U_{i,1} & \text{with probability } 0.5 \\ -U_{i,1} & \text{otherwise} \end{cases}$$

We then assume D_1 and D_2 follow exponential distributions with parameter $\lambda = 1$ and apply the inverse CDF of the exponential distribution to each U_j to get D. We get Ω by applying the Gaussian Copula **forward transform**. We then can fit a Gaussian Copula to Ω and obtain Ω' by sampling from the fitted Gaussian Copula. D' and U' are calculated from Ω' through the **reverse transform**.

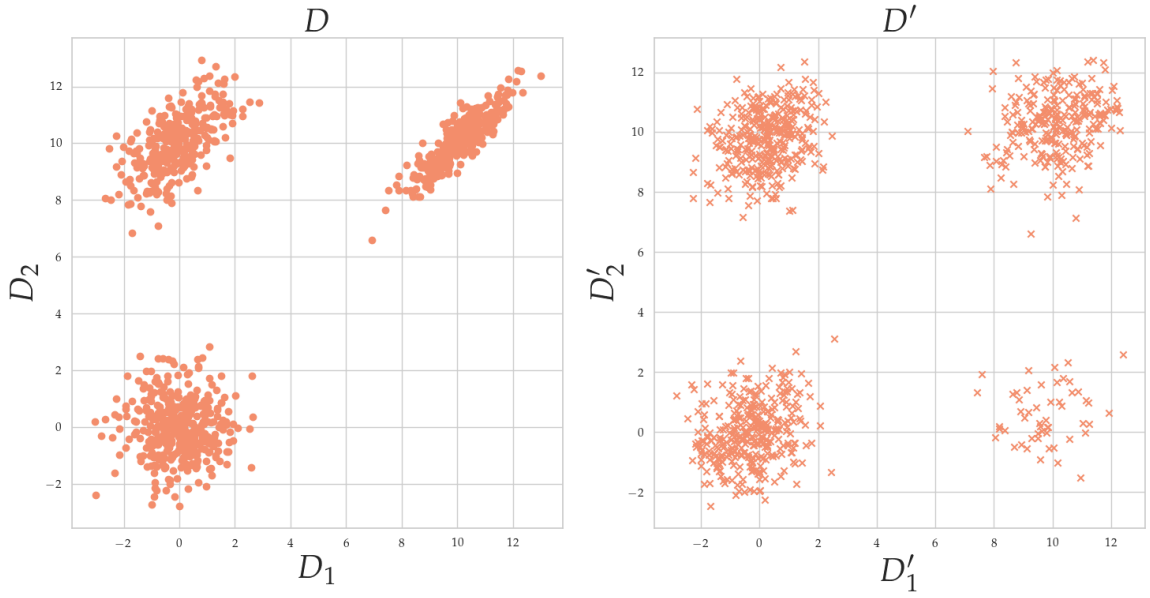


Figure 4-6: Plot of Three Mixture Data. The top row represents the **forward transform**. D is real data sampled from the Cross Data and D' is data sampled from a fitted Gaussian Copula.

Three Mixture Data We create a three mixture Gaussian Mixture Model and sample D from it. The mixture model is parameterized as follows:

- Mixture probabilities: $[0, .9, .6]$
- Mixture Means: $[(0, 0), (10, 10), (0, 10)]$
- Mixture Covariances:

$$\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}, \begin{bmatrix}1 & 0.9\\ 0.9 & 1\end{bmatrix}, \begin{bmatrix}1 & 0.6\\ 0.6 & 1\end{bmatrix}\right]$$

We can then use the **forward transform** to get U and Ω. After fitting a Gaussian Copula on Ω we can sample Ω' from the Copula and use the **reverse transform** to calculate U' and D'. We plot D and D' in figure 4-6.

4.3.2 Let's see how Gaussian Copula did on Simulated Datasets

Gaussian Copulas have several limitations. They can only provide symmetric and elliptical modeling of data. By only modeling covariance, we can only model linear

dependence between the Copula and sampling variables $\mathbf{U}$ and $\mathbf{\Omega}$. Note that this relationship can become nonlinear after transforming to $\mathbf{D}'$, because of the marginal inverse CDF transform, $F_j^{-1}(\mathbf{U})$. Some types of data are modeled well by Gaussian Copulas, as observed in figures 4-2, 4-3, and 4-4.

However, the Gaussian Copula assumptions made on the distribution of $\mathbf{U}$ and $\mathbf{\Omega}$ fail on more complex datasets. We try a more challenging example with the Cross Data. The plots of real and sampled data are in figure 4-5. We can see that the scatter plot for D' and D is completely different. This is because the Gaussian Copula only models the covariance, and it records a covariance of 0 between Ω_1 and Ω_2, causing D_1 and D_2 to have zero covariance after the **reverse transform**. We could instead fit a Mixture of Gaussian Copulas, fitting the data with a positive covariance to one mixture and the data with a negative covariance to the other mixture, which would properly model the data. This is an example where a generalization of the Gaussian Copula for modeling mixtures can potentially model these more complex distributions.

Another challenging dataset is the Three Mixture Data. The plots of real and sampled data are in figure 4-6. We can see that the scatter plot for D' and D is completely different. We see that a fourth mixture is hallucinated in the bottom right of D'. Since the real data is a guassian mixture model, a Gaussian Mixture Copula Model would be able to correctly model this data.

For both the Cross Data and Three Mixture Data , the Gaussian Copula fails to recreate a similar distribution because the dependencies between the two variables are more complex than just their covariance once transformed to $\mathbf{\Omega}$. These failure modes of the Gaussian Copula beg for a generalization of the Gaussian Copula in order to model more complex data. In the next two sections, we introduce two models that attempt to generalize the Gaussian Copula and overcome these failures.

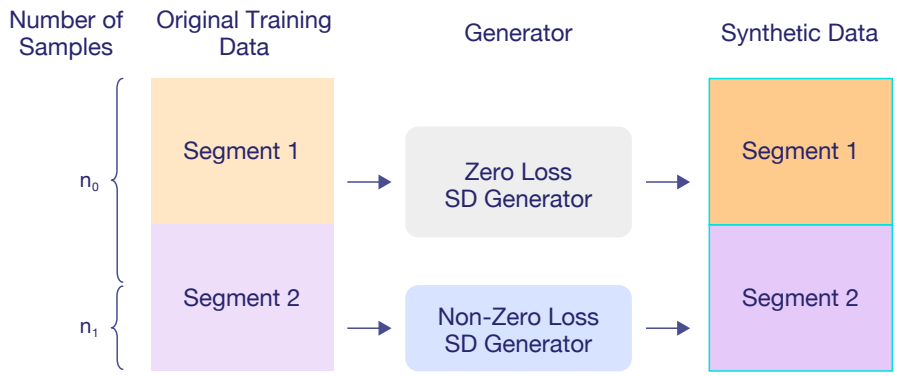


Figure 4-7: Segmenting training data into two pieces based on the target. We can assume we are working with a regression dataset, and segment 1 represents clients with zero loss while Segment 2 represents clients with non-zero losses. We train a Gaussian Copula Generator on each segment, and proportionately sample Synthetic Data from each generator.

4.4 Segmented Gaussian Copula

Here we define our own generalization of the Gaussian Copula that can potentially overcome the failure modes we defined in section 4.3. The Gaussian Copula model is very restrictive, but it is very quick to fit, so we can try an heuristic for splitting the data into mixtures that we individually fit a Gaussian Copula to. Specifically, we segment the data based on quantiles or categories. Then we fit a Gaussian Copula to each quantile to try to improve the modeling of column dependencies. As an heuristic, we segment the data over the target column by segmenting it into two pieces (as depicted in figure 4-7) corresponding to the two categories if it is a binary classification task, or two quantiles (split at the median) if it is a regression task. Without loss of generality, we denote the first piece of the segmented data D_1 and the second D_2. We fit a Gaussian Copula generator to each segment of the dataset. Denote p_1, p_2 as the empirical probability that a sample of the data is in segment 1 or segment 2. When sampling from the Segmented Gaussian Copula, we sample each row from Copula 1 with probability p_1 and from 2 with probability p_2. This procedure can be generalized to higher numbers of segments than just two. You can split a continuous target into segments for smaller quantiles or for a multi-class target we can split it into segments based on each class or groups of classes. We can then

apply this same methodology of fitting Copulas on each segment, and proportionately sample from them. The hope with segmenting is that we can approximately separate the data into segments that can be individually modeled by a Gaussian Copula.

4.5 The Gaussian Mixture Copula Model (GMCM)

To overcome the failure modes of Gaussian Copula models and avoid using heuristics like the Segmented Gaussian Copula model, we introduce the Gaussian Mixture Copula Model (GMCM). Gaussian Copula models have significant column dependency limitations as they only model covariance between columns. Segmented Gaussian Copulas try to resolve this by modeling different covariances between columns as we vary the segment, however, the creation of segments using user defined quantiles may not be optimal. We introduce one more generalization of Gaussian Copula models which we experiment with in this thesis, the Gaussian Mixture Copula Model (GMCM) [6, 27], which has a Gaussian Mixture Model (GMM) as its Copula density $c(\cdot)$. It can learn more complex column dependencies as the Copula density is a GMM rather than a single Gaussian. Since we learn this GMM, we expect it to better model the data than the segmented GMCM where we naively segment the data based on target quantiles or classes. We first describe the GMCM model, then present the algorithms for fitting its parameters, and finally discuss initialization methods for these initialization sensitive fitting algorithms.

4.5.1 Gaussian Mixture Copula Model Definition

Variable	Definition
r	Number of mixtures in GMM.
k	Mixture index $\in [1, \ldots, r]$.
Θ_k	Represents the parameters for a multivariate Gaussian. Equals the tuple (μ_k, Σ_k). Σ_k is the covariance. V_k is the cholesky decomposition of Σ_k. It is also denoted as $\sqrt{\Sigma_k}$. μ_k is the mean.
π_k	is the probability.
Θ	Represents the tuple $(\Theta_1, \ldots, \Theta_r)$. Fully parameterizes a GMM.
ψ	is the PDF of a multivariate normal distribution with parameters Θ_k. The CDF is Ψ.
$\phi^{\text{GMM}}(\mathbf{\Omega}; \Theta)$	PDF for GMM. CDF is $\Phi^{\text{GMM}}(\mathbf{\Omega})$.
$\phi_j^{\text{GMM}}(\mathbf{\Omega_j}; \Theta)$	Marginal PDF for GMM over j^{th} column. CDF is $\Phi_j^{\text{GMM}}(\mathbf{\Omega_j}; \Theta)$
$C^{\text{GMM}}(\mathbf{U}; \Theta)$	Copula CDF for GMCM. PDF is $c^{\text{GMM}}(\mathbf{U}; \Theta)$.
τ	Maximum number of epochs for AD and PEM GMCM algorithms.
t	Is the current epoch number during training. We index variables that are updated over epochs using the notation $\Sigma_k^{(t)}$ to represent the value of Σ_k after epoch t.
Z	Represents the latent variables for the PEM GMCM algorithm. $Z \in R^{n \times r}$ and $Z_{i,:}$ represents a categorical distribution over the clusters data point $D_{i,:}$ may belong to.
ϵ	Is the convergence threshold. If loss over two PEM GMCM epochs changes by less than ϵ, then training is stopped early.
ζ	Is the number of noise columns added to a dataset. Noise columns are columns that follow an independent normal distribution.

Table 4.4: GMCM Notation Table

In this section we define the Gaussian mixture Copula model [47, 27, 6, 26]. See notation table 4.4 for quick reference of the variables used.

$$\Psi(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta_k) \text{ and } \psi(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta_k)$$

are the CDF and PDF respectively of a multivariate normal distribution with parameters $\Theta_k = (\mu_k, \Sigma_k)$ where μ_k represents the mean and Σ_k the covariance for mixture k. Ψ_j and ψ_j are the marginal CDF and PDF of Ψ and ψ. The PDF of a GMM with r mixtures is

$$\phi^{\text{GMM}}(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta) = \sum_{k=1}^{r} \pi_k \psi(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta_k)$$

where π_k represents a scalar weight for each mixture. The CDF of the GMM is:

$$\Phi^{\text{GMM}}(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta) = \sum_{k=1}^{r} \pi_k \Psi(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta_k).$$

The j^{th} dimension marginal CDF and PDF of the GMM are denoted as $\Phi_j^{\text{GMM}}(\mathbf{\Omega}_j; \Theta)$ and $\phi_j^{\text{GMM}}(\mathbf{\Omega}_j; \Theta)$ respectively where $\Phi_j^{\text{GMM}}(\mathbf{\Omega}_j; \Theta) = \mathbf{U}_j$[1]. Thus $\Phi_j^{\text{GMM}}(\mathbf{\Omega}_j; \Theta) = \sum_{k=1}^{r} \pi_k \Psi_j(\mathbf{\Omega}_j; \Theta_k)$ and $\phi_j^{\text{GMM}}(\mathbf{\Omega}_j; \Theta) = \sum_{k=1}^{r} \pi_k \psi_j(\mathbf{\Omega}_j; \Theta_k)$ The Gaussian Mixture Copula CDF is defined as

$$\begin{aligned} C(\mathbf{U}_1, \ldots, \mathbf{U}_m; \Theta) &= \Phi^{\text{GMM}}((\Phi_1^{\text{GMM}})^{-1}(\mathbf{U}_1; \Theta), \ldots, (\Phi_m^{\text{GMM}})^{-1}(\mathbf{U}_m; \Theta); \Theta) \\ &= \Phi^{\text{GMM}}(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta), \end{aligned}$$

and by taking the derivatives of both sides, the Copula PDF is

$$c(\mathbf{U}_1, \ldots, \mathbf{U}_m; \Theta) = \frac{\phi^{\text{GMM}}(\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_m; \Theta)}{\prod_{j=1}^{m} \phi_j^{\text{GMM}}(\mathbf{\Omega}_j; \Theta)}. \tag{4.2}$$

.

[1]Note that there is no closed form for the inverse marginal CDF $\Phi_j^{-1}(\cdot)$, so it must be estimated as discussed in appendix section B.

4.5.2 Fitting Gaussian Mixture Copula Models

Most algorithms for learning GMCM parameters maximize

$$\mathcal{L} = \prod_{i=1}^{n} c(U_{i,1}, \ldots, U_{i,m}; \Theta) = \prod_{i=1}^{n} \frac{\phi^{\text{GMM}}(\Omega_{i,1}, \ldots, \Omega_{i,m}; \Theta)}{\prod_{j=1}^{m} \phi_j^{\text{GMM}}(\Omega_{i,j}; \Theta)} \tag{4.3}$$

which is simply the empirical form of the Copula density from equation 4.2 and thus the likelihood of the data given the GMCM parameters. Maximizing this likelihood is very difficult, and in appendix B we discuss two methods[2], the Auto-Differentiation[27] and Pseudo Expectation Maximization[6] methods, for attempting to maximize this likelihood and fit a GMCM.

These methods are sensitive to initialization, so in the remainder of this section, we give a qualitative assessment of different initialization methods for GMCMs.

Initialization Methods

Both GMCM learning methods, AD and PEM are known to be sensitive to initialization and prone to reaching local optimas [6, 27]. It is most common to use KMeans or a random initialization in the current Gaussian mixture Copula literature, but these initialization methods do not perform well on high dimensional data with lots of random noise columns are added to the data. In this section, we attempt to improve the performance of AD and PEM GMCM by using a better initialization. We compare different initialization methods on the Cross Data and Three Mixture Data distributions we introduced in section 4.3, and visually inspect how well they cluster the CDF transformed data, U, into clusters such that a Gaussian Copula could model each individual cluster. This is a desirable initialization because the GMCM simply models each mixture as a Gaussian Copula. We find that clustering methods in conjunction with feature selection techniques provide a good initialization.

[2]Note that these methods are the state of the art for fitting GMCMs when $n < m$, if you happen to have data where $m \gg n$, you would want to try a different approach such as the HD-GMCM model [26].

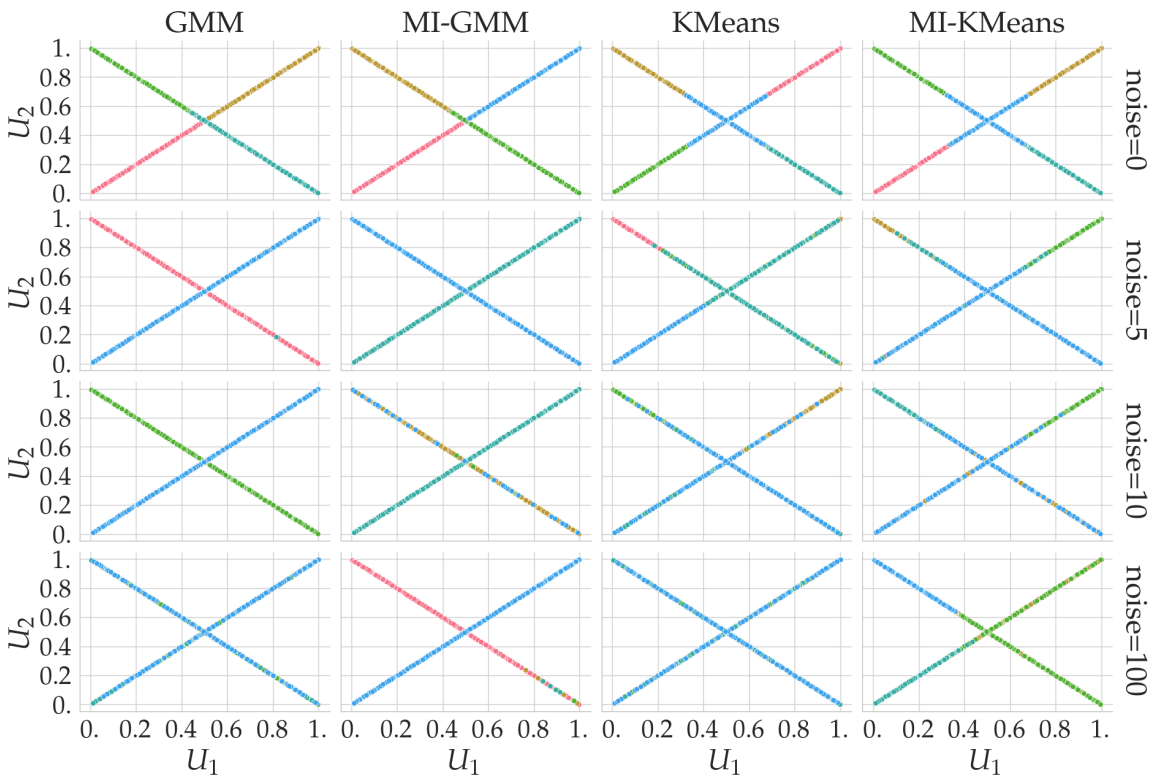


Figure 4-8: In this figure, we plot a plot distribution and the results of clustering the Cross Data from 4.3 with 5 clusters. The color of each data point corresponds to its cluster assignment. Each column we vary the clustering algorithms: GMM, MI-GMM, KMeans, and MI-KMeans. Each row varies the number of added noise columns to the data, from top to bottom row we have: [0, 5, 10, 100] added noise columns.

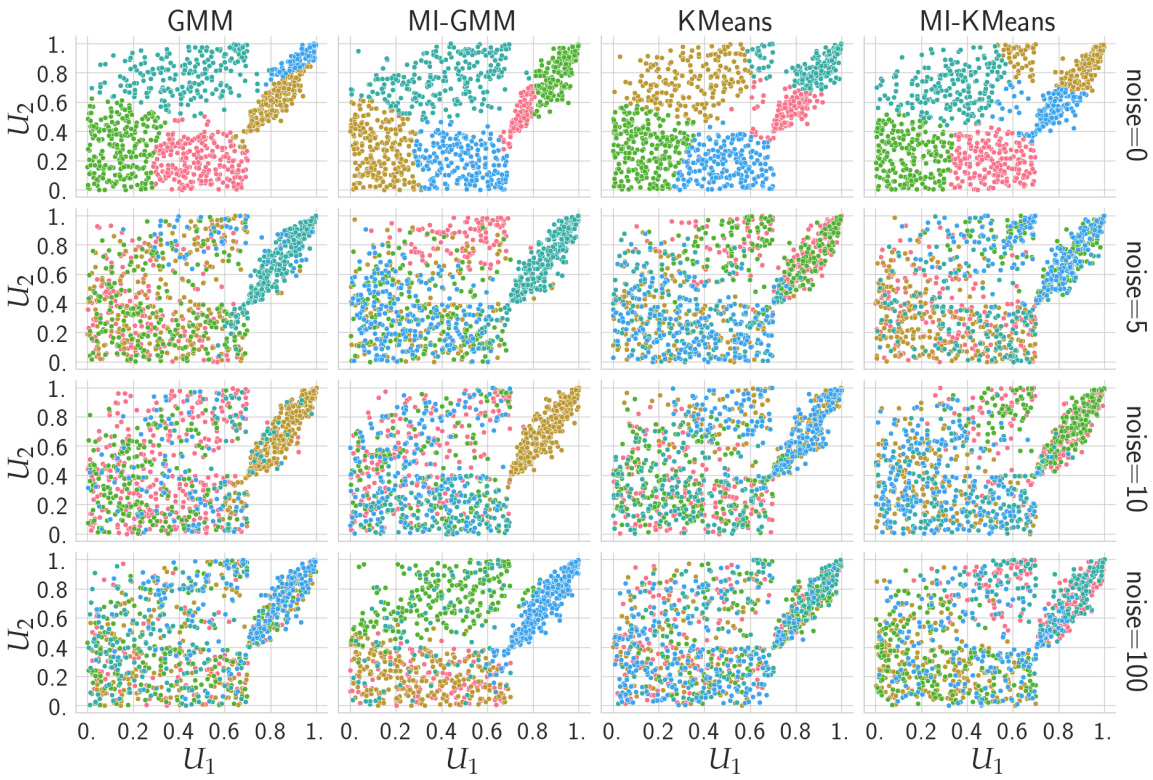


Figure 4-9: In this figure, we plot a plot distribution and the results of clustering the Three Mixture Data from 4.3 with 5 clusters. The color of each data point corresponds to its cluster assignment. Each column we vary the clustering algorithms: GMM, MI-GMM, KMeans, and MI-KMeans. Each row varies the number of added noise columns to the data: [0, 5, 10, 100].

We plot the clustering results of fitting 5 clusters to the Cross Dataafter adding $0, 5, 10$, and 100 noise columns (columns of data sampled from independent uniform distributions) to U. To clarify, if we add g noise columns, the data would have $g + 2$ columns in total, as two columns are the real bivariate data. We repeat this for the Three Mixture Data . We would expect a good clustering of the Cross Data distribution to cluster each of the four lines that make up the cross, and for the Three Mixture Data, we expect it to be clustered into three sections corresponding to each of the mixtures in the original GMM from which the data was sampled. For our Synthetic Data, as we add noise columns, GMM and Kmeans fail to give reasonable clustering assignments. Specifically, look at the last row of figures 4-8 and 4-9, columns GMM and KMeans, we see that for the Cross Data we are not getting any separation of the lines, and for the Three Mixture Data, we are getting random

assignments. Since we are modeling the data for data augmentation, we can assume that one of the non-noise columns is the target and perform feature selection before clustering to ensure columns the target depends on are prioritized when clustering. We calculate the mutual information of the target with each non-target column, and select the top v columns with the highest mutual information values. We then fit a GMM or KMeans clustering to the target along with these v columns, and denote this clustering approach MI-GMM and MI-KMeans respectively[3]. For GMM we set $v = 10$ and for KMeans we set $v = 5$. We plot visualizations of the outputs of these clustering methods as we vary the amount of noise columns in figure 4-8 for the Cross Data and in figure 4-9 for the Three Mixture Data. For all Synthetic Data experiments, we use the MI-GMM method as it almost always lead to the highest log-likelihood values for both PEM-GMCM and AD-GMCM fitting.

[3]In addition, we remove mixtures with extremely low probability, $p < 0.01$, to ensure every cluster has a significant amount of data assigned to it as we don't want a mixture in the GMCM model to overfit to a very small amount of data as this can cause numerical instability.

Chapter 5

Data Augmentation For Tabular Data

In this section, we first describe the Data Augmentation workflow. We then address some common questions pertaining to the application of data augmentation methods. Finally, we provide justification for the use of Data Augmentation in the tabular data setting to improve ML efficacy.

5.1 Data Augmentation Workflow

First, we define the goal of Data Augmentation and our workflow. The end goal is to improve machine learning performance for a specific task. We must first define some terms, and note that we provide notation table 5.1 for quick reference. We have a training dataset of real data, D_{train}, consisting of machine learning model inputs, x_{train} and targets y_{train}. We have a test dataset of real data, D_{test}, consisting of machine learning model inputs, x_{test} and targets y_{test}. Additionally, for Data Augmentation, we have some Synthetic Data sampled from a generator, G, that was trained on D_{train}, denoted D'_{train}, with machine learning model inputs and targets denoted as x'_{train} and y'_{train} respectively. Any number of samples can be generated and used to augment the real data before learning a machine learning model, M, for our goal. We also denote L as the learning algorithm used to train a model, M, on dataset D. So $M_D = L(D)$.

Original workflow: In part (a) of figure 5-1, we present the original workflow in which we train an ML pipeline without Synthetic Data. We train a model M on

Variable	Definition
D_{train}	Real training dataset, includes ML model input data and targets.
x_{train}	Real training ML model input data.
y_{train}	Real training ML model targets.
G	is a Synthetic Data generator model.
D'_{train}	Synthetic training dataset, includes ML model input data and targets.
x'_{train}	Synthetic training ML model input data.
x'_{train}	Synthetic training ML model targets.
D_{test}	Real testing dataset, includes ML model input data and targets.
x_{test}	Real testing ML model input data.
y_{test}	Real testing ML model targets.
M	Machine learning model.
M_D	Machine learning model trained on dataset D.
L	Learning algorithm for fitting a machine learning model M. $M_D = L(D)$.
$\hat{y}_{\text{test}}$	Target values predicted by ML model.

Table 5.1: Notation for Data Augmentation and SID Table

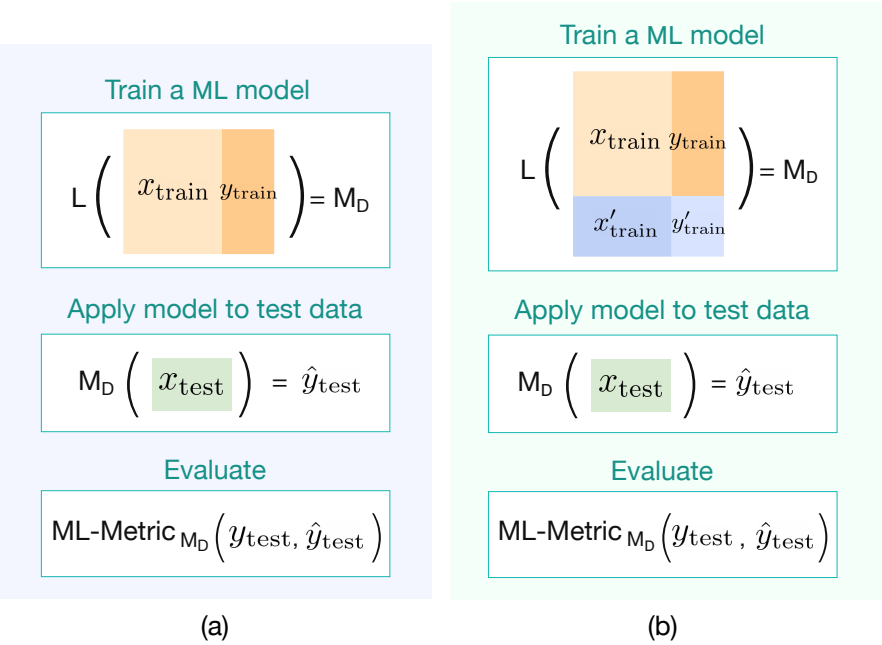


Figure 5-1: Comparison of machine learning original workflow (a) and Data-Augmentation workflow (b).

our real training data and then apply this model to testing data to get our predicted target, $\widehat{y}_{\text{test}}$. We then compute our ML model evaluation metric using the ground

truth target, y_{test}, and the predictions, $\widehat{y}_{\text{test}}$, as inputs

Synthetic Data augmentation workflow: In part (b) of figure 5-1 we present the Synthetic Data augmentation workflow, where we train an ML pipeline using real and Synthetic Data. We train a model M on our real training data concatenated with our Synthetic Data, and then apply this model to our testing data to get our predicted target, $\widehat{y}_{\text{test}}$. We then evaluate our ML evaluation metric using the ground truth target, y_{test}, and the predictions, $\widehat{y}_{\text{test}}$, as inputs.

5.2 Common Questions About Data Augmentation Workflow

Now that we have established the workflows, here are some common questions that arise when using a Synthetic Data augmentation workflow.

How do you know adding data helped? In the figure above, we notice that the test data used is the same in the original workflow and the workflow with the Synthetic Data. This enables an equivalent comparison and allows us to answer this question.

Do you synthesize the target column as well? Yes, to augment our training data we synthesize the target column as well.

Do you use synthetic data to evaluate the trained machine learning model? No, we do not. Since Synthetic Data generators are imperfect, samples generated by them may not perfectly follow the real distribution, and thus these samples should not be used to evaluate ML models in place of real data.

What data do you use to train the Synthetic Data generator? In this thesis we train our genrators on the same training samples that will be fed to the machine learning model. It is important not to train your generator on test data, as this will allow information from your test set to leak into your ML model via the Synthetic Data from your generator.

Next, we review related work in Data Augmentation to justify its application in

the tabular data setting.

5.3 Justification for Tabular Data Augmentation

In this section, we motivate Data Augmentation for tabular data. We reference work applying Data Augmentation in computer vision (CV), natural lanuage processing (NLP), and time series domains, all of which has led to improved ML performance. We justify that these methods help improve ML performance by regularizing ML models and helping them learn domain-specific invariances. Finally, we discuss how tabular, Copula-based, Data Augmentation can potentially help regularize ML models and help them learn a form of invariances for tabular data.

Recent work has shown significant improvements in ML performance in various domains through the use of Data Augmentation.

In computer vision, many Data Augmentation methods have improved ML performance. Some simple examples of Data Augmentation methods that have been widely used on tasks such as image classification or object detection are cropping images, applying rotations, and adding Gaussian noise[51, 52]. Deep learning methods for generating Data Augmentation have lead to improved performance on tasks such as image classification as well [51, 52]. For example, ACGAN [38] is a conditional generator that learns to take the image class as input. It will generate images for this class and has been used to improve performance when there is class imbalance [51].

Applying NLP to Data Augmentation is a bit more challenging as text data is modeled using discrete token representations. The CV field uses continuous data, so the NLP field has developed its own methodologies for augmenting text data. Many of these Data Augmentation strategies have successfully improved ML performance on various tasks[19]. For example, previous work has shown improved ML performance for several text classification methods using token level (tokens represent a word or part of a word) random perturbations, including random token inserts, deletions, or swaps[48, 19]. Additionally, deep learning methods for generating Data Augmentation have led to improved performance on tasks such as text classification [19].

Synthetic Data generative models are also commonly used. These models are generally conditioned on a label class. In [3], a label-conditioned generator was trained by fine-tuning GPT-2 [42] on the task-specific training data. The generator was then used to generate candidate examples per class. In [30], researchers showed how to effectively use pretrained models to generate Synthetic Data for Data Augmentation by conditioning the pretrained model on a label class.

In time series, Data Augmentation methods have been used to improve ML performance on various tasks. Cropping, flipping, and adding noise (jittering) to input data in the time or frequency domain are methods that have improved ML performance for time series forecasting[49]. Synthetic Data generative models also exist and have been used to improve performance on several time series tasks[24]. An example would be the use of GAN[23, 24] time series generative models for the tasks of predicting normal and abnormal heart behavior from electrocardiogram (ECG) time series data[53].

Past research has theoretically modeled Data Augmentation and has demonstrated that Data Augmentation improves ML models by

- making the model learn invariant representations of data[12, 10].
- regularizing the ML model[12, 10].

This begs the question of what these invariances and regularizations will be with our Copula models for tabular data. For our predictions, we lean on learnings from other domains. Regularization has a clear interpretation across domains of preventing ML models from overfitting; however, we must ask what the invariances are. In CV, augmenting with rotations of images will clearly lead to invariance to rotations. In NLP, augmenting with token perturbations with lead to invariance to these perturbations. In time series data, augmenting with jittered input data will lead to invariance to jittering. It is not obvious at first glance what invariance is being learned when augmenting with Gaussian Copula or Gaussian Mixture Copula Model Synthetic Data and whether this invariance is desirable for ML tasks.

We hypothesize that GCs and GMCMs provide Synthetic Data invariant to small

perturbations on the Copula random variables, $\mathbf{\Omega}$, and are regularized by the smooth distribution we assume $\mathbf{\Omega}$ follows. Data sampled from these Copula models is sampled from a Gaussian mixture model or a multivariate Gaussian distribution. These are very smooth simple distributions, and we are effectively regularizing models trained on this Synthetic Data to follow this Copula density distribution. Additionally, data sampled from GC and GMCM is perturbation-invariant in the space of Ω because a small perturbation over the value for a Gaussian or GMM will lead to a very small change in the probability distribution (as these distributions are smooth). Thus, the SD directs the model to learn a form of permutation invariance, where target values should be invariant to small perturbations in the space of $\mathbf{\Omega}$.

Several Synthetic Data generator models exist for tabular Data Augmentation. In the field of classification, SMOTE [9] and SMOTE-enc [36] allow for the generation of interpolated Synthetic Data samples from real datapoints of the same class; however, SMOTE is meant for continuous data, while our data has mixed continuous and discrete elements. Synthetic Data generative models, such as Gaussian Copula [41], ctgan [50], and ctabgan [57], can help model the probability distribution of the data while maintaining various levels of invariance in the marginal distribution and column dependencies. These models are often researched and evaluated for the purpose of privacy preservation (such as maximizing test set performance when only training on Synthetic Data) rather than Data Augmentation. Recent work[33] has compared empirical results of Data Augmentation methods SMOTE[9], GMM, and VAE[29] on imbalanced classification tasks with tabular data. No work, to our knowledge, has provided systematic controlled experiments to understand tabular Data Augmentation. In this thesis, we hope to contribute a systematic study of data augmentation for tabular data, along with a library for helping push forward innovation and understanding of Data Augmentation in the broader research community. We know that generative models have lead to large improvements in performance in CV and NLP, and we expect to achieve similar gains by applying Synthetic Data to tabular data. We detail our data augmentation methods and the SID library in the following section.

Chapter 6

Synthetic Insurance Data (SID) Testbed

The goal of this chapter is to provide a solution to **RQ5**, how we can use Synthetic Data in an ML workflow between entities and collaborators such that both benefit. We do this by designing the `SID` framework, an API which enables entities to collaborate on data augmentation workflows without sharing private data. In our case, we assume the entity is interested in Data Augmentation. First we give the motivation for creating the SID framework, and then we describe the SID API and software design implemented in the SID library. Note that we use the notation defined in table 5.1

6.1 SID as a Solution to Collaborative Data Augmentation

ML collaborative frameworks for developing prediction models should allow flexible model development. Additionally, while data augmentation for tabular data is not well understood in current research literature, it is a promising approach to improving machine learning performance that should be supported in the SID framework.

Many entity-collaborator interactions are hindered by stringent data regulations. Two solutions to this issue are:

- Collaborators may be required to use only the entity's machines to access the data; this limitation constrains the development tools available to collaborators, leading to less efficiency and options for collaborators. SID proposes that collaborators iterate on Data Augmentation methods using Synthetic Data on their local machines, so they can use whatever tools they are used to, and can then share their code and environment with the entity.
- The SID framework alternatively allows collaborators to use their own development environment and thus saves entities overhead costs associated with managing individual collaborators' access to internal systems. Furthermore, this framework saves Entities from sharing direct access to their data and in turn reduces their risk exposure while making it easier to start external collaborations. This allows for entities to scale their collaborations while simultaneously allowing validations of models on private data.

6.2 SID Workflow

This section gives a high level overview of how entities and collaborators can interact through the SID API to allow collaborators the ability to evaluate novel ML Data Augmentation methods on real data while also keeping sensitive real data private.

1. **The entity prepares real data and the ML task.** The entity aggregates a train and test dataset, D_{train} and D_{test}. The entity must also define an evaluation metric of interest. The entity may optionally provide a default machine learning model architecture to help collaborators get started.

2. **The entity creates a Synthetic Data model.** The entity uses `SID` to train a Synthetic Data model, G, on D_{train}. They generate a synthetic dataset D'_{train} and test set D'_{test} of sizes approximately equal to D_{train} and D_{test} datasets (not exactly equal to not expose the real dataset sizes) curated to have correlated inference accuracy with D_{train} and D_{test}.

3. **The entity Shares Synthetic Data and Generator.** The entity shares:

- API access for collaborators to submit their code. The entity runs the collaborators code on the real data and returns the resulting ML score.
- the ML score of interest.
- The synthetic datasets D'_{train} and D'_{test}

4. **Collaborators develop Models and Methods.** Collaborators experiment with different data augmentation methods on the synthetic datasets. They can design data augmentation methods. Collaborators locally design and build their approaches on the synthetic datasets, and then submit their code to the entity to validate on the real data.

5. **The entity validates sampled Synthetic Data for Data Augmentation.** The entity now runs the code submitted by the entity putting D_{train} as input to fit the model and then passes x_{test} to get the model's predictions, $\hat{y}_{test}$. The ML score is evaluated on these predictions and is shared with the collaborator. With this information we go back to step 4, as collaborators can iterate on their data augmentation method based on the feedback.

6.3 SID API

This section outlines how entities and collaborators interact through the SID API. We first describe the entity-side and collaborator-side of the API, then explain how the two sides interact through this API.

6.3.1 Entity API

Figure 6-1 describes how entities interact with the SID API at a high level. Below is a description of the actual API the entity uses. The API is broken up into **Internal Methods** that the entity implements internally in order to prep the data, and **External Methods** that the entity exposes to collaborators.

Internal Methods

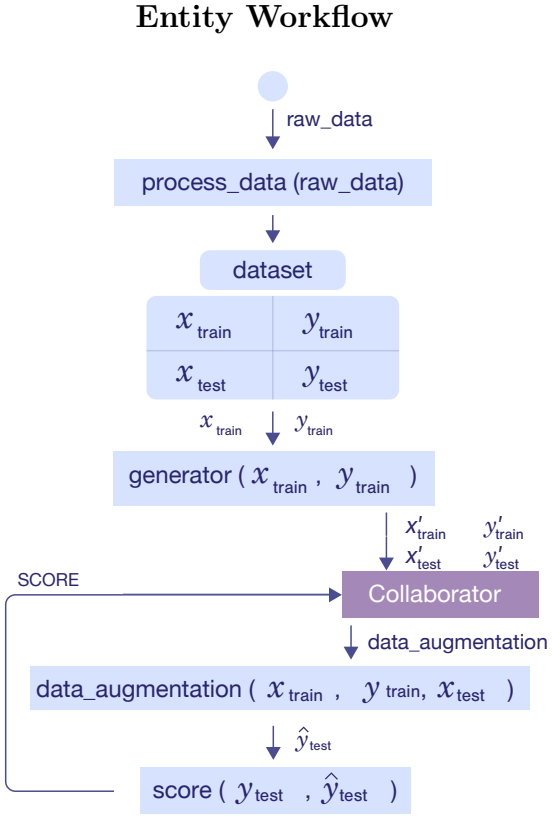


Figure 6-1: We assume entities have domain expertise on their dataset and can clean the data into a tabular format that maximizes the inference value of the data. The entity then fits a generator to their training data and allows collaborators to sample synthetic data, denoted as x' y'. The collaborator will send the entity a `Collaborator` class with a `data_augmentation` function which the entity can then run in a sandbox environment (for example a docker container) to get test predictions $\hat{y}_{\text{test}}$, score these predictions, and update the collaborator with the score. All the blue rectangles mark methods that are executed on the entity's machines, purple denotes code sourced from a collaborator.

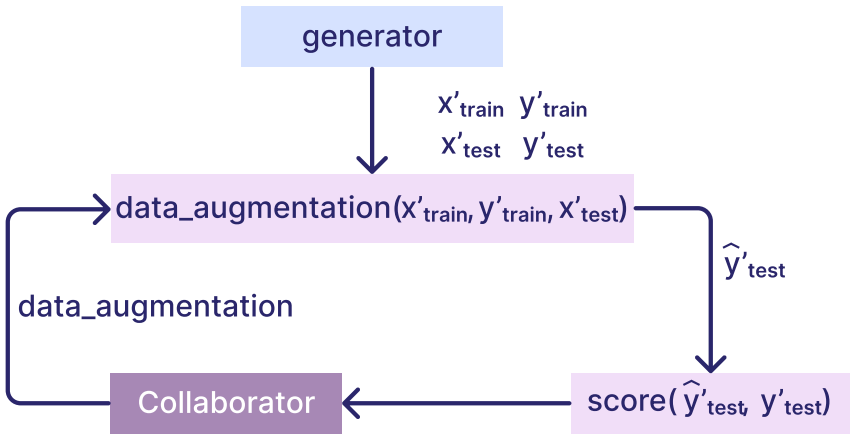


Figure 6-2: Collaborators receive synthetic data from the generator and develop a data_augmentation method that generates predictions for $\hat{y}'_{\text{test}}$, the test split of the synthetic data. The collaborators can (on their local machine) iterate and improve their `data_augmentation` method locally using this score, and, when ready, validate on the real data by uploading their `data_augmentation` method to the entity to get feedback in the form of a score on real data. All the blue rectangles mark methods that are executed on the entity's machines, light purple denotes any methods run on a collaborator's machine, and the dark purple block represents the collaborator.

1. `preprocesser(data)`

   `data`: raw unaggregated data

   This preprocesses data into a tabular form and returns a tabular dataset.

2. `fit_generator(data, multivariate_transform, CopulaModel)`

   `data`: Processed data.

   With categorical data and continuous data labeled, fits a generative model to the data. In the SID library we assume the generator a Copula model; thus `Multivariate-Transforms` (defined in section 6.4.3) are used to convert the marginal distribution of each column to the uniform distribution, before a Copula is fit to the CDF transformed data. This generator can then be used to sample synthetic data.

   `multivariate_transform`: Is a transform used to convert data to have uniform marginals.

   `CopulaModel`: Copula generator model to fit to the data.

External Methods

1. `score(`y_{test}`,` $\hat{y}_{\text{test}}$`)`:

   Returns ML score where y_{test} is real test data and $\hat{y}_{\text{test}}$ is the predictions.

2. `generator()`

   Returns D'_{train} and D'_{test}

3. `score_collaborator(data_augmentation)`.

   Runs collaborator's `data_augmentation` method on the real data to get predictions $\hat{y}_{test}$.

   Then returns the ML score of these predictions:

   `return score(`y_{test}`,` $\hat{y}_{\text{test}}$`)`.

6.3.2 Collaborator API

The Collaborator API simply provides a single function:

1. `data_augmentation`($x'_{\texttt{train}}$, $y'_{\texttt{train}}$, $x'_{\texttt{test}}$): outputs $\hat{y}'_{\text{test}}$, the predictions resulting from the collaborators' Data Augmentation method.

6.4 Synthetic Data Generation in SID

This section describes the SID transforms[1] for transforming a general dataset (with continuous, discrete, mixed continuous and discrete, or datetime column datatypes) to a continuous multivariate uniform distribution. Modeling tabular data using Copulas requires a two-stage approach. First, each column of data must be converted to the uniform distribution (we cover this transform in this section); then, we fit a Copula to the uniform marginal data (as covered in section 4). We experiment with five ways of converting tabular data to uniform distribution which we call `Multivariate-Transforms`. These transforms allow us to observe how different univariate modeling strategies will affect Copula based data augmentation. We hypothesize that generating data which more closely follows the column distribution of the original data will yield better ML performance than data with worse univariate modeling strategies.

In the SID library, we provide `Univariate-Transforms` for applying invertible transforms to individual columns of data to and from the uniform distribution and we provide `Multivariate-Transforms` for transforming entire datasets to have uniform marginals. In the following subsections, `Univariate Continuous Transforms` and `Univariate Non-Continuous Transforms` are both introduced, and then `Multivariate-Transforms` (which combine the Univariate transforms) are introduced.

Note that all columns are assumed to be either Categorical or Continuous for our modeling. (We convert DateTime columns to Unix Time Stamps, an integer seconds count, making them continuous.) We aim to model several datatypes in

[1]Find the SID source code here: https://github.com/DAI-Lab/sid

order to model data well with Copulas. We list these datatypes below (including the challenging datatypes discussed in section 2.3):

1. `Categorical data`

2. `Categorical NAN`
   Missing Categorical values.

3. `Continuous data`
   For continuous data, we can fit a `Univariate Continuous Transformer`.

4. `Continuous NAN values`
   Missing Continuous values.

5. `Mixed Continuous and Discrete data`
   Continuous data can have point masses at specific values. It is common in insurance datasets for loss to include a large number of zero-loss clients and then a smoother continuous distribution of clients with loss greater than zero.

6. `Low Cardinality Continuous`
   Sometimes a column with numerical data will have a very small number of unique values - for example, a binary column containing only zeros and ones.

Our methods focus on ensuring that data is properly transformed for these different datatypes. The current most popular Open-Source Library that uses a version of invertible `Multivariate-Transforms`, SDV [41], does not properly address all of these challenges, and we will compare the approach used in the SDV library with custom methods we develop in the following sections. We begin by defining the custom methods we develop from the univariate level (transforms that operate on a single column) to the dataset level (transforms that transform the entire dataset).

6.4.1 Univariate Transforms for Continuous Data

In this section, univariate Continuous Transforms are introduced to model the CDF and Inverse CDF of continuous datatypes.

Transform Name	Summary
`Spline`	Fit a PCHIP spline[21] to the empirical CDF and inverse CDF of the data to approximate it's CDF and inverse CDF.
`Beta`	Fit a Beta distribution to the data and we use the CDF and inverse CDF as transforms.
`BMM`	Fit a Beta Mixture Model to the data and we use it's CDF and inverse CDF as transforms.

Table 6.1: Summary of SID `Univariate Continuous Transforms`.

The SID library uses several `Univariate Continuous Transforms` to transform continuous data to and from the uniform distribution. We summarize these in table 6.1 and describe them in more detail below:

Spline A PCHIP spline [21] (a monotonically increasing spline) is fitted to the empirical CDF of the data before a second PCHIP spline is fit to approximate the inverse CDF. For the CDF spline, we extrapolate by returning the boundary value as the output should remain between 0 and 1. For the Inverse CDF, also known as the Percentage Point Function(PPF), we do not extrapolate as the inputs are in $[0, 1]$.

Beta Data is linearly scaled to a $[0, 1]$ interval, and a Beta distribution is used to fit to the data.

Beta Mixture Model Data is linearly scaled to a $[0, 1]$ interval, and Beta Mixture Model (BMM) is fit to the univariate data. The CDF of this Beta Mixture model is used to perform CDF transforms on this continuous data. Getting the inverse CDF for the BMM is not trivial because there is no closed form solution for it. Thus we approximate the inverse CDF with a PCHIP spline.

6.4.2 Univariate Transforms for Non-Continuous Data

Now we define the `Univariate Non-Continuous Transforms`, transforms that will allow us to transform non-continuous tabular datatypes like categorical data and mixed continuous and discrete data to and from the unit uniform distribution. We

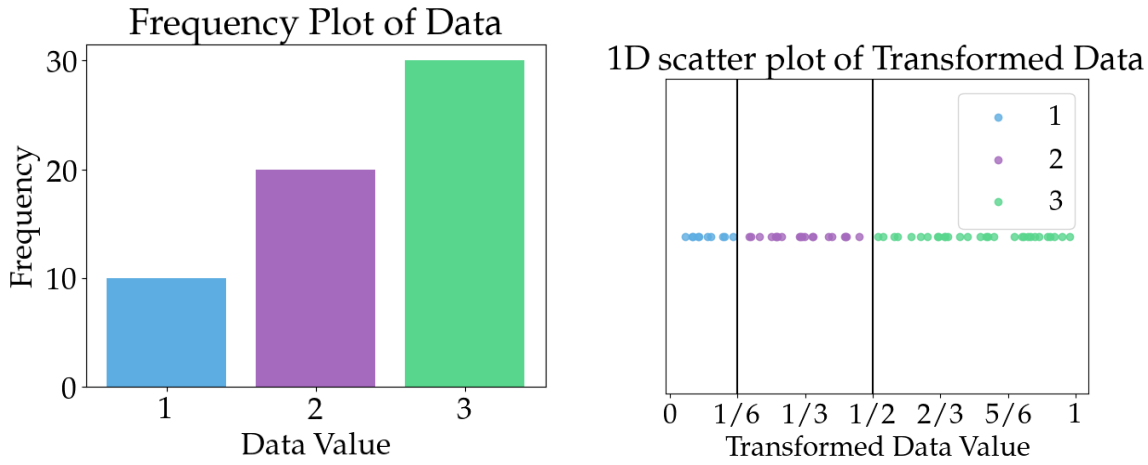
Transform Name	Summary
`Noisy label encoder`	Transforms categorical data to a uniform distribution.
`Low Cardinality Numerical`	Transforms continuous data with a low cardinality to a uniform distribution.
`Target Ordered Categorical`	Transforms categorical data to a uniform distribution.
`Target Ordered Mixed`	Transforms mixed continuous and discrete data to a uniform distribution.
`Ordinal Mixed`	Is a bijection for transforming mixed continuous and discrete data to a uniform distribution.

Table 6.2: Summary of SID `Univariate Non-Continuous Transforms`.

want to transform data to the uniform distribution so we can fit our Copula generators to it. We experiment with different approaches for this task in this section. We provide summaries of the approaches in table 6.2 and give a detailed description below:

Noisy Label Encoder Applies a label encoder to categorical data and then adds a small amount of uniform noise to make the data follow a continuous distribution and avoid point masses. Since the amount of noise added is low, this transformation is easily reversible by rounding values and reversing the label encoding. Then, a `Univariate Continuous Transform` from section 6.4.1 is fitted on the noisy label encoded data and used to convert to and from the uniform distribution. This method is used in the sdv library [41]. NaN Values are treated as an additional category. A drawback of this method is that fitting a continuous distribution to the Noisy label encoded data often leads to incorrect boundaries between each label being learned. Furthermore, in the context of data augmentation, there is no particular order used in the SDV [41] implementation to order the label encoded categorical values. This can make it harder to model column dependencies, especially with a Gaussian Copula which models a single covariance value between columns.

Low Cardinality Numerical Transformer Used for continuous column with n data points with low cardinality, m such that $m << n$. This column only has



(a) Frequency plot of original data values.

(b) 1D Scatterplot of Transformed Data Values.

Figure 6-3: In this figure, we have a dataset of low cardinality numerical data. The three values in our dataset are 1, 2, and 3 and they have a frequency of 10, 20, and 30 respectively. In figure 6-3a we provide a frequency plot for the data. The `Low Cardinality Numerical Transformer` transforms the data to subintervals of the unit uniform distribution, and we plot the output in figure 6-3b. We maintain the ordering of the data as we see all the transformed values for the data obey that $1 < 2 < 3$. Additionally, we observe that 1 has empirical probability $p_1 = \frac{1}{6}$ and all the 1 values are scattered uniformly over $[0, \frac{1}{6}]$. Similarly, the transformed data for 2 and 3 are scattered uniformly on a $p_2 = \frac{1}{3}$ and $p_3 = \frac{1}{2}$ sized sub-interval of $[0, 1]$. None of the intervals on which 1, 2, and 3 transformed data lie intersect, which ensures we can reverse the transform by sampling a point uniformly on $[0, 1]$ and mapping it to 1, 2, or 3 depending on which value's subinterval the point lies in.

point masses as $m << n$ and thus we model it as an ordinal categorical variable. Assume data takes values $v_1, \ldots, v_m$. Assume that p_i is the empirical probability of value v_i. We create a bijective mapping to the uniform distribution that maintains the ordinality of the data by mapping each v_i to the uniform distribution on the interval $[\sum_{j=1}^{i-1} p_i, \sum_{j=1}^{i} p_i]$. For example, v_1 would be mapped to $[0, p_1]$, and v_2 would be mapped to $[p_1, p_1 + p_2]$. NaN Values are imputed with the mean before the transform. We provide a visual depiction of this mapping from values to subintervals of the unit uniform distribution in figure 6-3. This approach of converting categorical variables to subintervals of the uniform distribution is used in several other methods in this section.

Target Ordered Categorical Transformer Used for categorical data. The `Noisy Label Encoder` for modeling categorical data gives a random ordering to categories. This is not optimal when using Gaussian Copula and GMCMs which model column dependencies using covariance. Instead it would make more sense to order categories by the expected value of the target given the category. This will allow the Gaussian Copula to learn a more meaningful covariance between the categories and the target than a random ordering. We describe a procedure for this task below. Assume the column has m categories $c_1, \ldots, c_m$. We impose ordinality on these categories by indexing them such that $i \leq j \leftrightarrow \mathbb{E}[\text{target}|c_i] \leq \mathbb{E}[\text{target}|c_j]$. Assume that p_i is the empirical probability of c_i. We create a bijective mapping (using essentially the same procedure as for `Low Cardinality Numerical Transformer` and depicted in figure 6-3) to the uniform distribution that maintains the ordinality of the data by mapping each c_i to the uniform distribution on the interval $[\sum_{j=1}^{i-1} p_i, \sum_{j=1}^{i} p_i]$. Notice that we have imposed ordinality on the categories. In comparison to the Noisy Label Encoder method, we remove the need for fitting continuous distribution as this is both inefficient and error prone because the CDF transform will not be uniform if the `Univariate Continuous Transform` does not fit the data well. NaN Values are treated as an additional category. The imposition of ordinality

should improve modeling using a Gaussian Copula as the covariance between the target and column will be more accurate, and for a GMCM, categories with similar conditional expectations for the target can be lumped into a single mixture as they are spatial neighbors.

Target Ordered Mixed Transformer This transform is meant for mixed continuous and discrete data. Assume the data has continuous values $c_1, \ldots, c_k$ and discrete values $d_1, \ldots, d_m$. We add a categorical column with categories $c, d_1, \ldots, d_m, NaN$ indicating whether the univariate data value is continuous, NaN, or corresponding to a discrete value. This categorical column will be transformed to the uniform distribution using the `Target Ordered Categorical Transformer`. Then, for the original continuous column, we either impute or remove NaNs when fitting the continuous distribution to it as follows:

Impute and Noisy Label Encode NaN values are imputed with the mean (plus a small epsilon if the mean is an observed value in the data). All the discrete values and the imputed NaN values are point masses, so we add a small amount of uniform noise, just as we do with the `Noisy Label Encoder`, in order to make sure there are no point masses and the data follows a smooth continuous distribution.

Remove Remove NaNs and discrete values and then fit the `Univariate Continuous Transform` to the remaining continuous data. We then sample from the uniform distribution when we transform NaNs or discrete values to the uniform distribution.

We fit a `Univariate Continuous Transform` to the continuous data using either the `Impute and Noisy Label Encode` or `Remove` approach above, and use the CDF to transform it to the uniform distribution. We can reverse this transformation by using the inverse CDF to transform back to original continuous distribution, and then using the categorical column to keep their value (if the categorical value is c) or map to NaN or the respective discrete value (if the category is NaN or d_i for some i).

Ordinal Mixed Transformer For this method, we transform mixed continuous and discrete data to the uniform distribution while maintaining ordinality of the data. Note that NaN values are imputed with the mean (plus a small epsilon if the mean is an observed value in the data) and are treated as discrete values. At a high level, we are mapping the the continuous and discrete data into subintervals of the uniform distribution such that the order of variables is maintained after the mapping. We provide a simple example of the approach in figure 6-4. Below we give that algorithm for computing the subintervals for the mapping and give a more detailed description of the algorithm.

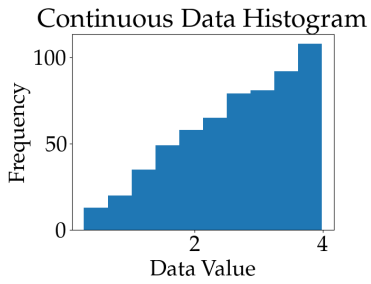
Algorithm 1 Ordinal Mixed Transformer

Input : $c_1, \ldots, c_k$ - the continuous data values
$d_1, \ldots, d_m$ - the discrete values
p_i - the empirical probability of each discrete value
p_c - the empirical probability of a value being continuous

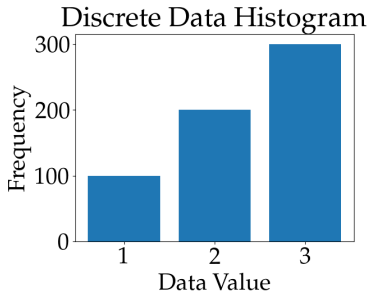
for $i \leftarrow 1$ **to** m **do**
  $l \leftarrow$ the left side of the interval equals the empirical probability a datapoint is less than d_i
  $r \leftarrow l + p_i$, the right side of the interval equals l plus the probability of the discrete value
  $I_i \leftarrow [l, r]$, this is the uniform interval to which the discrete data is transformed. $I_c \leftarrow [0, 1] \ \cup_i I_i$, this is the uniform interval on which the continuous data will be transformed to.
end

Output: $I_1, \ldots, I_m$ - Intervals discrete values are mapped to.
I_c - Interval the continuous data is mapped to.

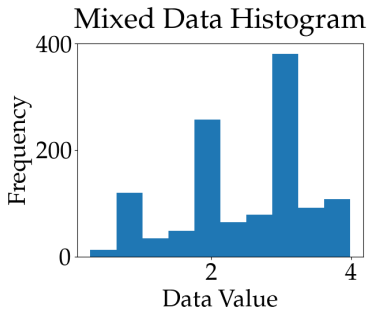
$c_1, \ldots, c_k$ are the continuous values, $d_1, \ldots, d_m$ are the sorted discrete values, so $i < j \leftrightarrow d_i < d_j$. We define p_i as the empirical probability of discrete value of d_i and p_c as the empirical probability of a value being continuous. Then we fit the `Univariate Continuous Transform` to the continuous data and perform a CDF transform so $u_1, \ldots, u_k$ are the CDF transformed continuous values. We partition the unit uniform marginal we are transforming the data to into intervals. Each discrete value d_i is mapped to uniform interval on I_i, and the continuous data is mapped to be uniformly distributed on I_c as described in algorithm 1.



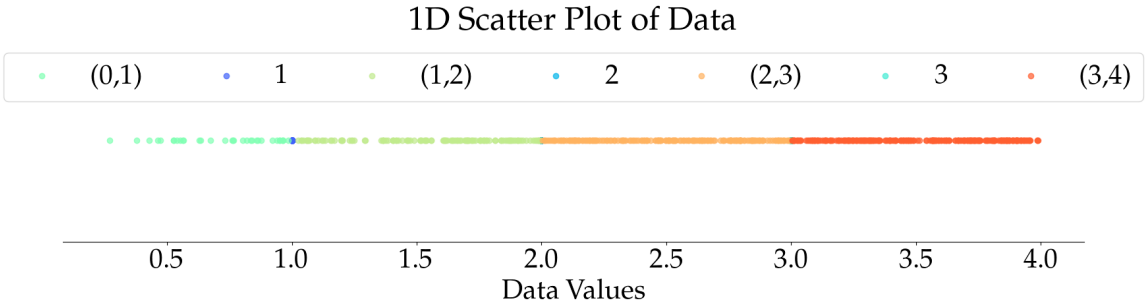
(a) Histogram of 600 Continuous Data samples from a beta distribution.



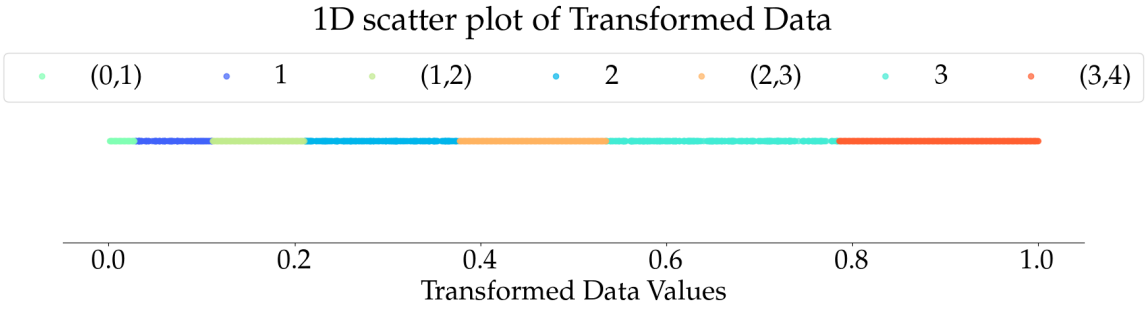
(b) Frequency plot of 600 samples from a discrete distribution. Same as 6-3a except with 600 instead of 60 total samples.



(c) We plot the mixed data distribution of combining the data in figure 6-4a and figure 6-4b.



(d) We plot a 1D Scatterplot of the data, coloring each point depending on which discrete value it is equal to, 1, 2, or 3, or which interval it lies in, $(0, 1)$, $(1, 2)$, $(2, 3)$, or $(3, 4)$.



(e) 1D Scatterplot of Transformed Data Values, with same coloring as in figure 6-4d.

Figure 6-4: In this figure, we give an example of a mixed discrete dataset in figure 6-4c. We show the 1D scatter plot of the Data, coloring points by where they fall in relation to the discrete points in figure 6-4d. Finally, we transform the data using the `Ordinal Mixed Transformer`, which first takes all continuous data in $(0, 1)$ and maps it to the lowest interval in the unit uniform distribution. Then it maps the discrete data equal to 1 to the next lowest, and so on until we get to the data in $(3, 4)$ which is mapped to the highest interval. The width of the intervals that each color maps to is $\frac{\text{the number of points for that color}}{\text{the total number of points}}$, the empirical probability of the data in the interval.

Transform Name	Modeled Univariates			
	Cont	Cat	Low	Mixed
`SDV Default (SDV)`	✓	✓	✗	✗
`SDV Categorical (SDV_CA)`	✓	✓	✓	✗
`Ordinal Mixed Multivariate`	✓	✓	✓	✓
`Target Ordered Mixed Multivariate`	✓	✓	✓	✓

Table 6.3: Summary of SID `Multivariate-Transforms`. These transforms take a full dataset and transform it to have uniform marginal distributions. The `Multivariate-Transforms` vary in which Univariate distributions the are designed to model. The Cont, Cat, and Mixed columns in the table refer to whether the multivariate transform can model univariate marginals for data that is continuous, categorical, or mixed continuous and discrete respectively. The Low column in the table refers to whether the multivariate transform can model univariate marginals for low cardinality numeric univariate data.

For the transformation from the original mixed continous discrete data to the unit uniform distribution, it is straightforward that for each d_i, we simply sample a uniform value from the interval I_i. For continuous values, we CDF transform from $c_i \to u_i$, and multiply by p_c (the length of I_c). Then we add the length of each I_i where $d_i < c_i$ to ensure the data point is in I_c and that the data is uniformly distributed on $[0, 1]$.

For the reverse transformation from the unit uniform distribution to the mixed continuous discrete data, we map datapoints in an interval I_i to discrete value d_i (and convert the appropriate d_i to NaN). For a value r in I_c, we subtract $\sum_{i|d_i<r} p_i$. This transforms the continuous variables to a uniform distribution on $[0, p_c]$. We then divide by p_c which transforms the continuous data back to the $[0, 1]$ interval. We can then apply the inverse CDF to transform these continuous values back to the original distribution.

6.4.3 Multivariate-Transforms

In this section we present different invertible transforms for tranforming entire tabular datasets to have uniform marginals. We provide a summary of these transforms in table 6.3. These are implemented in SID and we emperically compare these methods

in later experiments. Note that `SDV Default` serves as a baseline implementation to which we compare our methods. We summarize our transforms in the table 6.3 and give a more detailed description below. For each transform, we provide the Uniform marginal transform, which transforms from the original data distribution to a distribution with all uniform marginals. Additionally we provide the Inverse Transform, which inverts the Uniform marginal transform.

SDV Default (SDV) This is the default transform used by the SDV [41] library, one of the most popular python libraries for fitting Copulas to tabular data.

Uniform marginal transform

1. Add NaN indicator categorical column for all columns with missing values.
2. Apply `Noisy Label Encoder` to all categorical columns to make them continuous.
3. Fit a continuous distribution to each column and apply CDF transform.

Inverse Transform

1. Apply inverse CDF transform to each column.
2. Inverse `Noisy Label Encoder` for all categorical columns.
3. Use NaN indicator column to map indicated rows to NaN values.

SDV Categorical (SDV_CA) This uses the same approach as the SDV library for handling categorical data and NaNs, but uses the `Ordinal Categorical transformer` to model continuous data with low cardinality, and the `Ordinal Mixed Transformer` for the rest of the continuous columns. This change should allow for better handling of mixed continuous and discrete data.

Uniform marginal transform

1. Add NaN indicator categorical column for all columns with missing values.
2. Apply `Noisy Label Encoder` to all categorical columns to make them continuous.

3. Apply `Ordinal Mixed Transformer` transformer to each column and apply CDF transform.

Inverse Transform

1. Apply inverse `Ordinal Mixed Transformer` transform to each column.
2. Inverse `Noisy Label Encoder` for all categorical columns.
3. Use NaN indicator column to map indicated rows to NaN values.

Ordinal Mixed Multivariate (OM) Apply `Target Ordered Categorical Transformer` to categorical data, `Ordinal Categorical Transformer` to continuous data with low cardinality, and `Ordinal Mixed Transformer` to the rest of the continuous columns. To invert, we simply apply the inverse of the univariate transforms to the corresponding columns.

Target Ordered Mixed Multivariate (TOI or TOR) Apply `Target Ordered Categorical Transformer` to categorical data, `Ordinal Categorical Transformer` to continuous data with low cardinality, and `Target Ordered Mixed Transformer` to the rest of the continuous columns. Note that the `Target Ordered Mixed Transformer` can either use the **Impute and Noisy Label Encode** (which we denote as `TOI`) or the **Remove** (which we denote as `TOR`) setting can be is used for handling NaNs and discrete values; we experiment with globally using one setting or the other. To invert, we simply apply the inverse of the univariate transforms to the corresponding columns.

6.5 SID Sampling Methods

We have discussed multiple approaches to converting raw data to data with uniform marginals using our `Multivariate-Transform`. We have also discussed how to fit several Copula models to this data with uniform marginals in section 4. Now we will discuss different sampling methods that we experiment with for sampling synthetic data. The question of what data to sample from the Copula is not trivial; neither

Transform	Summary
`Unconditional Sampling`	Sample approximately half of the real dataset size from the Synthetic Data generator.
`Mixture Sampling`	Same as `Unconditional Sampling` except we sample from a single mixture in GMCM or segment in the Segmented Gaussian Copula model.
`Quantile Sampling`	Same as unconditional sampling except we use rejection sampling to filter for samples that are in a specific quantile for a column.
`Unconditional Tuned Sampling`	Is the same as `Unconditional Sampling` except we tune the amount of Synthetic Data sampled.

Table 6.4: Summary of SID `Multivariate-Transforms`.

is the question of how much data to sample for the purpose of Data Augmentation (recall these are research questions 3 and 4 from section 1). Here we introduce and describe four sampling methods in the SID library to help assess different tabular Data Augmentation schemes. Note that n denotes the number of training data samples.

Unconditional Sampling A fixed amount of Synthetic Data is sampled from the data generator model. A generator that creates synthetic data that follows the distribution of the original data may help an ML model better learn the distribution of the data, and could potentially improve model learning. We simply sample $n/2$ Synthetic Data samples for our experiments.

Mixture Sampling For generators with a notion of mixtures (such as the segmented Copula generator or the Gaussian mixture Copula model from section 4), we can sample a fixed amount of data from each mixture. The mixtures that provide a higher ML performance on a validation set, after augmenting with Synthetic Data sampled from the mixture, are then used to augment the model. If the mixture model learns the distribution of a portion of the data that the baseline ML model normally struggles with, sampling from this mixture could help the model perform better on this portion. We simply sample $n/2$ Synthetic Data samples for our experiments.

Quantile Sampling For each of the m columns in the dataset, we split the data into

r quantiles (except for categorical data which we just iterate over all categorical values). For each column-quantile pair, we sample synthetic data that has satisfied the condition that all data from this column is within the quantile by using rejection sampling. This is essentially a brute force method to search over the space of the Synthetic Data in an attempt to find pockets that we can use to bias the ML model to perform better. We simply sample $n/2$ Synthetic Data samples for our experiments.

Unconditional Tuned Sampling The SID library also alows for tuning the amount of Synthetic Data using a tree-structured parzen estimator(TPE) [5]. We assume the optimal number of samples is between 0 and n. In our implementation, we estimate the performance of a specific number of Synthetic Data samples, t, by performing 10-fold cross validation and sampling a fresh batch of t Synthetic Data samples to augment with for each fold. We average the ML performance for each fold to get the score for t, and use TPE to find the t value that optimizes ML performance. For our experiments we sample the data unconditionally, but you could perform this in conjunction with Mixture or Quantile sampling.

6.5.1 `SID`: Entity Presets

In the SID library we provide preset entities for all datasets in section 2. These preset entities are meant to serve as examples to help other researchers interested in applying these data augmentation experiments to their data, and also serve as an example that entities can use for considering how to pre-process and prepare their data and APIs to engage in collaborations involving restricted access to data while still allowing experimentation with Data Augmentation workflows.

6.6 `SID`: Collaborator Presets

Collaborators can sample Synthetic Data from the entity to visualize their data, and then propose their own Data Augmentation or baseline ML model approaches to evaluate on the entity's private data. In the SID library, we implement three

Collaborator	Copula Generator
`Gaussian Copula Collaborator`	Gaussian Copula.
`Segmented Gaussian Copula Collaborator`	Segmented Gaussian Copula.
`Gaussian Mixture Copula Model Collaborator`	Gaussian Mixture Copula Model.

Table 6.5: Summary of SID `Collaborator` classes. Each `Collaborator` class performs Tabular Data Augmentation by generating Synthetic Data from a specific Copula Generator. Collaborators use `Multivariate-Transform` to transform the data to have uniform marginals before fitting the Copula Generator. All of the previously defined `Multivariate-Transform` are supported for each `Collaborator` class.

collaborator presets to test the SID API, while also providing useful approaches for experimenting with Data Augmentation on tabular data. We provide a summary of the collaborator presets in table 6.5. Below is a description of the implemented collaborator presets implemented in the SID library:

Gaussian Copula Collaborator This collaborator fits a Gaussian Copula to the data with a predefined `Multivariate-Transform`. It allows support for all sampling methods from section 6.5 except `Mixture Sampling` as the Gaussian Copula has no notion of mixtures.

Segmented Gaussian Copula Collaborator This collaborator fits a segmented Gaussian Copula to the data with a predefined `Multivariate-Transform`. It allows support for all sampling methods from section 6.5.

Gaussian Mixture Copula Model Collaborator This collaborator fits a Gaussian mixture Copula model to the data with a predefined `Multivariate-Transform`. It allows support for all sampling methods from section 6.5, and allows the use of fitting the Copula using the AD-GMCM method or the PEM-GMCM methods discussed in section 4.

Chapter 7

Tabular Data Augmentation Experiments

In Section 2, we introduced several challenges pertaining to training Synthetic Data generator models using real-world insurance data, as well as using Synthetic Data to improve ML models. We designed experiments to verify whether our proposed methods can help improve the final performance of ML models, in comparison with a baseline where Synthetic Data is not introduced at all. We describe our experiments in this section.

7.1 General Experiment Setup

In this section we cover our baseline and the setup for all of our future experimentation.

For our experiments, we only use the `Databricks`, `Emicen Auto, Emicen Rail, and Fraud 2` datasets we describe in section 2. Most of the other datasets had extremely long runtimes (around 2 days on a system with with 64 cores and a 3060ti GPU) for training the Gaussian Mixture Copula Synthetic Data model, making it intractable to get mean and average performances over 10 end-to-end runs with different initializations (as we do for all of our experiments). Additionally, for some datasets, we were able to get perfect prediction metrics (1 Gini score or 1 AUC for

example). We removed those datasets in order to benchmark on more challenging ones.

For our baseline, we train an XGboost model on only training data. Note that we add no Synthetic Data. See section 3.2 for details and baseline results. Each experiment shares the same settings, as follows.

1. Run the experiment on each of the following datasets: `Databricks`, `Emicen Auto, Emicen Rail, and Fraud 2`, described in section 2.

2. Use a Copula Synthetic Data generator with the `Spline Univariate Continuous Transform` as the default.

3. Augment training data with n' samples of Synthetic Data sampled (using `Unconditional Sampling` by default) from the generator. n' = half the size of the training dataset by default.

4. For evaluating all our experiments on each dataset, D, we split D into $D_{\text{train}} \cup D_{\text{test}}$, keeping this split constant. We additionally hold out 10% of the data in D_{train} as our validation set.

5. We use the same ML model and preprocessing pipeline described in section 3.2 for all experiments.

6. We collect and compare the Gini score as our regression performance metric for continuous loss prediction and AUC for binary loss and fraud prediction.

7. For each experiment, we fit the generator model once. Using a sampling method (`Unconditional Sampling` is used by default), we sample 10 size n' batches of Synthetic Data. For each batch, we augment our training data with the batch, train a model on the augmented dataset, and record our test set ML score.

8. We define the best model as the one that achieves the highest ML performance consistently on all datasets. If there is no consistent winner, we select one and provide our reasoning.

9. We define the ML score improvement as the test set ML score (when using Data Augmentation) minus the baseline test set ML score.

7.2 Experiment Definitions

7.2.1 Univariate Modeling Experiment

We perform two Univariate Modeling Experiments to understand the impact of univariate modeling on tabular Data Augmentation.

1. We compare the Data Augmentation performance over all the `Multivariate-Transforms`. This will show us whether specific modeling of low cardinality, categorical, and mixed continuous and discrete univariate data is beneficial. We use the best-performing `multivariate-transform` for all of the other experiments.

2. We compare the Data Augmentation performance over all the `Univariate Continuous Transforms`. This will reveal the importance of accurate modeling of the continuous data (and its CDF).

7.2.2 Copula-Dependency Modeling Experiment

We experiment with different Copula models, defined in section 4.1, to better understand how important column dependency modeling is to tabular Data Augmentation. Specifically, we run the Segmented Gaussian Copula, the PEM-GMCM, and the AD-GMCMs. We use the best Copula model from this experiment for the rest of the experiments. For both GMCM models, we use `MI-GMM` initialization; for the PEM-GMCM model we use only 3 mixtures, as the PEM-GMCM model R program language implementation [6] struggles with singularities and produces errors on many datasets when using higher numbers of mixtures. Because our AD-GMCM implementation is robust to singularities, for AD-GMCM we vary the number of mixtures using 3 and 20 mixtures to compare how this number affects ML performance.

7.2.3 Sampling Method Experiment

Our goal is to understand whether sampling from certain mixtures or quantiles of the data is beneficial. We experiment with the `Mixture Sampling` and `Quantile Sampling` methods defined in section 6.5 on the 3 and 20 mixture AD-GMCM. We compare the best-performing mixture and quantile between the 3 and 20 mixture AD-GMCM to see whether performance scales with the number of mixtures (indicating that more complex column dependency modeling improves tabular Data Augmentation performance). Additionally, we wish to see if these sampling methods are able to find better Synthetic Data for Data Augmentation than `Unconditional Sampling` can.

In a real ML workflow, we would decide on which mixture or quantile to sample from based on the validation set performance. However, this assumes that the test set performance is correlated with the validation set, which may not be the case. We examine the practicality of mixture and quantile sampling by assessing whether the cross-validation ML score correlates with the test set ML score. Specifically, we record the cross-validation ML scores for the 20 AD-GMCM, and see whether they correlate with the test set ML scores.

7.2.4 Amount of Synthetic Data Experiment

To better understand how much Synthetic Data should be used, we take the best Copula model from experiment 7.2.2 and optimize the amount of SD sampled by using the `Unconditional Tuned Sampling` method.

Chapter 8

Tabular Data Augmentation Results And Analysis

In this section, we analyze the results of our experiments as detailed in section 7. Full experimental results are recorded in appendix C.

8.1 Univariate Modeling Results

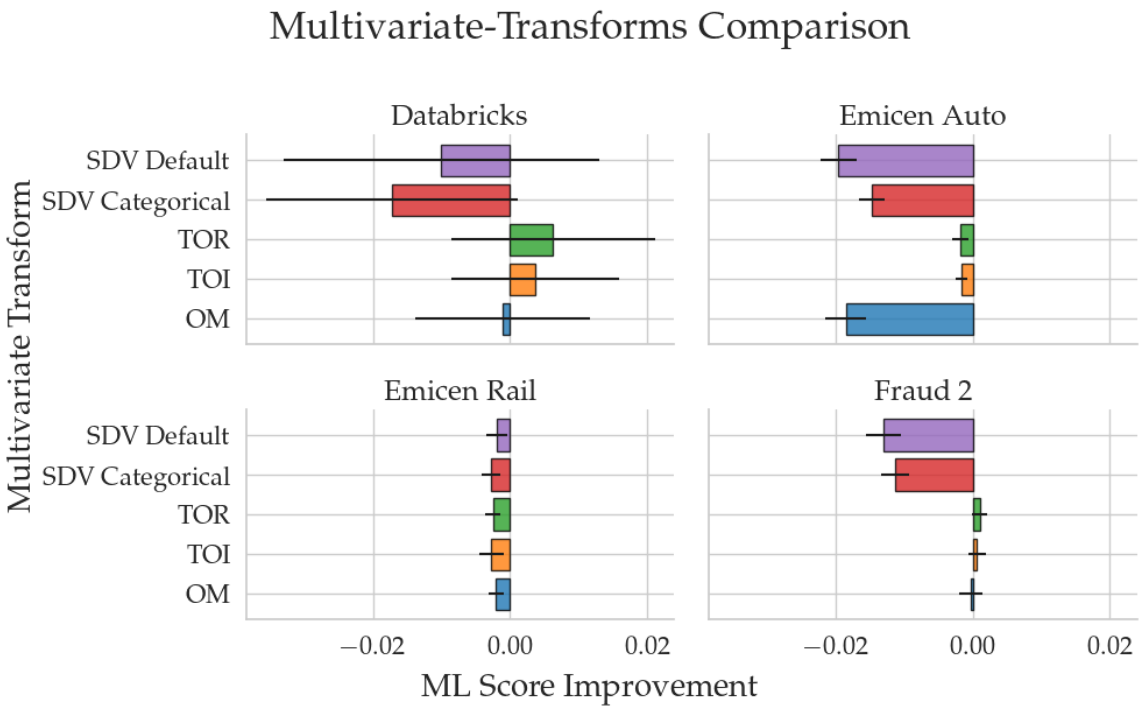


Figure 8-1: Bar plot of the Gaussian Copula data augmentation results. We use the `Spline` continuous transform, and plot the ML score improvements. The improvement is averaged over 10 runs, and the black lines denote a standard deviation for the sample mean.

In this section, we are trying to answer **RQ1**: is accurate univariate column modeling important for tabular Data Augmentation? In figure 8-1, we present the ML score improvements as we vary the `Multivariate-Transforms`. Note we are using a Gaussian Copula Synthetic Data generator with a `Spline Univariate Continuous Transform`. We see that the `TOR` and `TOI Multivariate-Transforms` perform significantly better than `SDV_CA` and `SDV` on 3 out of 4 of the datasets, performing similarly to them on the remaining one. `OM` outperforms `SDV_CA` and `SDV` on half the datasets, and performs similarly on the remaining two. This provides proof that our `Target Ordered Categorical Transformer` transform (used by `TOR, TOI, and OM`) better models categorical data compared to the noisy label encoding (used by `SDV and SDV_CAT`) resulting in the improved Data Augmentation.

We expected to see an improvement when using `SDV_CA` instead of `SDV` as we

introduce modeling of mixed continuous and discrete data, with the `Ordinal Mixed Transformer`, and modeling of low-cardinality continuous data, with the `Ordinal Categorical Transformer`. However, there isn't a significant difference between Data Augmentation performance for these two `Multivariate-Transforms`. It is most probable that these datasets simply don't contain significant target column correlations that require mixed continuous and discrete column modeling. For the rest of our experiments, we use only `TOR`, as it performed best on all datasets.

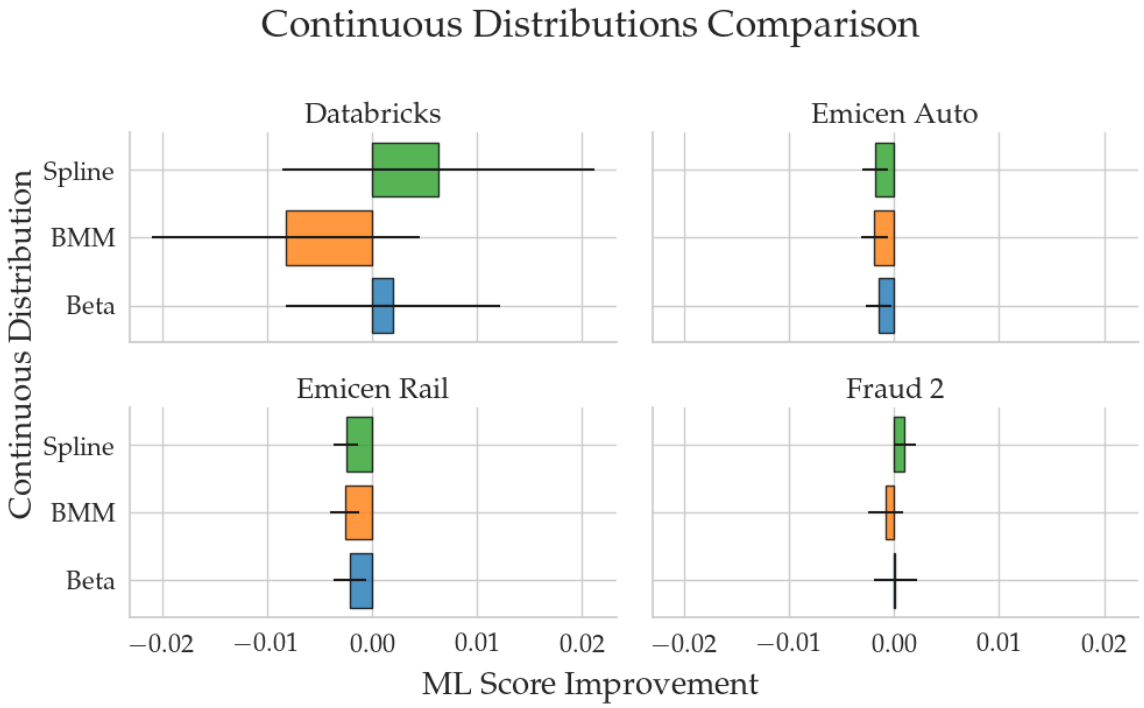


Figure 8-2: Bar plot of the Gaussian Copula data augmentation results. We use the `TOR Multivariate-Transform`, and plot the ML Score improvements over all `Univariate Continuous Transforms`. The improvement is averaged over 10 runs, and the black lines denote one standard deviation.

In figure 8-2, we plot the ML score improvements when varying the `Univariate Continuous Transforms`. We see that they all perform relatively similarly on all datasets. We expected to see an improvement as we moved from fitting with the Beta distribution to fitting with `BMM`; however, the `BMM` parameter estimation suffered from instabilities, making it have a bad fit for the `Databricks` dataset. We also expected the `Spline` to have the best results, and to more accurately model the CDF

transforms for continuous variables, but there isn't a significant difference on any dataset. For the rest of the experiments we use the `Spline`, because it behaves stably (compared to the BMM), and its non-parametric form makes it more flexible, giving it a more uniform CDF transform for data (compared to the Beta). This is desirable for fitting GMCMs, which can have stability issues. See the full Univariate Modeling Results in the appendix section C.1

8.2 Copula-Dependency Modeling Results

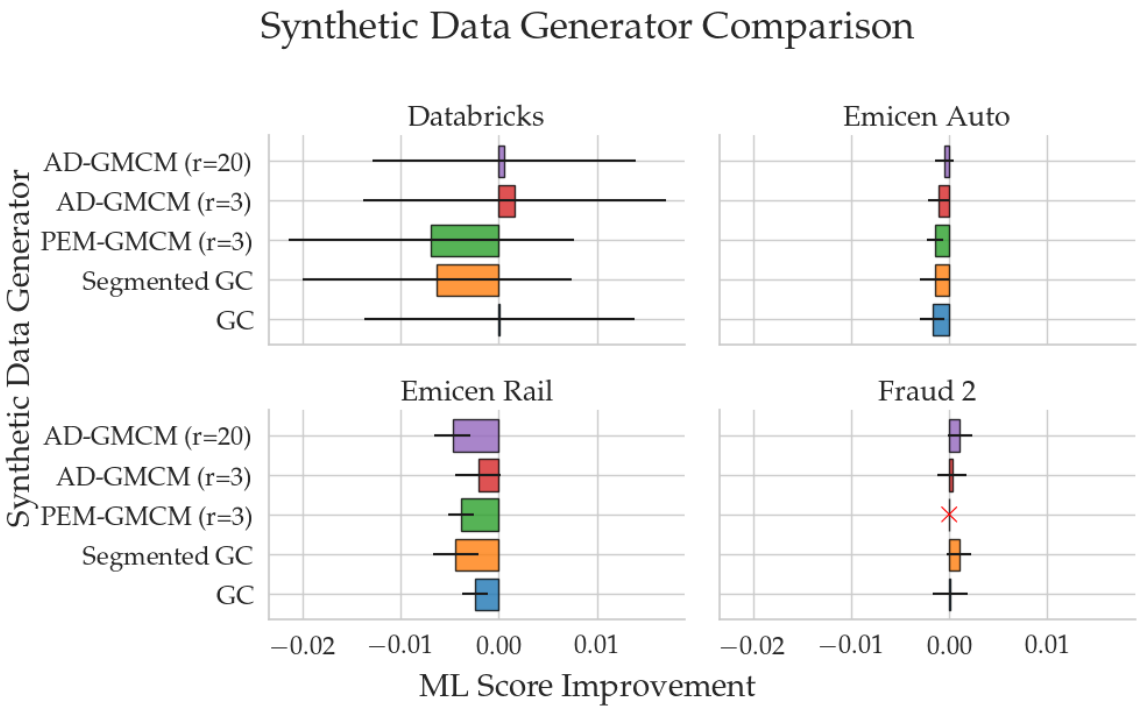


Figure 8-3: We report the ML score improvement for each Copula model and dataset combination, using the TOR `Multi-Transform`. r denotes the number of mixtures for the GMCM models. Note that the PEM-GMCM(r=3) errors when it assigns zero probability to a mixture, making it fail for the `Fraud 2` dataset.

In this section, we are trying to answer **RQ2**: Is accurate column dependency modeling important for tabular Data Augmentation? In figure 8-3, we compare the performance of each Copula model: Gaussian Copula, Segmented Gaussian Copula (with two segments), 3 mixture PEM-GMCM, 3 mixture AD-GMCM, and 20 mixture AD-

GMCM.

We do not see any consistent significant differences in performance when varying the Copula models. We expected that moving from Gaussian Copula to the Segmented Gaussian Copula and then to the GMCMs would incrementally improve performance by allowing more flexible modeling. We also expected the 20 Mixture AD-GMCM to perform significantly better than the 3 Mixture AD-GMCM, but again, we see a lack of improvement on these datasets.

We note that none of the models consistently improves the ML performance over the baseline. We expected adding Synthetic Data from a more flexible Copula model would improve performance on all datasets. However, for all of the datasets, there is no significant improvement. Two possible reasons for this behavior are:

- The amount of Synthetic Data we are sampling may need to be tuned to improve the Data Augmentation performance. We address this in section 8.4.

- Additionally, it may be the case that `Unconditional Sampling` for tabular Data Augmentation is not beneficial, so we try alternative sampling methods, `Mixture Sampling` and `Quantile Sampling`, in the next section.

Note that we provide the full Copula-Dependency modeling results in the appendix section C.2.

8.3 Sampling Method Results

In this section, we are trying to answer **RQ3**: given a generator, how should we sample for data augmentation? In these experiments, we observe that specific mixtures and quantiles improve performance. However, we also find that for these sampling methods, validation performance is not a good proxy for ML performance, thus it is not possible to know apriori whether your Synthetic Data will improve performance on the test set using only validation accuracy.

8.3.1 Mixture Sampling

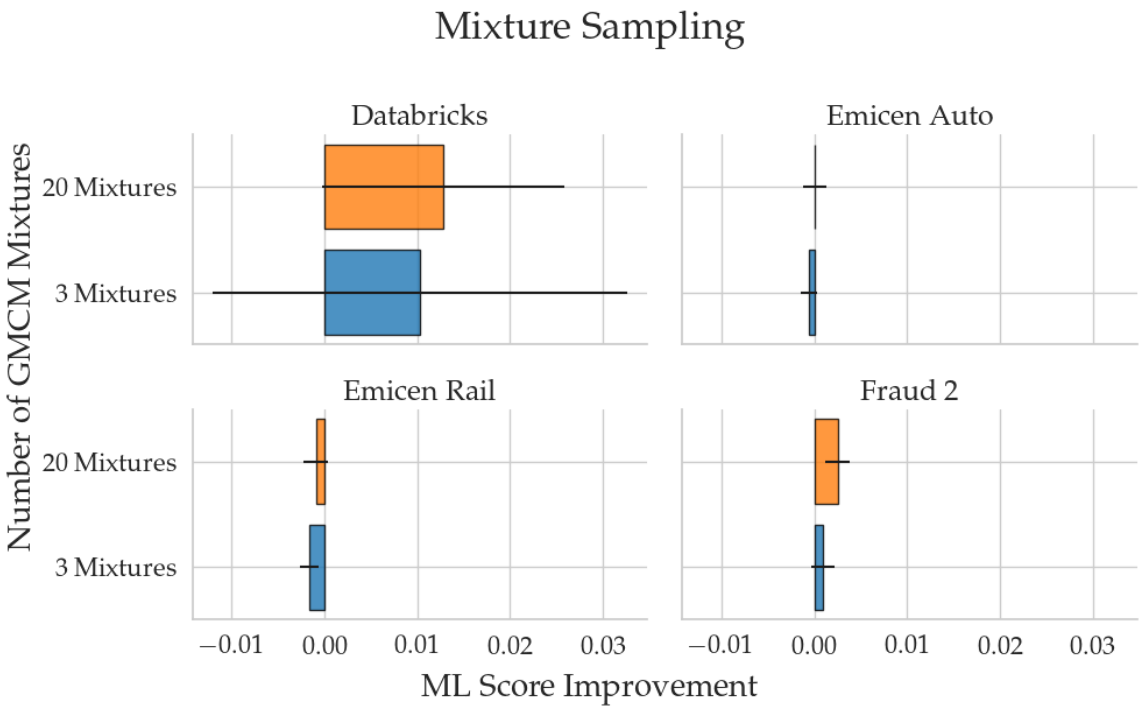


Figure 8-4: 3 and 20 mixture AD-GMCM Mixture Sampling results. We plot the ML Score Improvement for the Mixture with the highest Test Set Score. The TOR `Multivariate-Transform` is used.

See full results for Mixture Sampling in the appendix section C.3.1. We plot the best results obtained using mixture sampling on a 3 and 20 mixture AD-GMCM in figure 8-4.

We observe that mixture sampling sometimes improved performance and sometimes hurt. Now we ask whether it is possible to know a priori, from validation metrics, whether test set performance will be higher with this augmented training data. We plot in figure 8-5 the test set ML improvement vs validation set performance for each data augmentation trial in order to see if there is a correlation. We observe that across the 20 mixtures we sample from, there is not consistent correlation between performance on the validation and test sets when augmenting with Synthetic Data. (Fraud 2 is the only dataset that has a stronger correlation – the rest do not.) If Synthetic Data were helping our model generalize, we would expect to observe a correlation between validation and test set accuracies. The lack of such

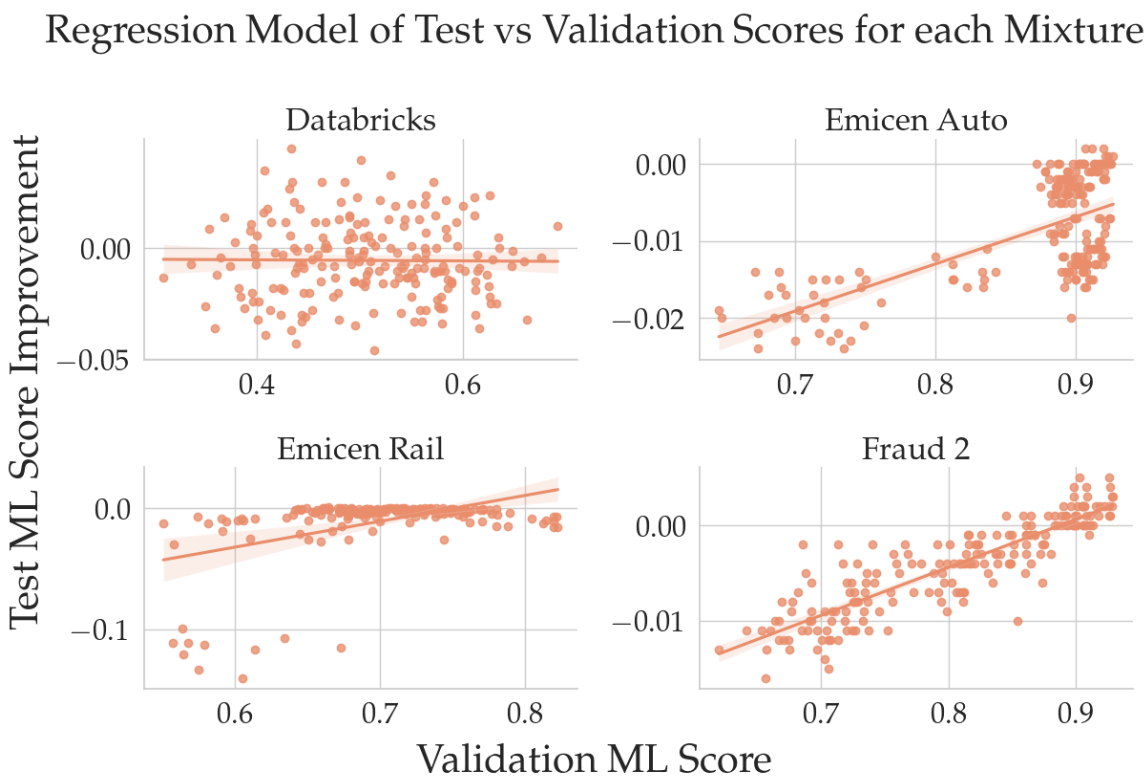


Figure 8-5: This figure presents linear regression plots with 95% confidence intervals of the test set ML score improvement vs validation set ML score for each dataset when sampling using the `Mixture Sampling` method. We are sampling and augmenting with synthetic data sampled from the 20 mixture AD-GMCM. We computed 10 trials for each mixture, for a total of 200 points.

a correlation in several datasets implies that Synthetic Data from a mixture does not improve generalization, but has a random effect on training performance (potentially by overfitting to certain synthetic samples that may or may not be in the test and validation set).

8.3.2 Quantile Sampling

See full results for Quantile Sampling in the appendix section C.3.2. Some Quantiles significantly improve performance as shown in figure 8-6, where we share the best performing quantile result for each dataset when sampling from a GMCM with 3 and 20 clusters. Note that the Fraud 2 dataset was not assessed for this experiment because it has extremely high-cardinality categorical columns which make it intractable

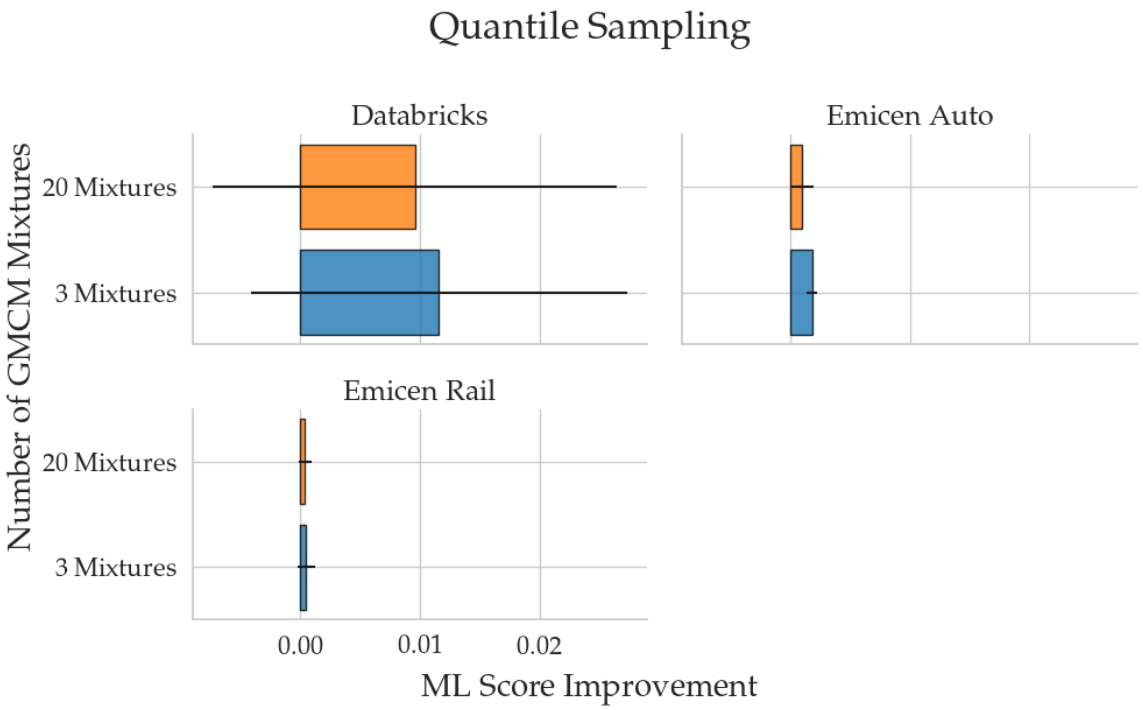


Figure 8-6: Quantile with highest Average Test Set ML Score after Data Augmentation is plotted for 3 and 20 Cluster AD-GMCM. There are no results for the Fraud 2 dataset as it is not tractable to run Quantile sampling on it as it has very high dimensionality categorical columns.

to perform quantile sampling[1]. As we increased the number of clusters from 3 to 20, we again did not see a statistically significant improvement in ML performance.

Quantile sampling cannot be employed practically because, just like with mixture sampling, there is no way to validate whether your Synthetic Data will improve ML performance by using the validation set ML score. In figure 8-7, we observe no clear correlation between the validation set ML Score and the test set ML Score, implying that validation set performance is not a good estimate of test set performance. This could potentially be resolved by using a larger validation set size, as in our experiments, we only hold out 10% of our training data as the validation set. Alternatively, other estimates for test set accuracy could be experimented with, such as Affinity and Diversity [22], but because most Data Augmentation metrics for evaluating the utility of Synthetic Data are developed with non-tabular data modalities such as images in

[1]A variation of quantile sampling that lumps categories together to handle high-cardinality categorical columns is necessary, but it is not clear how to lump columns together.

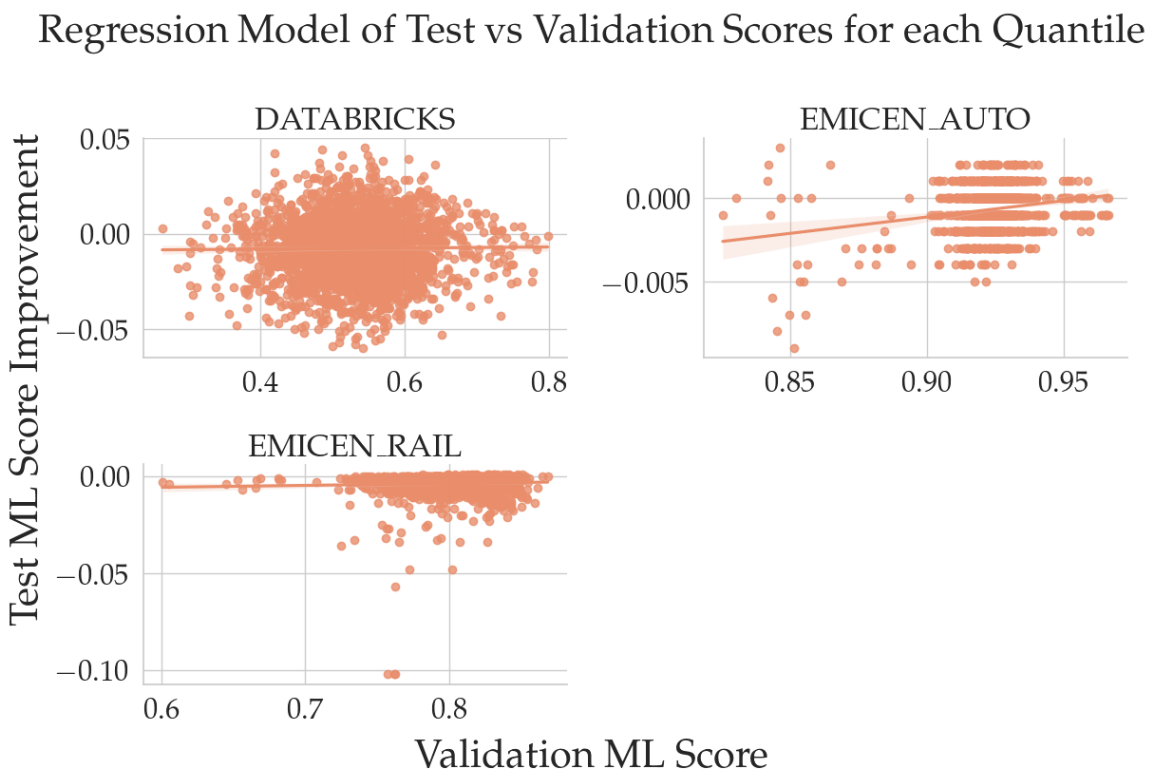


Figure 8-7: This figure presents linear regression plots with 95% confidence intervals of test set ML score improvement vs validation set ML score. Each point is from augmenting with synthetic data conditioned on a single quantile, and is generated by a 20 cluster AD-GMCM. We collect 10 points for each quantile. We observe that there is no consistent positive correlation across datasets between performance on the validation and test sets when augmenting with Synthetic Data.

mind, and their application in the tabular data setting is not well understood.

8.4 Amount of Synthetic Data Results

In this section, we are trying to answer **RQ4**: how many Synthetic Data samples should we generate for data augmentation? We expected `Tuned Unconditional Sampling` to improve performance when compared to `Unconditional Sampling`; however, as shown in figure 8-8, it does not. Recall that we tuned by using 10-fold cross validation, augmented with a fresh batch of Synthetic Data for each fold. Even though the data is tuned on the average ML score from 10-fold cross validation, it seems that the TPE parameter tuner is not able to learn to sample an amount of

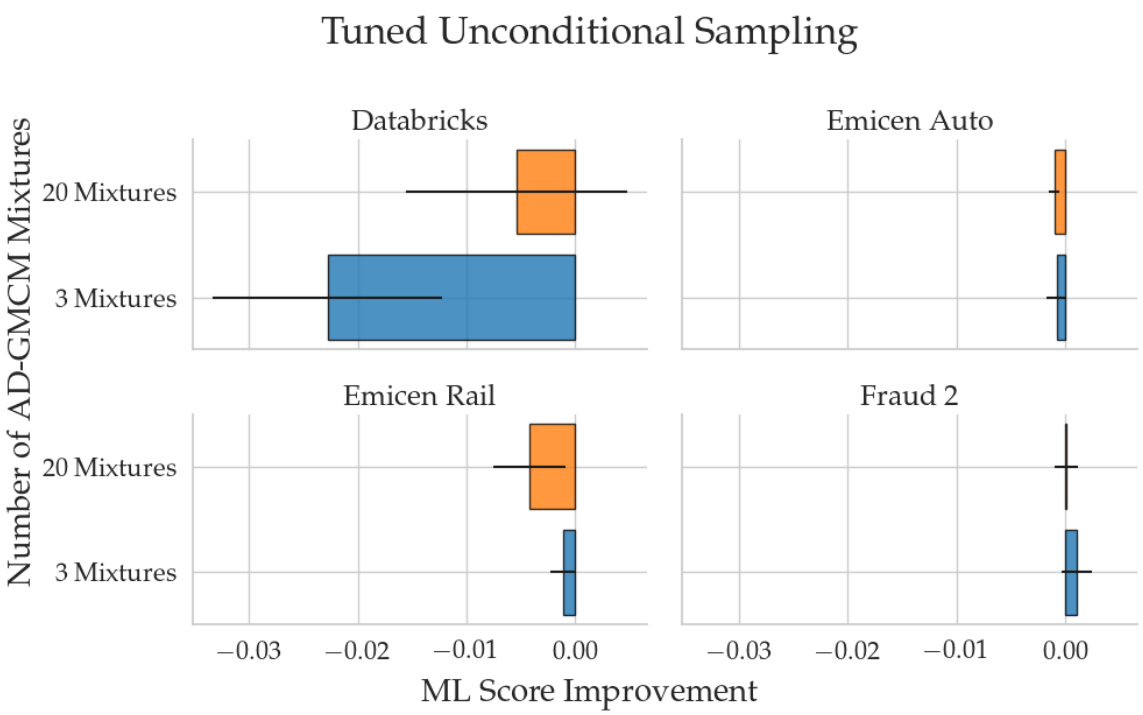


Figure 8-8: Results After Tuning the amount of Synthetic Data for Data Augmentation. We Report Results for 3 and 20 Cluster AD-GMCM.

Synthetic Data that provides a better ML score than is achieved by simply sampling half of the size of the training dataset (which we did in all other experiments). This is likely caused by the a lack of validation performance providing a good estimate of ML performance for tabular Data Augmentation, as we saw with quantile and mixture sampling. Future work should find better metrics for usability of Synthetic Data for Data Augmentation and experiment with different optimization methods for tuning the amount of data to sample.

Chapter 9

Discussion

In our experiments we showed that using Data Augmentation with tabular data from Synthetic Data generators can help improve ML performance. In this section we cover the implications of our results, the limitations of our experiments, and future work in this area.

9.1 Implications

Our experiments show three implications for using Copula-generated Synthetic Data for Data Augmentation.

1. Univariate modeling has a significant impact on the quality of Synthetic Data. `Noisy Label Encoders` were outperformed by the `Target Ordered Categorical` tranformers we introduce, showing that improved modeling of categorical data is important. Surprisingly, the `Univariate Continuous Transforms` seemed to not have a large impact on performance. Additionally, improved univariate modeling of mixed continuous and discrete data and NaNs did not seem to have a significant affect on ML performance either.

2. Variations in column dependency modeling, as we tried different Copula models and varied the number of mixtures in the GMCM model, did not have a significant effect on ML performance on our datasets. The GMCM model is very

difficult to train and likely is reaching local optimas in its training, causing this lack of improvement. The Segmented Gaussian Copula and Gaussian Copula models are also likely too restrictive to fit the data well. It would be best to use a different model architecture for learning complex dependencies such as GANs or VAEs.

3. Sampling methods such as `Quantile Sampling` and `Mixture Sampling` show promise, as we were able to get higher test set accuracies with them; however, these sampling methods are not practical because validation set accuracy did not give a good estimate of test set accuracy.

4. Determining the optimal amount of Synthetic Data is a task subject to a significant amount of noise, and using the TPE[5] hyperparameter tuner did not produce good results at learning the optimal amount of Synthetic Data. Having a better metric for estimating test set accuracy could help make tuning perform better. Other methods for tuning the amount of Synthetic Data may perform better as well in this task.

9.2 Limitations and Future Work

We only ran our experiments on public insurance and fraud datasets with fewer than 60 columns, so our results are limited to datasets of such size with similar properties and don't necessary hold for tabular datasets as a whole. We also only used insurance datasets in our analysis. It is important for future work to experiment on a broader selection of tabular datasets, both those that are more general and those that may come with different modeling challenges. We hope that SD test beds will improve access to more diverse datasets.

We only consider models based on the Gaussian Copula and Gaussian Mixture Copula generators as they allow for more interpretable experiments by setting the number of mixtures and the `multi-transforms`. We did not explore other methods for Data Augmentation such as with Vine Copulas, GAN-based generators, VAE-

based generators, or diffusion model based generators, which could better capture column dependency information and provide better Synthetic Data for Data Augmentation. Future effort should be put into assessing these models for tabular Data Augmentation and interpreting when and why a batch of Synthetic Data improves ML performance.

In our experiments, we assumed validation performance would provide a strong estimate of test set performance when performing tabular Data Augmentation, but it turns out validation set performance is not always a good estimate when performing tabular Data Augmentation. Developing better methods for determining, a priori, the ML value of Synthetic Data for Data Augmentation is an under-explored area for tabular data, and is a necessary problem for future work to solve in order for any tabular Data Augmentation methods to predictably improve ML performance. Reinforcement learning methods may be useful for developing better sampling methods and tuning the amount of synthetic data, and would be an interesting direction.

Chapter 10

Conclusion

In this thesis, we make two main contributions:

- We provide `SID`, a flexible framework to foster collaborations between entities with private data and data scientists seeking to validate Data Augmentation methods on real complex datasets.

- We investigate the real world applicability of tabular Data Augmentation on complex insurance claim and fraud datasets. We experimented with different univariate modeling strategies, different Copula models, and different sampling methods, and attempted to tune the amount of Synthetic Data we augment with. Assessing how Synthetic Data affects regression and classification performance on our datasets left us with mixed results, as well as many directions to explore to better understand tabular Data Augmentation and build better methods in the future.

We hope that future researchers and data scientists build off the generators and sampling methods in the SID library, and develop better tabular generative models and improved sampling methods while contributing to the open problem of how and when tabular Synthetic Data improves ML performance. A tabular dataset benchmark with several challenging tabular datasets from different domains could help foster innovation in the tabular data the same way that ImageNet [14] helped in CV;

however, most complex tabular datasets are kept private by entities in different domains such as healthcare, finance, or insurance. The proliferation of SD testbeds is essential for progress in tabular Data Augmentation as it will enable massive collaborations and improve data accessibility for researchers and data scientists, so that better ML methods can be validated on complex real-world datasets.

Appendix A

Glossary

Data Augmentation The practice of using Synthetic Data along with real data to train an ML model. Described in detail in section 5.

Gaussian Copula A synthetic data generator model for tabular data [41].

Generator A machine learning model that can generate new data samples..

Gini score Somer's D Gini Coefficient.

Kaggle An online data-science community where datasets and data-science competitions are hosted by users.

loss Amount of money paid out for an insurance client.

Synthetic Data Fake data generated by a generator.

Appendix B

Gaussian Mixture Copula Model Fitting Algorithms

Variable	Definition
τ	Maximum number of epochs for AD and PEM GMCM algorithms.
t	Is the current epoch number during training. We index variables that are updated over epochs using the notation $\Sigma_k^{(t)}$ to represent the value of Σ_k after epoch t.
Z	Represents the latent variables for the PEM GMCM algorithm. $Z \in R^{n \times r}$ and $Z_{i,:}$ represents a categorical distribution over the clusters data point $D_{i,:}$ may belong to.
ϵ	Is the convergence threshold. If loss over two PEM GMCM epochs changes by less than ϵ, then training is stopped early.
ζ	Is the number of noise columns added to a dataset. Noise columns are columns that follow an independent normal distribution.

Table B.1: GMCM Fitting Notation Table

Here we describe the AD-GMCM[27] (and our implementation of it) and the PEM-GMCM[6, 31] algorithms for fitting a GMCM to data. We use the GMCM notation defined in table 4.4 and additionally define some extra fitting algorithm related pa-

rameters in B.1.

B.1 Auto-Differentiation (AD) GMCM

AD-GMCM, uses automatic differentiation and gradient descent to learn the copula parameters. Previous work[27] implemented this algorithm in R, and for this thesis we re-implemented it in python. The AD-GMCM algorithm is copied for convenience here in algorithm 2, using the notation defined in this thesis.

Algorithm 2 AD-GMCM [27]

Input : U - data with uniform marginals
r - number of clusters
`InitMethod` - For initializing $\pi^{(0)}, \mu^{(0)}, \Sigma^{(0)}$ or $V^{(0)}$
τ - number of epochs.

Initialize $\pi^{(0)}, \mu^{(0)}, V^{(0)}$ based on `InitMethod`

for $t \leftarrow 1$ **to** τ **do**

  Reset y

  $\Omega_j^{(0)} \leftarrow \Phi_j^{-1}(U_j; \pi^{(t-1)}, \mu_k^{(t-1)}, \Sigma_k^{(t-1)})$

  Gradient Step

  $\pi_k^{(t)} \leftarrow \pi_k^{(t-1)} + \eta \frac{\partial \mathcal{L}}{\partial \pi_k}$.

  $\mu_k^{(t)} \leftarrow \mu_k^{(t-1)} + \eta \frac{\partial \mathcal{L}}{\partial \mu_k}$.

  $V_k^{(t)} \leftarrow V_k^{(t-1)} + \eta \frac{\partial \mathcal{L}}{\partial V_k}$.

  $\Sigma_k^{(t)} \leftarrow V_k^{(t)} V_k^{(t)} T$.

end

Output: π, μ, Σ

We note that different `InitMethod`s can be used to initialize the starting parameters of the GMCM. Since $\mathcal{L}$ is not convex, convergence is not assured and AD-GMCM is very sensitive to starting points, so it is important to try different initializations. Below we define the initialization methods we try:

- `ClusterMethods`

  1. Use a cluster method such as Kmeans or Gaussian Mixture Model to cluster U into r clusters.
  2. $\pi_k^{(0)} \leftarrow$ empirical probability of cluster k.
     $\mu_k^{(0)} \leftarrow$ mean of cluster k.

$\Sigma_k^{(0)} \leftarrow$ covariance matrix of cluster k.
$V_k^{(0)} \leftarrow \sqrt{\Sigma_k^{(0)}}$.

- `RandomInit` Randomly sample r matrices V_k, vectors μ_k, and probabilities π_k s.t. $\sum \pi_k = 1$. Set $\Sigma_k = V_k V_k^T$ (this ensures Σ_k is positive definite).

Note that any auto-differentiation library can be used to compute the gradients for algorithm 2, and in the SID library we use pytorch [40] and the adam optimizer [28].

Gaussian Mixture copula suffer from identifiability issues as there is no unique representation for any Gaussian Mixture Copula as you can translate means or multiple covaraiances while maintaining the sample copula model. This can cause issues when performing inference on the Gaussian Mixture Copula as you can get exploding and vanishing covariance or mean parameter values. To try to overcome this, we follow the same procedure as previous work [6], after clustering the data we anchor the means by subtracting the mean of the first cluster from all clusters. This makes the first mean be at the origin and all clusters are relative to it. We then divide all covariance matrices by the diagonal of the covariance of of the first cluster. This makes all covariance matrices relative to the first cluster having unit covariance. Gradients are also clipped. Since the inverse marginal CDF, $\Phi_j^{-1}(U_j)$ has no closed form, we also use the grid search and interpolation method from previous work [27, 6] to calculate the inverse marginal CDF.

Numerical Stability Techniques

AD-GMCM can have very unstable training if run naively. Here we provide some techniques that make training more stable. For our experiments a learning rate of $1e-2$ with 200 epochs is generally our training setup. One major limitation with GMCM is the unidentifiability of parameters. GMCM parameters do not define a unique model because you can translate cluster mean values and scale cluster means and covariances while maintaining the same model. This can lead to issues with exploding parameter values when optimizing the parameters of the GMCM using

Automatic Differentiation. To resolve this, we anchor the first mixture to have a mean of 0 and have unit variance in each dimension as previous work has done [6].

Additionally, the GMCM model struggles with singularities in the covariance which make the loss non-finite. To resolve this we optimize the log-cholesky decomposition of the covariance for each mixture as described in [6]. Note that the log-cholesky decomposition is the cholesky decomposition but with the log function applied elementwise to the diagonal. In addition we add a small ϵ to the diagonal of the cholesky decomposition to ensure the covariance matrix is non-singular. We found that scaling the initial parameters so the means are on the order of 100 (by multiplying the data by 100 before running the initmethods on it) ensures that ϵ is very small relative to the initial cholesky values and thus does not bias the model significantly.

B.2 PEM GMCM

We use the same PEM algorithm from [6, 31] and provide a python implementation in the SID library as well. Denote $\psi(\Omega_{i,:}; (\mu, \Sigma))$ as the probability of data point i in Ω assuming a multivariate normal distribution with mean and covariance μ and Σ. See algorithm 3 for the PEM GMCM algorithm.

Algorithm 3 PEM-GMCM [6, 31]

Input : U - data with uniform marginals
r - number of mixtures
`InitMethod` - Cluster method for initializing $\pi^{(0)}, \mu^{(0)}, \Sigma^{(0)}$
τ - max iterations.

$U \leftarrow U * n/(n+1) + 1/2(n+1)$ as this helps avoid large inverse CDF values in copula density. Note that n is the number of data points.

Initialize $\pi^{(0)}, \mu^{(0)}, \Sigma^{(0)}$ based on `InitMethod` with r mixtures

for $t \leftarrow 1$ **to** τ **do**

  Reset Ω

  $\Omega_j \leftarrow \Phi_j^{-1}(U_j; \pi^{(t-1)}, \mu_k^{(t-1)}, \Sigma_k^{(t-1)})$

  E Step for GMM

  for $k \leftarrow 1$ **to** r **do**

    $Z_{i,k} = \frac{\psi(\Omega_{i,:}; (\mu_k^{(t)}, \Sigma_k^{(t)}))}{\sum_{j=1}^{r} \psi(\Omega_{i,:}; (\mu_j^{(t)}, \Sigma_j^{(t)}))}$ Setup latent variables (probability data-point i in mixture k)

    if $\sum_{i=1}^{n} Z_{i,k} = 0$ **then**

      Remove mixture k as it has 0 probability. This handles cases where we have a singular covariance matrix.

    end

  end

  M Step for GMM

  $\pi_k^{(t)} \leftarrow \sum_{i=1}^{n} Z_{i,k}$.

  $\mu_k^{(t)} \leftarrow \frac{\sum_{i=1}^{n} Z_{i,k} \Omega_{i,:}}{\sum_{i=1}^{n} Z_{i,k}}$.

  $\Sigma_k^{(t)} \leftarrow \frac{\sum_{i=1}^{n} Z_{i,k} (\Omega_{i,:} - \mu_k^{(t)})(\Omega_{i,:} - \mu_k^{(t)})^T}{\sum_{i=1}^{n} Z_{i,k}}$.

  if $|\mathcal{L}^{(t)} - \mathcal{L}^{(t+1)}| < \epsilon$ **then**

    Convergence achieved, break loop

  end

end

Output: π, μ, Σ

Appendix C

Full Experimental Results

C.1 Univariate Results

Dataset Baseline Transform	DATABRICKS 0.626	EMICEN_AUTO 0.953	EMICEN_RAIL 0.866	FRAUD_2 0.908
OM	0.626 ± 0.011	0.938 ± 0.001	0.864 ± 0.001	0.908 ± 0.002
TOI	0.628 ± 0.007	0.952 ± 0.001	0.863 ± 0.004	0.908 ± 0.002
TOR	0.628 ± 0.01	0.951 ± 0.001	0.864 ± 0.002	0.908 ± 0.002
SDV	0.609 ± 0.014	0.939 ± 0.002	0.855 ± 0.003	0.898 ± 0.002
SDV_CA	0.611 ± 0.015	0.943 ± 0.001	0.864 ± 0.001	0.901 ± 0.003

Table C.1: Gaussian Copula Experimental Results - `Beta` Distribution with `Unconditional Sampling`

Dataset	DATABRICKS	EMICEN_AUTO	EMICEN_RAIL	FRAUD_2
Baseline	0.626	0.953	0.866	0.908
0.972	0.826			
Transform				
OM	0.63 ± 0.014	0.936 ± 0.002	0.864 ± 0.001	0.908 ± 0.002
TOI	0.625 ± 0.015	0.952 ± 0.002	0.863 ± 0.001	0.907 ± 0.002
TOR	0.618 ± 0.013	0.951 ± 0.001	0.863 ± 0.001	0.907 ± 0.002
SDV	0.62 ± 0.016	0.945 ± 0.002	0.793 ± 0.003	0.901 ± 0.002
SDV_CA	0.61 ± 0.014	0.941 ± 0.002	0.862 ± 0.002	0.898 ± 0.001

Table C.2: Gaussian Copula Experimental Results - 3 Mixture `BMM` Distribution with `Unconditional Sampling`

Dataset	DATABRICKS	EMICEN_AUTO	EMICEN_RAIL	FRAUD_2
Baseline	0.626	0.953	0.866	0.908
Transform				
OM	0.625 ± 0.013	0.935 ± 0.003	0.864 ± 0.001	0.908 ± 0.002
TOI	0.63 ± 0.012	0.951 ± 0.001	0.863 ± 0.002	0.909 ± 0.001
TOR	0.632 ± 0.015	0.951 ± 0.001	0.863 ± 0.001	0.909 ± 0.001
SDV	0.616 ± 0.023	0.933 ± 0.003	0.864 ± 0.002	0.895 ± 0.003
SDV_CA	0.609 ± 0.018	0.938 ± 0.002	0.863 ± 0.001	0.897 ± 0.002

Table C.3: Gaussian Copula Experimental Results - varying `multivariate-transforms` with `Spline` Distribution with `Unconditional Sampling`

C.2 Copula Dependency Results

Dataset Baseline Transform	DATABRICKS 0.626	EMICEN_AUTO 0.953	EMICEN_RAIL 0.866	FRAUD_2 0.908
TOR	0.62 ± 0.014	0.951 ± 0.002	0.862 ± 0.002	0.909 ± 0.001

Table C.4: Segmented Gaussian Copula - `Unconditional Sampling`

Dataset Baseline Transform	DATABRICKS 0.626	EMICEN_AUTO 0.953	EMICEN_RAIL 0.866	FRAUD_2 0.908
TOR	0.619 ± 0.015	0.952 ± 0.001	0.862 ± 0.001	✗

Table C.5: PEM-GMCM, 3 Mixture - `Unconditional Sampling`

Dataset Baseline Transform	DATABRICKS 0.626	EMICEN_AUTO 0.953	EMICEN_RAIL 0.866	FRAUD_2 0.908
TOR	0.628 ± 0.015	0.952 ± 0.001	0.864 ± 0.002	0.908 ± 0.001

Table C.6: AD-GMCM - 3 Mixture - `Unconditional Sampling`

Dataset	DATABRICKS	EMICEN_AUTO	EMICEN_RAIL	FRAUD_2
Baseline	0.626	0.953	0.866	0.908
Transform				
TOR	0.626 ± 0.013	0.952 ± 0.001	0.861 ± 0.002	0.909 ± 0.001

Table C.7: AD-GMCM - 20 Mixture - `Unconditional Sampling`

C.3 Sampling Methods

C.3.1 Mixture Sampling

Dataset Baseline Mixture	DATABRICKS 0.626	EMICEN_AUTO 0.953	EMICEN_RAIL 0.866	FRAUD_2 0.908
1	0.636 ± 0.022	0.952 ± 0.001	0.861 ± 0.002	0.909 ± 0.001
2	0.634 ± 0.023	0.938 ± 0.001	0.864 ± 0.001	0.907 ± 0.001
3	0.62 ± 0.02	0.951 ± 0.001	0.863 ± 0.003	0.906 ± 0.002

Table C.8: AD-GMCM - 3 mixture `mixture sampling` with `TOR Multivariate-Transform`

Dataset Baseline	DATABRICKS 0.626	EMICEN_AUTO 0.953	EMICEN_RAIL 0.866	FRAUD_2 0.908
1	0.639 ± 0.013	0.953 ± 0.001	0.865 ± 0.001	0.911 ± 0.001
2	0.635 ± 0.017	0.953 ± 0.001	0.864 ± 0.002	0.91 ± 0.002
3	0.633 ± 0.018	0.953 ± 0.001	0.864 ± 0.001	0.91 ± 0.001

Table C.9: AD-GMCM - 20 mixture `mixture sampling` with `TOR Multivariate-Transform`. Top three mixtures.

C.3.2 Quantile Sampling

Dataset	Quantile Column	Score
EMICEN_AUTO	Income	0.955 ± 0.0
	Total Claim Amount	0.954 ± 0.001
	EmploymentStatus	0.954 ± 0.001
EMICEN_RAIL	DEPARTURE CITY	0.867 ± 0.001
	DEPARTURE CITY	0.867 ± 0.001
	DEPARTURE CITY	0.867 ± 0.001
DATABRICKS	incident_hour_of_the_day	0.638 ± 0.016
	auto_year	0.633 ± 0.017
	police_report_available	0.632 ± 0.019

Table C.10: AD-GMCM - 3 mixture `quantile sampling` - `Multivariate-Transform TOR`. Top three Quantiles scores for each Dataset and the corresponding Quantile Columns that was conditioned on.

Dataset	Quantile Column	Score
EMICEN_AUTO	Total Claim Amount	0.954 ± 0.001
	Policy	0.954 ± 0.001
	Education	0.954 ± 0.001
EMICEN_RAIL	DAMAGED	0.866 ± 0.001
	DEPARTURE CITY	0.866 ± 0.0
	DAMAGED	0.866 ± 0.001
DATABRICKS	incident_city	0.636 ± 0.017
	insured_hobbies	0.632 ± 0.015
	auto_model	0.632 ± 0.017

Table C.11: AD-GMCM - 20 mixture `quantile sampling` `Multivariate-Transform TOR`. Top three quantiles scores for each Dataset and the corresponding Quantile Columns that was conditioned on.

C.4 Amount of Synthetic Data

Dataset	Transform	Actual Amount SD	score
DATABRICKS	TOR	0.374	0.603 ± 0.011
EMICEN_AUTO	TOR	0.410	0.952 ± 0.001
EMICEN_RAIL	TOR	0.090	0.865 ± 0.001
FRAUD_2	TOR	0.811	0.909 ± 0.001

Table C.12: AD-GMCM, 3 mixture - `Tuned Unconditional Sampling`

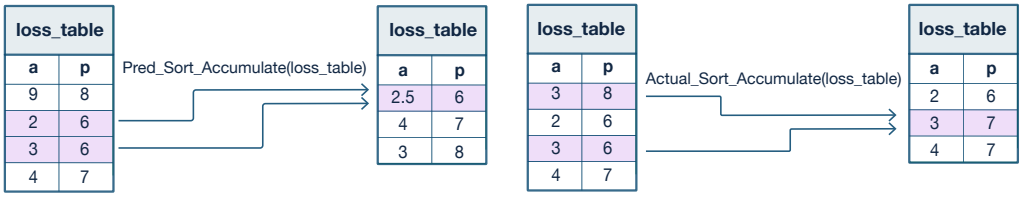
Dataset	Transform	Actual Amount SD	score
DATABRICKS	TOR	0.081	0.62 ± 0.019
EMICEN_AUTO	TOR	0.417	0.952 ± 0.001
EMICEN_RAIL	TOR	0.085	0.865 ± 0.001
FRAUD_2	TOR	0.142	0.909 ± 0.002

Table C.13: AD-GMCM, 20 mixture - `Unconditional Tuned Sampling`

Appendix D

Gini Score Algorithm

Here we share the algorithm for calculating Gini score. Note that instead of calculating the area between the lorenze curves and the random model, we calculate the area between the lorenze curves and the X-axis. The same inaccuracy when using the rectangle rule and accuracy when using the trapezoid rule discussed in section 3.1 still holds for this area calculation.



(a) Actual_Sort_Accumulate Algorithm Example. Accumulates and sorts based on predicted loss column, p.

(b) Pred_Sort_Accumulate Algorithm Example. Accumulates and sorts based on actual loss column, a.

Figure D-1: Pred_Sort_Accumulate and Actual_Sort_Accumulate Algorithm Examples. p and a columns represent predicted and actual loss values.

Algorithm 4 Gini_Score(`actual_loss`, `pred_loss`)

Input : `actual_loss` - N size vector of actual losses.
`pred_loss` - N size vector of predicted losses from ML model.

`loss_table` ← Two column table, first column is `actual_loss` and the second column is `pred_loss`

`actual_loss_table` ← `Actual_Sort_Accumulate`(`loss_table`). The `Actual_Sort_Accumulate` function takes the average of rows that have the same `actual_loss` value in order to break ties and then sorts the table by `actual_loss` as shown in table D-1b

`pred_loss_table` ← `Pred_Sort_Accumulate`(`loss_table`). The `Pred_Sort_Accumulate` function takes the average of rows that have the same `pred_loss` value in order to break ties and then sorts the table by `pred_loss` as shown in table D-1a

Normalize all the tables by dividing every column by the sum of all elements in the column. Thus every column sums to 1.

`cum_actual_loss_table` ← Take the cumulative sum applied to both columns in `actual_loss_table`.

`cum_pred_loss_table` ← Take the cumulative sum applied to both columns in `pred_loss_table`.

`area_under_equality` ← 0.5 This represents the area under a completely random model.

`area_under_perfect` ← The area under the cumulative sum of the actual loss when we sort by the actual predictions. This is calulated by using the trapezoid rule to calculate the area under the line with $x = \frac{1}{N}, \ldots, \frac{N}{N}$ and $y =$`cum_actual_loss_table`.

`area_under_lorenze` ← The area under the cumulative sum of the actual loss when we sort by predictions. This is calculated by using the trapezoid rule to calculate the area under the line with $x = \frac{1}{N}, \ldots, \frac{N}{N}$ and $y =$`cum_pred_loss_table`. Note that this curve is called the Lorenze curve.

$A \leftarrow$ `area_under_equality` - `area_under_lorenze`

$A + B \leftarrow$ `area_under_equality` - `area_under_actual`

`gini` $\leftarrow \frac{A}{A+B}$.

Output: `gini` - Somer's D Gini score

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[2] Allstate. Allstate claims severity, how severe is an insurance claim? Kaggle competition, 2016.

[3] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. Not enough data? deep learning to the rescue!, 2019.

[4] Azin Asgarian, Rohit Saha, Daniel Jakubovitz, and Julia Peyre. Autofraudnet: A multimodal network to detect fraud in the auto insurance industry, 2023.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[6] Anders Ellern Bilgrau, Poul Svante Eriksen, Jakob Gulddahl Rasmussen, Hans Erik Johnsen, Karen Dybkaer, and Martin Boegsted. Gmcm: Unsupervised clustering and meta-analysis using gaussian mixture copula models. *Journal of Statistical Software*, 70(2):1â23, 2016.

[7] Yabir Canario. Home insurance, 2007-2012 polices of a home insurance company. Kaggle dataset, 2017.

[8] Ihsan Chaoubi, Camille Besse, HÃ©lÃ¨ne Cossette, and Marie-Pier CÃ´tÃ©. Micro-level reserving for general insurance claims using a long short-term memory network, 2022.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, jun 2002.

[10] Shuxiao Chen, Edgar Dobriban, and Jane H Lee. A group-theoretic framework for data augmentation, 2020.

[11] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016.

[12] Tri Dao, Albert Gu, Alexander J. Ratner, Virginia Smith, Christopher De Sa, and Christopher RÃ©. A kernel theory of modern data augmentation, 2018.

[13] Siddharth Kulkarni Databricks. Insurance claims - fraud detection. Kaggle dataset, December 2021.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[15] Fabrizio Durante and Carlo Sempi. *Principles of copula theory*. CRC Press, Taylor et Francis Group, 2021.

[16] Emicen. Auto insurance claims â automobile insurance claims including location, policy type and claim amount. Published to EmicenPatterns Product Website.

[17] Emicen. Rail insurance claims â 2013 insurance claims made by rail companies. Published to EmicenPatterns Product Website.

[18] Mustafa Fatakdawala. Insurance claims fraud data, fraud detection on insurance. Kaggle dataset, December 2021.

[19] Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp, 2021.

[20] Edward W Jed Frees, Glenn Meyers, and A David Cummings. Insurance ratemaking and a gini index. *J. Risk Insur.*, 81(2):335–366, June 2014.

[21] F. N. Fritsch and J. Butland. A method for constructing local monotone piecewise cubic interpolants. *SIAM Journal on Scientific and Statistical Computing*, 5(2):300–304, 1984.

[22] Raphael Gontijo-Lopes, Sylvia J. Smullin, Ekin D. Cubuk, and Ethan Dyer. Affinity and diversity: Quantifying mechanisms of data augmentation, 2020.

[23] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[24] Guillermo Iglesias, Edgar Talavera, Ángel González-Prieto, Alberto Mozo, and Sandra Gómez-Canaval. Data augmentation techniques in time series domain: a survey and taxonomy. *Neural Computing and Applications*, 35(14):10123–10145, mar 2023.

[25] Prudential Life Insurance. Prudential life insurance assessment, can you make buying life insurance easier? Kaggle competition, 2015.

[26] Siva Rajesh Kasa, Sakyajit Bhattacharya, and Vaibhav Rajan. Gaussian mixture copulas for high-dimensional clustering and dependency-based subtyping. *Bioinformatics*, 36(2):621–628, 08 2019.

[27] Siva Rajesh Kasa and Vaibhav Rajan. Improved inference of gaussian mixture copula model for clustering and reproducibility analysis using automatic differentiation. *Econometrics and Statistics*, 22:67–97, apr 2022.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[29] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[30] Varun Kumar, Ashutosh Choudhary, and Eunah Cho. Data augmentation using pre-trained transformer models, 2020.

[31] Qunhua Li, James B. Brown, Haiyan Huang, and Peter J. Bickel. Measuring reproducibility of high-throughput experiments. *The Annals of Applied Statistics*, 5(3), sep 2011.

[32] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Generating high-fidelity, synthetic time series datasets with doppelganger. *CoRR*, abs/1909.13403, 2019.

[33] Pedro Machado, Bruno Fernandes, and Paulo Novais. Benchmarking data augmentation techniques forÂ tabular data. In Hujun Yin, David Camacho, and Peter Tino, editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2022*, pages 104–112, Cham, 2022. Springer International Publishing.

[34] JosÃ© M. Maisog, Wenhong Li, Yanchun Xu, Brian Hurley, Hetal Shah, Ryan Lemberg, Tina Borden, Stephen Bandeian, Melissa Schline, Roxanna Cross, Alan Spiro, Russ Michael, and Alexander Gutfraind. Using massive health insurance claims data to predict very high-cost claimants: a machine learning approach, 2019.

[35] David Meyer and Thomas Nagler. Synthia: multidimensional synthetic data generation in python. *Journal of Open Source Software*, 6(65):2863, 2021.

[36] Mimi Mukherjee and Matloob Khushi. SMOTE-ENC: A novel SMOTE-based method to generate synthetic data for nominal and continuous features. *Applied System Innovation*, 4(1):18, mar 2021.

[37] Zahier Nasrudin. Travel insurance, travel insurance with the target attribute of: Claim status (yes or no). Kaggle dataset, 2018.

[38] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans, 2016.

[39] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*, 11(10):1071–1083, jun 2018.

[40] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[41] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410, 2016.

[42] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[43] Porto Seguro. Porto seguroâs safe driver prediction, predict if a driver will file an insurance claim next year. Kaggle competition, 2017.

[44] Amritha Sharma. Insurance fraud data, insurance fraud data for january 2011 - december 2017. Kaggle dataset, June 2022.

[45] Martin J. Sklar. Fonctions de repartition a n dimensions et leurs marges. 1959.

[46] Edossa Merga Terefe. Tree-based machine learning methods for vehicle insurance claims size prediction, 2023.

[47] Ashutosh Tewari, Michael J. Giering, and Arvind Raghunathan. Parametric characterization of multimodal distributions with non-gaussian modes. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 286–292, 2011.

[48] Jason Wei and Kai Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China, November 2019. Association for Computational Linguistics.

[49] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, aug 2021.

[50] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan, 2019.

[51] Mingle Xu, Sook Yoon, Alvaro Fuentes, and Dong Sun Park. A comprehensive survey of image augmentation techniques for deep learning, 2022.

[52] Suorong Yang, Weikang Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, and Furao Shen. Image data augmentation for deep learning: A survey, 2022.

[53] Zhenyu Yang, Yantao Li, and Gang Zhou. Ts-gan: Time-series gan for sensor-based health data augmentation. *ACM Trans. Comput. Healthcare*, 4(2), apr 2023.

[54] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[55] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019.

[56] Yanwei Zhang, Vanja Dukic, and James Guszcza. A bayesian non-linear model for forecasting insurance loss payments. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 175(2):637–656, 2012.

[57] Zilong Zhao, Aditya Kunar, Hiek Van der Scheer, Robert Birke, and Lydia Y. Chen. Ctab-gan: Effective table data synthesizing, 2021.