A Deep Generative Model Framework for Creating High Quality Synthetic Transaction Sequences

by

© **Kyle Nickerson**

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Supervisors

Dr. Ting Hu

Dr. Antonina Kolokolova

Dr. Terrence Tricco

Department of Computer Science

Memorial University

August 2023

St. John's, Newfoundland and Labrador, Canada

Abstract

Synthetic data are artificially generated data that closely model real-world measurements, and can be a valuable substitute for real data in domains where it is costly to obtain real data, or privacy concerns exist. Synthetic data has traditionally been generated using computational simulations, but deep generative models (DGMs) are increasingly used to generate high-quality synthetic data.

In this thesis, we create a framework which employs DGMs for generating high-quality synthetic transaction sequences. Transaction sequences, such as we may see in an online banking platform, or credit card statement, are important type of financial data for gaining insight into financial systems. However, research involving this type of data is typically limited to large financial institutions, as privacy concerns often prevent academic researchers from accessing this kind of data. Our work represents a step towards creating shareable synthetic transaction sequence datasets, containing data not connected to any actual humans.

To achieve this goal, we begin by developing *Banksformer*, a DGM based on the transformer architecture, which is able to generate high-quality synthetic transaction sequences. Throughout the remainder of the thesis, we develop extensions to Banksformer that further improve the quality of data we generate. Additionally, we perform extensively examination of the quality synthetic data produced by our method, both with qualitative visualizations and quantitative metrics.

Lay summary

In this thesis, we develop a framework for creating synthetic banking data. The main reason we are interested in this goal, is to help minimize privacy risks for organizations that use computational tools to analyze banking data. The basic idea of our work, is to use a machine learning algorithm to generate new *synthetic* banking datasets, which contain artificial data not linked to any real customer. In order to be useful for a wide range of applications, we aim to create high quality synthetic data, which have the same statistical properties of real banking data.

There is some limited existing work on this topic. The most common method for generating data of this type is using a simulator, and there are some simulators tailored to generating transaction sequences, however they are limited by their model assumptions. Our work leverages recent research on deep generative models, which are a type of AI model which can be used to generate data. Specifically, we modify the *transformer* architecture, which is able to generate realistic text sequences, to create *Banksformer*, the basis of our synthetic data framework. We choose to base our model on the transformer, because it is able to model complex statistical patterns, which previous simulator approaches could not. Further, as our model is *data-driven*, it is able to automatically learn the statistical patterns from a target real dataset, making it easier to adapt to different domains. For instance, many banks cater to specific clients, based on geographic and socioeconomic factors. By adopting a data-driven approach, we can create synthetic datasets tailored to specific customer groups, without changing our model. Instead, we would just need to train our model on whatever kind of customer data we wish to generate. Our work has produced a high quality synthetic banking data generator, and a framework for evaluating this type of synthetic data, which can be used to help preserve private user data, while enabling machine learning solutions.

Acknowledgements

There are many people without whom I would not have finished this thesis. I thank my group of three amazing supervisors, Dr. Terrence Tricco, Dr. Antonina Kolokolova and Dr. Ting Hu, for all the guidance you have provided me, and for your help preparing this thesis. I also thank the members of my Supervisory committee – Dr. Andrew Vardy and Dr. Amilcar Soares – for your guidance and helpful feedback.

I would like to thank Ben Morrison for many helpful discussions during the early phases of my research, and Bo Simango for encouraging me to get involved with Mitacs and Verafin. Friends and family for helping me stay sane throughout writing this.

I wish to acknowledge the support of Mitacs through Accelerate funding for applied research. I also acknowledge funding from Verafin, which supported this work, and I thank everyone I've worked with at Verafin for many helpful conversations and guidance, specifically Dr. Charles Robertson, Dr. John Hawkin, and Dr. Farzaneh Shoeleh.

Advanced computing resources were provided by the Digital Research Alliance of Canada, the organization responsible for digital research infrastructure in Canada, and ACENET, the regional partner in Atlantic Canada. ACENET is funded by Industry Science & Economic Development, the provinces of New Brunswick, Newfoundland & Labrador, Nova Scotia and Prince Edward Island, as well as the Atlantic Canada Opportunities Agency.

Statement of contributions

The contents of Chapter's 1, 3, 5 and 6 are the original work of Kyle Nickerson, with supervisors Ting Hu, Antonina Kolokolova and Terrence Tricco providing general advice and help with editing.

The contents of Chapter 2 were published in [114], which was co-authored by Kyle Nickerson, Terrence Tricco, Antonina Kolokolova, Farzaneh Shoeleh, Charles Robertson, John Hawkin and Ting Hu. Kyle Nickerson designed the Banksformer model, proposed the evaluation framework, conducted the Banksformer experiments, and wrote the paper draft. Farzaneh Shoeleh conducted the DoppelGANger experiments. Terrence Tricco, Antonina Kolokolova, Charles Robertson, John Hawkin and Ting Hu supervised the project, contributed to the development of the evaluation framework, and editing the paper.

The contents of Chapter 4 were published in [113], which was co-authored by Kyle Nickerson, Antonina Kolokolova, and Ting Hu. Kyle Nickerson designed the Neuro-MAP-Elites algorithm, proposed the evaluation framework, conducted the experiments, and wrote the paper draft. Antonina Kolokolova and Ting Hu supervised the project, contributed to the development of the evaluation framework, and editing the paper.

Table of contents

List of tables

List of figures

List of abbreviations

ACGAN	Auxiliary Classifier Generative Adversarial Network
AR	Autoregressive
AUC	Area Under The Receiver Operator Curve
BF	Banksformer
cGAN	Conditional Generative Adversarial Network
DG	DoppelGANger
DGM	Deep Generative Model
DP	Differential Privacy
DP-SGD	Differentially Private Stochastic Gradient Descent
EA	Evolution Algorithm
GAN	Generative Adversarial Network
GM	Generative Model
GP	Genetic Programming
JSD	Jensen Shannon Divergence
LDA	Latent Dirichlet Allocation
LGP	Linear Genetic Program
MHA	Multi-head Attention
NF	Normalizing Flow
NME	Neuro-map-elites
NMT	Neural Machine Translation
NSLC	Novelty Search With Local Competition
PCA	Principal Component Analysis
PATE	Private Aggregation Of Teacher Ensembles
PE	Positional Encoding
PII	Personally Identifiable Information
QD	Quality Diversity
RNN	Recurrent Neural Network
TD	Transformer Decoder
TG	TimeGAN
VAE	Variational Autoencoders

Chapter 1

Introduction

The field of banking and finance has undergone a significant transformation with the advent of digital technologies. The vast amount of data generated by these technologies presents unique opportunities for researchers to analyze and understand the behavior of financial systems, and detect harmful or malicious behavior. However, accessing real-world banking data for research purposes is often challenging due to privacy concerns and regulatory restrictions. Additionally, banks and other financial institutions often have limited incentives to share their data, particularly with third-party researchers.

As a result, there has been a growing interest in the use of synthetic banking data for research purposes [6, 17, 79], which allows researchers to analyze and understand the behavior of financial systems, without the need for direct access to real-world data. Synthetic data is data that has been artificially created to mimic the statistical characteristics of real-world data, as opposed to data created by measuring a real-world process. This approach allows researchers to generate large volumes of data that are representative of the real world, without relying on private information from individuals and organizations.

Despite this growing interest, there are still many open problems limiting the quality of modern synthetic banking data. Many types of banking data, are naturally model as sequences, whether those sequences describe the price of an asset over time or transactions on an account. Sequences of transactions are particularly interesting, because transactions typically occur at irregular intervals, unlike prices of an asset, which may vary continuously, but are measured at fixed intervals. A major challenge

of modeling transaction sequences, is that we must both *when* and *what* the next transaction will be. While there are already many tools for modeling irregular intervals, such as Poisson point processes [153], these make many simplifying assumptions, such as events being independent, which prevent them from being useful in creating high quality banking data, as there are often relationships between non-consecutive items in these sequences.

The goal of this thesis is to develop a methodology for generating realistic synthetic transactions sequences, which can mimic the statistical properties of real datasets. In the remainder of this Chapter, we provide an extensive literature review on the background work on which our framework is based, and provide an overview of the contributions made by this thesis.

1.1 Synthetic Data

The term *synthetic data* has been used in many contexts throughout the literature. Very generally, synthetic data can be defined as any data not obtained by direct measurement of a real-world process. This broad definition includes data produced by: simulation, data engineers, adding noise to real data, anonymising real data, and learning a model of real data.

Synthetic data can be useful in many scenarios. Once we have a properly configured synthetic data generator for a given domain, we can easily generate new samples from the target domain as needed. In areas where it is difficult or expensive to obtain data, such as medical imaging, synthetic data can be used to augment real datasets [24, 45, 49], or completely replace real data [57, 31]. Using synthetic data to augment real datasets is helpful when dealing with datasets that under-represent certain classes, if we can obtain new samples from these under-represented classes. Studies have shown that training machine learning classifiers on a combination of real and synthetic data leads to greater classification accuracy on real data, particularly when the datasets are small [45, 24], or contain rare classes [49]. However, as noted in [24], in some cases it is difficult to obtain high-quality synthetic samples from under-represented classes using deep generative models (DGMs). Completely replacing real data with synthetic data is often motivated by privacy or legal concerns associated with sharing data. In addition to medical data [57, 31], this use case is common in

other domains with similar concerns, such as retail banking data [6].

The idea of creating altered versions of datasets to preserve privacy is not new. It dates back to at least 1993 when Donald Rubin [142] used imputation techniques to create a synthetic version of the American Decennial Census. Rubin's method aimed to mask sensitive info and then train multiple regression models to estimate the masked information. This method, which he called *multiple imputation*, then created the final dataset by using each regression model to compute an estimate of the masked data and combining these predictions into the values used in the final synthetic dataset.

Generally, modern approaches to generating synthetic data are grouped into two categories: Hand-designed (simulators) and data-driven (models). Hand-designed approaches involve manually designing a simulator that simulates certain aspects of the real world. This approach can be more labor-intensive than data-driven approaches, as often domain experts are needed to develop simulators, and the simulators are usually designed for specific tasks. Further, the quality of the simulated data depends heavily on the simulator's design and may not accurately characterize the true data. The main benefit of hand-designed simulators is that they are not trained directly on the private data and thus are not concerned with exposing it.

In data-driven approaches, a machine learning model learns from training data to generate data from our target distribution, removing the need for a manually designed simulator. While some effort is still required to select or design a suitable generative model, this is a much simpler task than designing a simulator, and generative models are less domain-specific. This allows us to create a single framework for generating synthetic data, which is capable of mimicking various datasets. The main drawback of using a data-driven approach is that if care is not taken, the model may learn to 'memorize' the training data and can risk exposing the private data.

Currently, the idea of personally identifiable information (PII) is commonly used in legal frameworks addressing the sharing of private information in fields such as health care and finance [12]. The idea behind these frameworks is to designate a set of features (such as addresses, timestamps, ID numbers, first and last name, etc.), which could be used to identify an individual associated with a data record, and prevent the sharing of any data with these features, or other features which could lead to the re-identification of individuals. While intuitive, it has been shown that this approach can

be problematic, as it can be challenging to determine with certainty what information can be used to identify individuals. In fact, there are many well-documented instances of publicly released datasets that were later used to re-identify participants, including a popular Netflix dataset [109] and data from the human genome project [12, 95]. In general, when trying to release data without releasing private information, there is a trade-off between how well privacy is preserved and the utility of the released data, as methods to preserve privacy tend to remove information from the original data. Another issue with PII-based definitions of privacy is that, in addition to not always protecting privacy, they can also be overly strict and do not allow the user to make trade-offs between privacy and utility [12].

k-anonymity and (ϵ, δ)-differential privacy (DP) are two more recent definitions of privacy, which both have parameters indicating the degree of privacy [12]. The idea behind k-anonymity [144] is to ensure that for every record in a dataset, there are at least k indistinguishable copies. This is accomplished through a combination of removing data fields with unique information (such as phone numbers or ID numbers) and to achieve k-anonymity, two primary techniques are used. *Suppression*, which involves removing some fields, particularly those with unique information such as phone numbers or ID numbers, and *generalization*, which involves aggregating values into ranges. For example, an age field may be transformed into an age range field, where a 23-year-old is encoded as $age \in [20, 25)$, as opposed to $age = 23$. DP [36] instead relies primarily on randomization to preserve privacy. DP has two constants, ϵ, δ, where ϵ controls the strength of the privacy guarantee and δ is the probability of the privacy guarantee being violated. Essentially, DP works by adding random noise to the private data each time it is accessed.

The idea of creating synthetic datasets as a way to increase the accessibility of banking transaction data is not new. In fact, throughout the past decade, there have been various attempts at creating this type of synthetic data, such as BankSim [99], PaySim [98] and AMLSim [176, 156]. While each of these models are tailored to unique goals, they share many of the same underlying limitations.

BankSim and PaySim were developed by the same research group, and thus share many commonalities. Both simulators rely on agent based simulation, which models a set of entities that produce transactions. For instance, in BankSim there are three

types of entities – customers, merchants and fraudsters [99] – which interact throughout the simulation, according to different behavior profiles. During the simulation, as these entities interact they produce transaction records, similar to the type of transaction records stored by banks. PaySim uses a similar approach, but aims to model mobile payment data, which consists of records of transactions between individuals, as opposed to customers interacting with merchants [99].

In AMLSim, agents represent bank accounts, and agents interact by transferring money to one another. Within this framework, there are a small number of agents which aim to conduct fraudulent transactions. The behaviour of these fraudulent agents was designed in consultation with experts [176]. AMLSim uses a graph representation, which represents agents and their interactions as nodes and edges respectively. However, the transactions data is generated using PaySim, and thus has some of the same limitations.

One of the main limitations of these approaches is that they all rely on Markov models to encode transaction probabilities. This means, if we are generating a sequence of transactions for a customer, each transaction only depends on the previous transaction and the customers profile, and is independent of other prior transactions in the sequence. However, this assumption is clearly a limitation when modeling banking sequences, as non-adjacent transactions can be related. Another limitation is that if we want to produce data with realistic fraud patterns, then we must detail these patterns when designing the simulator, as was done by AMLSim [176]. This may introduce added costs to the simulator design. Further, fraud patterns may change over time, and therefore the simulator would need to be manually updated in order to emulate these patterns. Because of these limitation, we are interested in creating a synthetic data generation framework based on deep generative models. While a generative model may also have the be updated over time, these updates can be done automatically, by retraining on more recent data.

1.2 Generative model use cases

The idea of generative modeling is ubiquitous in many areas of modern machine learning. As we briefly introduced earlier, a generative model (GM) is a probabilistic model which describes a process that produces observed data. In addition to sample

generation, which is the focus of this thesis, GMs can also be used for a variety of other tasks. This section provides an overview of the main use cases for GMs.

1.2.1 Sample generation

Arguably, the most straightforward use of a generative model is to generate new samples from a target distribution. State-of-the-art generative models have produced impressive synthetic samples in many difficult domains, including images [69], text [21], speech [130], and many more. Related to the task of *unconditional generation* (sampling $p(x)$) is the idea of *conditional generation*, where the synthetic data is generated based on conditioning variables. Many other tasks fall under the conditional generation framework, which essentially involves sampling the conditional distribution $p(x|conditions)$. Next, we discuss some other interesting examples of conditional generation.

One such example is text-to-image synthesis [136, 187], where images are generated conditioned on a text description of the image. In this application, an auxiliary model is used to map the text descriptions to fixed-size vectors, which are then used as conditioning variables for the model generating the image. In a similar vein, text-to-speech synthesis [119, 169] generates audio signals conditioned on an embedding of the input text, with the goal of the audio signals replicating a human reading the text. Conditional generation is also used in the opposite direction – for example *image captioning* [68, 173, 182], where the goal is to generate a text description of an image. In this application, the model generating the text is conditioned on a representation of the target image. There are actually many other uses for models which conditionally generate text based on other inputs. Two common examples are translation (text in the target language is generated conditioned on input text from a different source language) and summarization (a short amount of text is generated conditioned on a long amount) [96]. Denoising and imputation are two more related use cases for sampling GMs. Denoising models are designed to take as input data (often images) with some imperfections (noise) and output a new version of the input without the imperfections [172]. Imputation is more commonly used with tabular or sequential data, and its goal is to fill in missing data with generated data [43].

Another common use for data sampled from a GM is *data augmentation*. Data augmentation is a set of machine learning techniques for increasing the amount of

available training data. This is particularly important in modern deep learning applications – such as classification of medical images [148] – which often require massive amounts of training data [50, 82]. Not all data augmentation relies on GMs, and many methods create new samples by applying simple transformations – such as rotations, translations, cropping, and adding noise [148] – to samples from the original dataset. There are various ways to do data augmentation with GMs. Adversarial data programming [121] employs a generative adversarial network (GAN) model to generate new unlabeled samples and then a set of informative but imperfect labeling functions to generate noisy labels for the samples. A more common approach is to use a conditional GAN model [105, 117], which can generate samples conditioned on the labels. This approach, taken in works such as [34] and [44], has the benefit of allowing the user to control the label distribution.

1.2.2 Evaluating Data Likelihood

Some GMs allow us to evaluate the likelihood of samples according to the model. That is, if we have an approximate model $\tilde{p}_\theta(x)$ of a real probability distribution $p^*(x)$, and samples $x_1, x_2, ..., x_n$ from the sample space of $p^*(x)$, we can compute the likelihood of these samples occurring under $\tilde{p}_\theta(x)$. The main use cases that require evaluating the likelihood of GMs are anomaly detection and making predictions.

Anomaly detection is an unsupervised learning technique for detecting atypical data points [160]. In theory, using GMs for anomaly detection is straightforward: assuming we have a generative model $\tilde{p}_\theta(x)$ which we can evaluate and that aims to approximate a true distribution $p^*(x)$ then by evaluating $\tilde{p}_\theta(x)$ for a given sample x, we get an approximation of the likelihood of the sample occurring under $p^*(x)$. This likelihood score can then be used for anomaly detection, as lower likelihood scores indicate more anomalous data [14]. While this approach seems intuitive, more recent work shows that in practice, it can be problematic [175]. The problem is that while $\tilde{p}(x) \approx p^*(x)$ averaged over the state space of the variable x, there are often large regions of the state space where these values differ significantly.

Generative models can also be used for supervised machine learning. In this context, the generative model aims to learn an approximation of the joint distribution $\tilde{p}(x, y)$, where x are the features, and y is the label. To use a GM for this type of predictive task, the GM must be implemented in a way that allows us to evaluate

the conditional distribution $\tilde{p}(y|x)$. One benefit of using GMs for supervised ML is that when faced with limited or noisy training data, GMs may outperform traditional discriminative models at predictive tasks [110, 172]. Some common machine learning classification models based on GMs are Naive Bayes [139] Linear Discriminate Analysis [46] and Quadratic Discriminate Analysis [158].

1.2.3 Forecasting

The final task which we will mention is *forecasting*, or predicting future steps in sequential data. Forecasting is a common use of GMs in applications which deal with sequential data, such as predicting future trends in traffic and electricity usage [88, 92, 40] and influenza rates [179]. This is particularly common in autoregressive models, and is closely connected to how they generate new sequential data samples. For a sequence of n elements $s = (x_1, ..., x_n)$, an autoregressive models factors the sequence probability as

$$p(s) = \prod_{i=1..n} p(x_i|x_1, ..., x_{i-1}), \tag{1.1}$$

meaning the predicted distribution over the i^{th} element depends on all previous sequence elements. To forecast a single step in the future, we can simply evaluate the model on the current sequence to get a probability distribution over the next element. However, forecasting further ahead is more complex. When we try to predict the second next element in the sequence, we do not have the previous sequence element, only a distribution over what it could be. One approach that has seen some success is to use a model designed to make predictions multiple steps ahead, instead of relying on a strict autoregressive approach [92, 179].

1.2.4 Dimensionality Reduction

Certain GMs, such as variational autoencoders (VAEs), can be used for *dimensionality reduction*. VAEs are latent variable models, which operate on the assumption that high-dimensional data are produced by a random process operating on a small set of unobserved *latent variables* [74]. With VAEs, it is possible to estimate the latent variables, and then use the latent variables in place of the original data in downstream tasks. Originally, the main motivation for using VAEs for dimensionality reduction

was their ability to learn complex non-linear relationships [74], unlike some traditional dimensionality reduction approaches, such as principal component analysis. More recent work has used GM based dimensionality reduction to learn more interpretable representations [2, 32], as well as fairer representations, which can conceal information about sensitive attributes such as race or gender [28].

In summary, there are many possible use cases for GMs, and improving the state-of-the-art GMs in any domain can unlock improved results on a variety of tasks. Recent work throughout the past decade has proposed many novel types of deep generative models (DGMs), based on neural networks and other multi-layer architectures. As these models have proven very successful at generating many types of complex data, they feature prominently in our work. In the following section, we give an overview of DGM sub-types, focusing particularly on the DGMs used throughout this thesis.

1.3 Deep Generative Models

1.3.1 GANs

Generative adversarial networks (GANs) – originally proposed by Goodfellow et al. [51] – is a powerful generative modeling framework based on a game-theoretic approach. The basic idea behind GANs is that there are two parameterized neural networks, a *generator* network $G_\theta(\cdot)$ which aims to transform samples from a simple noise distribution $p(z)$ to the target distribution, and a *discriminator* network $D_\phi(\cdot)$ which attempts to distinguish real samples from samples generated by the generator. The training of a GAN model can be framed as a competition between the two networks, where the goal of the discriminator is to successfully determine which samples are real and which are synthetic. The generator's goal is to generate realistic samples that fool the discriminator. This idea was originally implemented by alternately training the generator and discriminator with the objective

$$\min_{G_\theta} \max_{D_\phi} V(G_\theta, D_\phi) = \mathop{\mathbb{E}}_{x \sim p^*(x)} [\log D_\phi(x)] + \mathop{\mathbb{E}}_{z \sim p(z)} [\log(1 - D_\phi(G_\theta(x)))]. \tag{1.2}$$

The first term in this expression is the expected log probability the discriminator assigns to real data points coming from the real distribution. The second term is the expected log probability the discriminator assigns to synthetic samples from the generator network. It has been shown that when training a GAN with the objective in Equation 1.2, the discriminators objective is equivalent to minimizing a lower bound on the Jensen Shannon divergence between the real distribution and the distribution produced by the generator. This led to the proposal of a plethora of different GAN models using objectives based on various metrics for comparing the real and generative distributions. Among these, Wasserstein GAN [5, 58] – which is based on the Wasserstein or 'earth movers' distance – has been particularly impactful due to its relatively stable training and high-quality results in many domains.

Another significant GAN model is the conditional GAN [105] and related variants such as [117]. The motivation behind these models is to gain greater control over the data generation, by training the generator to generate samples conditioned on other information. The original conditional GAN (cGAN), proposed in [105], adopts a straightforward approach to conditional generation. When generating data, the generator receives a randomly generated conditioning information y in addition to the random noise sample $z \sim p(z)$ and uses both to construct a new sample $\tilde{x}$. The conditioning information may be simple, such as a single label, or much more complex, such as the text-to-image synthesis application discussed in Section 1.2. The cGAN discriminator is modified so that it also takes conditioning information as input in addition to the input sample. cGAN can learn conditional generation under this framework because there should be a strong relationship between the conditioning information and samples in real data. The generator is forced to learn this relationship, otherwise the discriminator can use inconsistencies between the conditioning information and sample to identify the synthetic samples.

A related approach is auxiliary classifier GANs (ACGANs) [117], which also aim to perform conditional generation. The main difference between cGAN and ACGAN is that in ACGAN, the discriminator does not take the conditioning information as input; instead, the discriminator aims to predict the conditioning information from the samples, in addition to predicting which distribution they are from. The generator architecture is the same as in cGAN, however the training objective is modified slightly to include a term that incentives the generator to create samples such that the discriminator can identify their class, but not the distribution they are from. This

modified generator objective is necessary for ACGAN to learn the relationship between conditioning information and samples, and empirical evidence shows it may also help with stabilizing training of the model [117].

1.3.2 VAEs

Variational autoencoders (VAEs) – like GANs – are created by combining multiple neural networks (usually 2, but some models use more). The VAE contains a network called the *decoder*, and similar to the generator in GANs, the decoder learns to map samples from a base distribution $p(z)$ (often a multivariate isotropic Gaussian) to samples from a target distribution. However, this is where the similarities between GANs and VAEs end, and the role of the *encoder* network in VAEs is quite different from the discriminator in GANs.

The main modeling assumption behind VAEs, is that while the distribution of the data we observe, $p^*(x)$, may be very complex, it can be modeled as a set of latent features from a simple distribution $p(z)$, which undergo a complex generative process $g_\theta(\cdot)$ to produce the observed data. In VAEs, the generative process $g_\theta(\cdot)$ is modeled by the decoder network, which is a neural network with parameters θ. Concretely, the decoder does this modeling by learning to map samples from the latent space to distributions in the original data space. That is, for $g_\theta(z) = \tilde{p}(x|z)$. In practice, this is often implemented by treating the outputs of g_θ for each feature as means of independent Gaussian distributions, with some constant variance. The learning of this generative process requires the encoder portion of the VAE, which essentially learns the inverse of the generative process. The goal of the encoder network – also known as the inference model – is to infer the latent factors z which underlie observed data x. Specifically, the encoder learns to map samples from the data distribution $p^*(x)$ to a distribution in the latent space $q_\phi(z|x)$, which gives the likelihood of the latent factors given the observed data. In practice, a common way to implement $q_\phi(z|x)$ is as follows; for a D dimensional latent space, the encoder network outputs $2D$ values, representing means and variances of independent Gaussian's for each latent feature.

The overall training of a VAE follows this general recipe. First, draw a sample x from the training data, and pass it through the encoder, to obtain a distribution $q_\phi(z|x)$ over the latent factors which are likely underlying x. Next, sample latent features z from this distribution $z \sim q_\phi(z|x)$, and pass z through the decoder to

obtain a probability distribution over the sample space $\tilde{p}_\theta(x|z) = g_\theta(z)$. In the final step we compute the loss, which for the original VAE [74] is computed as

$$Loss = \log \tilde{p}_\theta(x|z) - D_{KL}(q_\phi(z|x)||p(z)) \tag{1.3}$$

and use this loss value to update the weights of both the encoder and decoder networks with a gradient based optimizer, such as gradient descent. The loss function is usually designed to encourage two objectives. The first term in the loss, $\log \tilde{p}_\theta(x|z)$, incentives the VAE to output distributions with high probability of producing the original sample x. The second term, $D_{KL}(q_\phi(z|x)||p(z))$, encourages to encoder to produce distributions in latent space that are close to the base distribution. The second term may seem spurious, but it encourages the learned latent space to approximately follow $p(z)$, which is necessary if we wish to generate new samples from our trained models. A more recent model called β-VAE introduced a modified loss function, which uses a constant to control the weighting of these two terms [61].

Additionally, as with other DGMs, several other extensions and modifications have been made to the original variational auto-encoder framework. One significant improvement – called Wasserstein auto-encoders [161] – has been shown to produce significantly higher resolution images than previous VAEs. The main difference between this method and a basic VAE, is the use of the Wasserstein distance as a metric for judging sample quality, as opposed to the KL divergence used in the original model. An interesting note here is that both variational auto-encoders and GANs have been improved by using the Wasserstein distance as a metric [3, 5, 161].

Research into VAEs has seen significant attention focused on relaxing the assumption that latent variables are independently normally distributed. This assumption is made in the original work because it allows for straightforward computation of $D_{KL}(q_\phi(z|x)||p(z))$ in the loss and still produces good results on many datasets. One technique for relaxing that assumption is to replace the normal distribution with other distributions with useful properties, such as simple mixture [163] and hierarchical models [75]. Another set of methods is based on another DGM called *normalizing flows* [137] (briefly discussed in Section 1.3.4), which allow users to define complex probability models by combining a simple based distribution with a chain of parameterized simple invertible and differentiable transformations. In the novel VAEs that use normalizing flows to model latent variables [73, 154, 164, 165], the VAE's encoder

network outputs parameters of the normalizing flow, as opposed to parameters of a normal distribution.

1.3.3 Transformers

Transformers are a DGM architecture, first proposed in 2017 [171], which has had a large impact on sequence modeling in general, and language modeling in particular. There are many different models which have been proposed based on the transformer design, so it can be hard to define exactly what constitutes a transformer. Originally transformers were proposed as a sequence model for tasks involving mapping one sequence to another (such as translation). Their main novelty was that they could process sequences without recursion, instead relying on a *multi-head attention* (MHA) mechanism combined with a *positional encoding* (PE) scheme to model sequences. First, we will briefly introduce the idea of *attention* in the context of neural networks and mention some early models which inspired the transformer's design. Then we will return to ideas of MHA and PE and discuss how these are used to replace the need for recurrence in transformers.

In the following discussion, we will consider the task of translating sentences from English to French. This is a typical sequence transduction task, meaning a task where an input sequence is mapped to a target sequence. Sequence transduction is a very generic task – in addition to translation, it also includes summarization and question answering – and it was the goal of early transformers. Early work on neural machine translation (NMT) [26, 155] – which is the name use for translation task which rely solely on neural networks – used *encoder-decoder* models, with recurrent neural networks (RNNs) used for both the encoder and decoder networks. In this approach, the recurrent encoder learns to map English sentences to fixed-length vectors, and the decoder learns to map the fixed-length encodings to target French sentences.

Soon after being proposed, the basic encoder-decoder approach to NMT was greatly improved by the introduction of an *attention mechanism* in [9] and [100]. In [9], the encoder network is modified to output an encoding for each element in the input sequence. When generating the target sentence, before each word is generated, an *alignment score* is computed for each element in the input sequence, indicating how relevant it is to the word which is currently being generated. These scores are then normalized to sum to 1, and then a weighted sum of the encoded sequence is provided

to the decoder, replacing the fixed-length vector used in the basic encoder-decoder models. This weighted sum is also a fixed length vector (the dimension is the same as the encoding dimension used by the encoder); however, the difference is that with the attention model, the fixed-length vector provided to the decoder is different for each word being generated. While the term *attention* is actually not used in [9], the normalized weights from the alignment scores are a form of attention scores. This is highlighted in [100], where the authors compare a novel type of attention they term *local attention* to a version similar to that used in [9], which they call *global attention*. In their local attention formulation, the main novelty is that only a fixed number of positions in the input sequence are used for computing the attention vectors at each step. These positions are either chosen based on the current position in the output sequence, or by an additional predictive model [100]. Since these early works on attention, many different variants have been proposed (for a comprehensive review of attention mechanisms, see [116]). For sequential data, the main idea that ties together the many variants of attention is computing a set of weights (which sum to 1) over elements of the sequence, and then using the weights to compute a weighted sum of the sequence.

The original transformer model [171] was an encoder-decoder model based on feed-forward neural networks, unlike the encoder-decoder models above, which are based on RNNs. The encoder and decoder are both deep neural networks and are composed of stacks of N encoder layers and decoder layers respectively (Note: in [171] $N = 6$). In addition to the encoder and decoder stacks, the transformer also has additional layers for embedding the input and output data, which also perform the PE.

The embedding layer performs two steps. The first is to map the input vectors – in [171] these are one-hot encodings of words – down to d_{model}, which is the dimensionality used for all intermediate representations. After embedding input vectors, a PE vector is added to each embedded vector, which encodes the vectors position in the input sequence. This is necessary because the transformer does not use recurrence and thus needs another way to learn the relevant positional information. While there are many possible options for creating PE vectors, a standard choice for d_{model}-dimensional PE vectors is for the i^{th} dimension, corresponding to input position t, to be $\sin(t/10000^{i/d_{model}})$ if i is even, and $\cos(t/10000^{i/d_{model}})$ otherwise [171]. The output from the embedding layer then feeds into the encoder stack.

The encoder network maps an input sequence of length L to a sequence of encodings also length L. Encoder layers are made of two sub-networks; the first implements a multi-head self-attention mechanism, and the second is a basic fully connected feed-forward neural network. In addition to these sub-networks, encoder layers also use residual connections [60] and layer normalization [7] between sub-networks, which helps with training deep models. In practice, this means they add the input to each sub-network to its output and perform a normalization step afterward.

As we mentioned above, the first sub-network implements a mechanism based on multi-head self-attention. Throughout the transformer, a novel form of attention – called *scaled dot product attention* – is used in all attention mechanisms. Scaled dot product attention is based on a system of queries, keys and values, and resembles another variant known as *dot product attention* [100]. To implement scaled dot product attention, we need to introduce 3 matrices with trainable weights W^Q, W^K, W^V, which are used to compute the queries, keys and values respectively. To show how these work, consider an example where we wish to translate the sentence "Hello world" to French. After passing through the embedding layer, this sentence will be represented as a sequence with length 2, because there are two words in the input, and each sequence element has dimensionality d_{model}. The next step is to transform the representation of each element in this sequence into a set of queries, keys and values, which is done by multiplying the vector representation of each element with the corresponding weight matrix, to produce vector queries, keys and values. As we have already discussed, the key features of attention mechanisms are to get a weighted sum over sequence elements. Here, the values which we want a weighted sum over are the values produced from the multiplication with the weight matrix W^V, and the weights are a function of the queries and keys. Specifically, suppose we wish to encode the first sequence element. In that case, we need to compute a set of 2 attention weights. The first indicates how relevant the first element is to its encoding, and the second indicates how relevant the second element is for encoding the first. In this simple example, we would expect the first weight to be much higher than the second, and in general, when encoding a single element in a length L sequence, L attention weights are needed.

To obtain attention weights for a given query vector q_i representing the i^{th} element in the sequence, we first take the dot product of q_i and each key vector to obtain a set of raw scores. The raw scores are then normalized by first dividing by the dimensionality

of the keys d_k and then performing a softmax operation to ensure the weights are non-negative and sum to 1. In practice, if Q, K, V are matrices representing the queries, keys and values for a sequence, with each row containing the values for a single element, then scaled dot product attention can be computed as

$$Attn(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.4}$$

This example of scaled dot product attention also serves as an example of *self-attention*. The idea of self-attention is that our attention mechanism is directed at data we are trying to encode instead of a different part of the computation. In self-attention, keys, queries and values are all computed from the same input embedding. In the example, we are using self-attention when we are trying to encode "Hello" and can attend to either the current encoding of "Hello" or "World". An example of attention that is not self-attention would be in the decoder layers, where there is a mechanism that attends to elements in the input sequence when words in the French translation are predicted.

When designing the original transformer, [171] found that instead of simply making the key, query and value vectors have dimensionality d_{model}, it was beneficial to use h different attention functions, called *heads*, with dimensionality $\lfloor \frac{d_{model}}{h} \rfloor$. This is implemented by linearly projecting the matrices Q, K, V into h different sub-spaces, using trainable weight matrices. Specifically, the attention function for the i^{th} head is $Attn(QW_i^Q, KW_i^K, VW_i^V)$, where $Attn(\cdot)$ is Equation 1.4, and W_i^Q, W_i^K, W_i^V are the projection matrices.

The decoder stack takes two inputs, the encoding of the input sentence, which is output by the encoder stack, and an encoding of the target output sentence. Because each element output by the transformer is conditioned on all previous elements in the output sequence, during training, we use the target sentence as input so that the transformer can predict the entire sequence in parallel, using the target data as previous elements for conditioning. This technique, known as *teacher forcing* [35], is commonly used when training neural networks on sequential data. When using the transformer to translate novel sentences, the decoder stack is called once for each word generated in the output. In the translation example, the encoder stack would output some sequence $[e_1, e_2]$ encoding of the sentence "Hello world". The first time we call the decoder, the inputs would be $[e_1, e_2]$ and an encoded symbol signifying the start

of a sentence, and the network should output "Bonjour", The second time we call the decoder, the inputs are $[e_1, e_2]$ and an encoding of "⟨start⟩ Bonjour", and the output should be "Bonjour le", and so on.

The decoder layers of the transformer are similar to the encoder layers but contain a third sub-network and a modified version of the self-attention layer. The additional sub-network is inserted between the self-attention and the feed-forward sub-networks, and it performs MHA on the encoding produced by the encoder stack. In the decoder layers, self-attention refers to attention over the current output sequence. The reason the self-attention needs to be modified is so that during training, the network cannot "cheat" and attend to the whole input sequence. Instead, the modified self-attention uses a masking approach to ensure that when processing the i^{th} element of the output sequence, only elements in positions less than i can be attended to.

One important modification to the original transformer design is known as the *transformer-decoder* (TD), or decoder-only transformer [96]. As the name suggests, the TD architecture is a transformer model built solely using decoder layers. The idea behind the TD model is that we can convert a problem of the form input = "Hello world", target output = "Bonjour le monde", to a single input sentence of the form "Hello world ⟨translate⟩ Bonjour le monde". Then, using sentences with this representation, the TD is simply trained with the same objective as the basic transformer: given the current elements, predict the next element. One significant benefit of this approach is by getting rid of the encoder stack, the number of parameters in the network is nearly cut in half. Another benefit is that this model can easily perform unconditional generation – meaning generating new sequences from the training data distribution – and conditional generation tasks such as translation and question answering.

While originally designed for modeling language data, transformers have also been applied to modeling other types of time series data, including influenza prevalence [179], as well as electricity usage and traffic [88, 92, 40]. The only changes needed to the original transformer model to handle these other data types are modifications to the loss function and embedding layers.

1.3.4 Other DGMs

Autoregressive (AR) DGMs are a class of DGMs that model data with a sequential representation. The transformer model discussed above is an example of an AR model; here, we highlight some other interesting and important AR models. AR models have shown great success at modeling data that is naturally represented sequentially, such as text [21] and audio [169]. Additionally, AR models have achieved strong results on data that are not inherently sequential. For example, image AR models such as PixelRNN [168] and PixelCNN [120, 143] model image data by first defining an ordering of the pixels (such as top to bottom, left to right), and generating each pixel sequentially, conditioned on all the previously generated pixels. And for basic tabular data, AR models such as NADE [80] and RNADE [166] define an ordering over features, and generate samples by conditioning the generation of each feature on all previous features. Extensions to NADE and RNADE allow these models to simulate being simultaneously trained on all possible feature orderings [48, 167].

A relatively new class of DGMs which have recently received much attention are *diffusion models*. The original idea behind these models was to learn both a forward process which maps data from the target data distribution to a simple noise distribution, and a reverse process which inverts the mapping [151]. Experimentally evidence suggests that in their current state, diffusion models are generally better image generators than GANs, particularly when image diversity is important [30]. This is further supported by the fact that current state-of-the-art image generators, such as DALLE-2 [134] and Stable Diffusion [141], are based on diffusion models. Recent work has also shown that models combining diffusion and transformers are able to outperform other generative models at certain text generation tasks [89]. While the work on diffusion models, particularly for sequential data, is still in its infancy, results to date indicate it is a powerful paradigm for generative models.

The final class of DGMs we will discuss in this section, is *normalizing flows* (NFs). NFs are a recently developed type of DGM [137] that models complex probability distributions by combining a simple base distribution $p(z)$ with a chain of parameterized functions which are invertible and differentiable. Learning in NFs is straightforward; the parameters of the function chain are optimized with a gradient-based optimization method (such as [71]) to maximize the likelihood of the observed data. Because all of the functions in the NF chain are invertible, we can perform an exact mapping between

the base distribution and the approximate data distribution. However, one drawback of the invertibility condition is that it requires $p(z)$ have the same dimensionality as the data, unlike other DGMs, which can use base distributions with arbitrary dimensionality. In practice, the functions must be efficiently invertible, which led early works to focus on very simple parameterized functions [137]. More recently, flow-based models have been substantially improved and now can provide high-quality synthetic samples in many domains [72, 130].

1.4 Evolutionary Computing

Evolutionary computing is a broad area of study in computer science, which studies algorithms inspired by, and related to, natural evolution. Generally, evolutionary computing is mainly used for one of two purposes; running simulations to study the dynamics of evolution, or using evolution to solve application problems involving optimization or learning. Many early thinkers in the field of evolutionary computing, viewed natural evolution as a powerful optimization algorithm, which had produced organisms optimally suited for a vast array of environments [37, 132]. Despite the early hopes, as other techniques for optimization were refined, and new ones created, evolutionary computing was overshadowed by methods which tended to perform better in empirical studies, and was criticised for lacking empirical foundations by some [102].

Recently, some researchers have begun to argue that the true power of natural evolution is not merely as an optimizer of performance in a set environment, but that the power lies in its ability to jointly maximize the diversity of organisms, as well as the fitness of organisms to their environment [86, 132]. The idea of evolution favoring diverse populations, as opposed to populations where every individual is maximally fit, may seem unintuitive at first. However, one major limitation of purely maximizing fitness is that – both in natural evolution and evolutionary computing applications – we cannot be sure that the fittest solution from the past will be the fittest in the future. If the goal is to find a solution which will work well in the future, and all the data we have collected has been from the past, then we must necessarily have some uncertainty about which solution will be best in the future. Because of this uncertainty about what the future may hold, natural evolution succeeds not by placing all resources towards what seems best based on the current environment, but

by allocating resources to maximize diversity, subject to constraints of the current environment, and information about what worked best in the past. In Chapter 4 we propose a novel evolutionary algorithm in the context of classification problems, and in Chapter 5 we use a variant of this approach to improve our synthetic data generation framework. The remainder of this section provides a broad overview evolutionary algorithms, as well as details on the specific evolutionary computing techniques we employ in this thesis.

1.4.1 Evolutionary Algorithms

The general schema for an evolution algorithm (EA) involves a *population* of *individuals*, each of which are considered *candidate solutions* to a target problem. The population is *initialized* to contain an initial set of individuals, and throughout the course of the EA new candidate solutions are created by applying *variation operators* to create new individuals from the population. During each epoch of evolution, there is a *selection* step, which selects a subset of the individuals to remain in the population, and eliminates the remaining candidates. This general framework has been used to create a wide variety of specific evolutionary algorithms for solving specific problems. Next, we provide some addition details on these core EA components.

Individual

In the context of an EA, an individual represents a potential solution to the problem we are attempting to solve. The representation of an individual depends on the application, and there are many potential choices. For example, in Chapter 4, the individuals in our EA are *linear genetic programs*, which are used as binary classifiers. Whereas in Chapter 5, the individuals are parameter dictionaries, which encode hyperparameters for a neural network model. The encoding of individuals also effects how we apply variation to create new individuals, and how we *evaluate* individuals.

Evaluation of individuals is also highly dependent on the problem. To evaluate individuals, we use a *fitness function*, which maps individuals to a *fitness score*, which is used during selection to determine which individuals remain in the population.

Population

A population refers to a set of candidate solutions maintained throughout evolution. Traditionally populations are fixed-sized, and contain an unordered collection of individuals. However, some more recent methods, including the *quality diversity* algorithms described Section 1.4.2, use more complex population structures. Populations are often initialized by creating a random set of candidate solutions; however, some applications may use problem-specific initialization schemes [37].

Variation

Variation operators are applied to existing candidate solution, to create new candidate solutions. There are two types of variation operators used in EAs. The first is *mutation*, which takes as input a single candidate solution, and outputs a modified version of the input solution. The other is *crossover*, which takes two individuals as input, and outputs a new candidate solutions combining the two input solutions.

Selection

After creating a new batch of candidate solutions by applying variation to an existing population of solutions, *selection* determines which solutions are kept in the population, and which are eliminated. The simplest from of selection is *elitist selection*, which combines the existing population and new batch of solutions, and selects the k best solutions, where k is the population size. There are many variants of *non-elitist selection*, which are designed to encourage greater diversity within a population.

1.4.2 Quality Diversity

Traditionally, the goal of EAs is to produce a single final solution, with the maximal fitness score. Quality Diversity (QD) algorithms, in contrast, aim to produce a collection of solutions which are both highly fit and qualitatively diverse. There are many benefits of this explicit focus on diversity. For example, optimization problems often have search spaces filled with many local optima, and it has been shown that focusing

on population diversification helps prevent evolution from getting stuck in these sub-optimal parts of the solution space. Another benefit is that sometimes there are trade-offs which can make determining the *best* solution ambiguous. QD algorithms can also be used to visual trade-offs between different types of high-performing solutions, such as neural networks with different levels of modularity and connectedness [108].

There are two main QD algorithms, Novelty Search with Local Competition (NSLC) [85] and MAP-Elites [108]. NSLC employs a variable sized *archive* structure as its population structure. New candidate solutions are added to the archive if they are sufficiently novel. If a new candidate is not sufficiently novel, it may displace an existing similar solution, if the new solution has higher fitness. MAP-Elites uses a fixed-sized, structured population, which is typically structured as a two-dimensional grid of cells. MAP-Elites uses a mapping function, which assigns candidate solutions to specific cells, based on their *behavior*, so that similar solutions tend to be mapped to the same cells. When a new candidate solution is created, it competes with the existing solution in its corresponding cell based on fitness values, and the winner is stored in the cell. If a new solution is assigned to an empty cell, it is always stored.

1.5 Sequential Data

To conclude this chapter, we give an overview of different types of sequential data and highlight the unique challenges in generating sequences of transactions.

Non-Temporal Sequences

Non-Temporal sequences are sequences without time-based features. Typically, they are represented as an ordered list of discrete symbols. One area where non-temporal sequences are common is in textual data, such as sentences, paragraphs, or articles. These types of data are naturally modeled as sequences of words and symbols, or at a lower level, as sequences of characters [54]. Biology is another domain that deals with non-temporal data. DNA strands are often represented as sequences of base pairs [70], and proteins can be represented as sequences of amino acids [138, 180].

Regular Time Series

The term *time series* generally refers to a sequence of measurements that involve temporal data. Commonly, time series data refers to sequences of measurements made at regular intervals, such as hourly electrical usage [88] or daily stock prices [149]. We use the term *regular time series* to specify time series data with regular measurement intervals. In the context of synthetic data generation, regular time series data are more straightforward to model than time series data with irregular intervals because there is no need to model the time between measurements.

Transaction Sequences

The final type of sequence we will mention here are *transaction sequences*, a type of irregular time series in which sequence elements correspond to real-world interactions. Each transaction in a transaction sequence contains information about *when* the transaction occurred, in addition to *what* the transaction is. The information about what the transaction is may be a simple categorical code, or it may be more complex. Transaction sequences typically have some dependency between the temporal and non-temporal features, with certain types of transactions occurring more or less often depending on temporal features, such as the time of day or the day of the week.

amount	flag	description	type	timestamp
45.66	Utility Bill	Energy	Debit	2017-04-24 11:00:00
2842.27	Income	Monthly	Credit	2017-04-24 17:36:55
167.25	Credit Card	Credit Card Payment	Debit	2017-04-25 16:00:00
40.58	Utility Bill	Water Bill	Debit	2017-04-27 16:00:00
43.16	Utility Bill	Energy	Debit	2017-05-23 17:00:00
2842.27	Income	Monthly	Credit	2017-05-23 18:40:45
775.20	Credit Card	Credit Card Payment	Debit	2017-05-25 09:00:00

Figure 1.1: Example of a transaction sequence (banking transactions). This example shows a sequence of 7 transactions, which were performed on a bank account. The first column (*amount*) shows the dollar value of the transaction. The next two columns (*flag, description*) contain categorical codes, which give information about what type of transaction is taking place. The *type* column is a binary variable, indicating if money is being transferred to the account (*Credit*) or from the account (*Debit*). The *timestamp* indicates when the transaction occurred.

Throughout this thesis, we focus on modeling transaction sequences associated with customer bank accounts, such as the transaction sequence shown in Figure 1.1. In this data, each transaction encodes details of a transfer of funds to or from a customers account. These transactions record the amount transferred, as well as multiple categorical features that give information about the transaction.

1.6 Thesis Contributions and Structure

Throughout this thesis, we make the following contributions:

1. Propose Banksformer (BF), a transformer-based generative model for generating synthetic banking data, and demonstrate it improves over existing GAN models.
2. Develop ensemble and elite-ensemble variants of Banksformer, which improve on our initial Banksformer results
3. Develop DP-BF as a way to produce lower quality synthetic data with provable privacy guarantees, as a step towards creating high quality private synthetic data.
4. Develop a suite of metrics to characterize the quality of synthetic banking data
5. Develop a novel evolutionary algorithm for creating diverse classifier ensembles
6. Adapt our evolutionary algorithm to produce populations of BF models, which improved results on some metrics, and provides an efficient way to do parameter tuning in future work.

The remainder of this thesis is structured as follows. In Chapter 2 we propose *Banksformer*, a transformer-based generative model for creating synthetic banking data. Chapter 3 explores methods for improving Banksformer, both by generating higher quality data and adding formal privacy guarantees. Chapter 4 temporarily diverges from the main thesis narrative, to develop a method for creating diverse sets of models. In Chapter 5, we apply a modified version of the method developed in Chapter 4, to further improve upon the quality of data we generate. Finally, Chapter 6 concludes this thesis by summarizing and reflecting on the impact of our work, and outlining future research directions.

Chapter 2

Banksformer: A Deep Generative Model for Synthetic Transaction Sequences

In this chapter, we present our initial work on Banksformer. It includes information on the motivation and design of Banksformer, the two banking datasets which we used when developing Banksformer, and a comparison between Banksformer and other state-of-the-art DGMs for sequential data. Results from this chapter highlight Banksformers' ability to capture the appropriate spacing between transactions and date-based patterns.

This work was published and presented at the 2022 ECML PKDD conference, a CORE-A ranked conference in the field of machine learning [114].

2.1 Introduction

Synthetic data are becoming an increasingly important component in machine learning systems. Recent work has demonstrated the ability of deep generative models (DGMs) to produce high-quality synthetic data in domains such as images [69], text [21], and audio [39]. Each of these domains has presented unique challenges, which were addressed by modifying model architectures from previous tasks to be more suited to the target task. Success in these general domains has led to the creation of focused,

domain-specific models. One domain that has received considerable recent interest is financial data.

Financial data is a broad category, however most existing work on DGMs in finance focuses on modeling price sequences for stocks and other financial instruments [87, 157, 178]. Another important type of financial data is transactional data; that is, data that contains sequences of records or transactions recorded at arbitrary intervals. Transactional data is common in finance but also occurs in other domains. For example, both a sequence of purchase records from a credit card and a sequence of entries in electronic health records are transactional. In general, modeling transactional data is more challenging than other time-series data, as we must learn to model the intervals between transactions in addition to the transaction features. This can be particularly challenging in a domain such as banking, where the date and time of a transaction can be strongly related to the transaction type and amount. Further, certain types of dates, such as the weekends or the end of the month, can significantly influence what transactions occur.

Evaluating the quality of synthetic data is a difficult problem without a single clear solution [4, 66, 159]. Ideally, we would like to measure a distance between the real and synthetic data distributions; however, this is not feasible for multidimensional sequence data. A seemingly general approach would be to use the log-likelihood the generative model assigns to validation data. Unfortunately, this approach is known to have issues [159], and also depends on the model being able to assign likelihood scores, which is possible for transformers but not generative adversarial networks (GANs) [51]. Existing work generating financial time series is limited but commonly evaluates the quality of generated data by comparing univariate features distributions [87, 178]. However, these univariate metrics only give a rough picture of the synthetic data quality. These metrics cannot measure how well the synthetic data captures feature interactions and interactions between sequence elements.

The main goal of this work is to produce high-quality synthetic financial transaction sequence datasets, with the same statistical properties as real data upon which they are based. We propose Banksformer, a novel transformer-based DGM designed to model transactional data with date-based patterns. GANs have typically been used as the generative model in previous work generating sequential financial data [87, 157, 178]. To demonstrate the benefits of our approach, we compare BF

against two high-quality GAN models – TimeGAN (TG) [185] and DoppelGANger (DG) [94] – on two datasets of banking transaction sequences.

2.2 Datasets

We used two datasets of banking transactions to compare the quality of synthetic data produced by BF with data produced by TG and DG. In this work, two datasets of banking transactions are used. The first is a set of real banking data from the Czech Republic in the 1990s[1] (*czech*), and the second is a synthetic dataset of transactions from the UK in 2017[2] (*uk*). We chose to include the synthetic dataset because we could not find a second real dataset, and wanted to show the method we built while focusing on the *czech* dataset would generalize.

Both datasets contain transaction records from many different bank accounts, with the *uk* dataset containing 5 000 unique accounts, and the *czech* containing 4 500 accounts. Each transaction contains the dollar value of the transaction, multiple categorical codes that have information about the transaction type, and a timestamp indicating when the transaction occurred. To create a uniform representation between datasets, we concatenate together all categorical codes into a single field called the *tcode* (transaction code). In the *czech* data there are 16 unique tcodes, and the *uk* dataset has 44 (Table 2.1). The timestamp in the *czech* dataset only contains the transaction date, and not the specific time of day. Because of this, we do not use the time of day information in the *uk* dataset and focus only on modeling the transaction dates.

Table 2.1: Dataset Summary. Properties of the *czech* and *uk* data sets.

	Accts	Total Trans	Trans per Acct			Tcodes	Date Range	
	count	count	min	max	mean	count	start	end
czech	4500	1.06×10^6	9	675	235	16	01/01/1993	31/12/1998
uk	5000	10^5	2	50	20	44	01/04/2017	25/05/2017

We are primarily interested in the *czech* dataset, which was initially made available

[1] https://data.world/lpetrocelli/czech-financial-dataset-real-anonymized-transactions

[2] https://pub.towardsai.net/generating-synthetic-sequential-data-using-gans-a1d67a7752ac; this blog post explores using DG to create synthetic data

as part of the Discovery Challenge at the 1999 PKKD conference [126]. This dataset is likely to lead to more meaningful results than the *uk* dataset for three main reasons. First, the *czech* dataset contains real banking data. This is in contrast to the *uk* dataset, which is a synthetic dataset. Second, the *czech* dataset contains over 1 M transactions, making it over ten times larger than the *uk* dataset, which has only 100K transactions. Because the datasets have a similar number of unique accounts, this means there are comparatively fewer transactions per account in the *uk* dataset (Table 2.1). Finally, the *uk* data is also from a much smaller range of dates, containing less than two months' data, whereas the *czech* dataset spans five years.

Transactional banking data often contains date-based patterns, which can be difficult for DGMs to emulate. In the *uk* dataset, the most significant date-based patterns are related to the day of the week. In that dataset, transactions never occur on Sunday. Further, certain types of transactions are related to the day of the week, and happen more or less often on certain days. Because the *uk* dataset spans less than two full months, we do not consider patterns related to the day of the month. In contrast, the *czech* dataset does not contain any apparent relationships involving the day of the week. However, in the *czech* data there are clear patterns related to the day of the month, with certain types of transactions only occurring at the month's end, and others only happening early in the month.

We transform this dataset of transactions into transaction sequences by grouping together transactions by account, and then sorting the transactions for each account by date (and time in the *uk* dataset). In order to create more uniform datasets, we filtered out sequences shorter than a minimum length parameter l_{min} (5 for *uk* and 20 for *czech*), and split sequences longer than l_{max} (20 for *uk* and 80 for *czech*) into multiple contiguous subsequences, so that all sequences used for training and validation have length in the range $[l_{min}, l_{max}]$. In addition to the features present in each transaction, there is also meta-data information associated with each sequence. This meta-data contains the starting account balance, start date of the sequence, and for the *czech* dataset, the customers' age at the start of the sequence. To preprocess the data for the generative models, continuous features are linearly scaled to have 0 mean and variance of 1, and categorical features are encoded with a one-hot encoding. In the generic preprocessing step used by all models, we follow the method of [94] and represent time information by providing the start date as meta-data and including a time delta feature with each transaction that indicates the amount of time that has

passed between transactions. When using BF, we perform a further preprocessing step (detailed in Section 2.5.1) to create additional date-based features which BF requires.

2.3 Methods

2.3.1 Generative adversarial networks (GANs)

GANs [51] are a commonly used generative model, and are capable of generating high-quality synthetic data in many domains [21, 39, 69]. TimeGAN (TG) [185] and DoppelGANger (DG) [94] are two GAN models that have been successful at generating complex multivariate sequence data. Each of these models have unique innovations that allow them to generate high-fidelity synthetic sequences. In TG, an embedding scheme is used so that the generator and discriminator are operating in an embedded space, and a supervised loss based on predicting the next sequence element is used in addition to the standard GAN training objective. In DG, there are many minor innovations, including *batch generation* to better capture long-term dependencies, a conditional generation mechanism to deal with relationships between metadata and sequences, and a custom *auto-normalization* scheme that reduces mode collapse, which is the tendency for GAN models to focus on generating typical samples, which hurts the diversity of data generated.

2.3.2 Transformers

The *transformer* architecture [171] was designed to perform sequence modeling tasks without a recurrence, instead relying on an attention mechanism and positional encoding scheme to model sequence ordering. While originally proposed as a language model [171], transformers have since been applied to modeling many types of sequences [88, 179]. In this work, we use the *transformer-decoder* (TD) [96] variant of the transformer, as this is most appropriate for generating novel sequences. TD is designed as an auto-regressive model that can model probability distributions over sequences.

The main innovations in the transformer and TD architectures are *positional encodings* (PEs) and *multi-head attention* (MHA). Since transformers do not use recurrence, and process all sequence elements simultaneously, the PEs are designed to allow the model to learn ordered sequences by adding a PE vector to the initial embedding. While there are many possible options for creating PE vectors, a standard choice for d-dimensional PE vectors is for the i^{th} dimension, corresponding to input position t, to be $\sin(t/10000^{i/d})$ if i is even, and $\cos(t/10000^{i/d})$ otherwise.

The MHA mechanism allows the model to create multiple sequence representations by projecting the encoded sequences into multiple sub-spaces. Scaled dot-product attention [171] is then applied separately in each sub-space.

When TD models are applied to sequences of discrete symbols, including language, they are trained using the maximum-likelihood objective of minimizing the negative log-likelihood of observed sequences, $-log(P(seq; \Theta))$, with parameters Θ. The probability of a length n sequence, $s = (x_1, ..., x_n)$, is computed using the auto-regressive factorization $p(seq; \Theta) = \prod_{i=1}^{n} p(x_n | x_1, ..., x_{n_1}; \Theta)$, which is implemented by the TD. In applications where transformers are used to model sequences of continuous data, the auto-regressive factorization is still used. While a maximum-likelihood based objective is used by some [40], a more common approach is quantile loss [88, 92].

2.4 Related work

There are several different approaches to creating and evaluating synthetic financial time series. Here, we give a brief overview of the most relevant works.

2.4.1 Synthetic financial time series

Traditionally, agent-based simulation models were used to generate synthetic sequences of financial banking data [6, 97], similar to the type of data modeled in our work, as well as for generating synthetic stock-market data [22, 81, 122].

Methods based on DGMs have recently been shown to generate more realistic, univariate financial sequences than agent-based approaches [76, 150, 157, 178]. There is less research on generating multivariate financial data. A GAN model for generating

multivariate sequences of stock option prices was proposed in [177]. The work most similar to ours is StockGAN [87], which generates synthetic stock-market order-stream data, where each sequence item contains information about the order price, quantity, type, and date. However, StockGAN is a GAN based approach which generates stock market data, making it still quite different from our work.

A critical difference between the banking data we are interested in and the datasets used in these works on financial time series is the transactional nature of our data. The previously mentioned works all aim to model sequences where measurements are taken at regular intervals, such as daily stock prices. In our transactional data, the time between transactions varies, and the timing information plays a critical role influencing the transactional properties. Existing work on modeling transactional data with DGMs is limited, and we are not aware of other works which have solely focused on this task. In the papers which introduced both TG [185] and DG [94], the authors briefly discuss how their models can be used on data with irregular time intervals. In both cases, the authors suggest adding a time delta feature to indicate the time between elements and modeling this like a typical continuous feature. However, neither of these works attempts to show that their models can learn patterns based on dates or times.

To the best of our knowledge, transformers have not yet been applied to the task of generating synthetic financial time-series data. Originally proposed as a language model, transformer models such as GPT-3 can generate novel text with narrative structure [21]. Transformers have also been applied to modeling other types of time series data, including influenza prevalence [179], as well as electricity usage and traffic [40, 88, 92].

2.4.2 Evaluation of synthetic sequence data

The evaluation of synthetic data depends upon its planned use. If synthetic data is planned to augment training data, then one approach is to train the model on synthetic data and evaluate its predictive performance on real data [94, 185]. If it can achieve comparable accuracy on real data to a model trained on real data, then this is taken as evidence of the quality of the synthetic data. This approach is less valuable when the use of the synthetic data is not known a priori.

Continuous data. A simple way to evaluate synthetic financial time-series data is to compare univariate distributions, using metrics such as the 1-Wasserstein distance [178] or Kolmogorov-Smirnov distance [6, 87]. For multivariate data, these distances can be computed separately for each feature of interest [87]. A limitation is that these metrics do not consider interactions between features, nor sequence order. Due to the limited work in generating multivariate banking data, there are no domain-specific metrics we are aware of. In works on financial sequences of asset prices, such as [87, 178], domain-specific metrics were used that focused on well-documented features that occur in real market data known as *stylized-facts* [27].

Categorical data. [87] studied synthetic financial time-series that generates data with both categorical and continuous-valued features, however, their evaluation only focused on continuous features. [186] use a randomly initialized LSTM model to generate a dataset of discrete sequences that were used to train their sequence generator. The LSTM model was then used to evaluate the likelihood of the data produced by the generator. [90] adopt a similar approach, performing additional validation experiments on real text sequences. To evaluate the quality of the generated text, they use BLEU scores [124], which measure the proportion of N-grams in the generated data that also occur in the real data.

2.5 Banksformer

We have created a modified TD model, called Banksformer (BF), to generate multivariate sequences of banking transactions. There are two main innovations in the design of BF. First, a preprocessing step (described below) allows BF to model sequences of items that contain multiple features of different types, including continuous and categorical features, as well as dates. Second, BF uses a novel method for generating multivariate time series data, in which each field of a transaction is generated sequentially. Our results indicate that this allows BF to better learn the joint distribution, such as $p(amount, tcode)$ as the product of two simpler distributions $p(amount, tcode) = p(tcode)p(amount|tcode)$.

2.5.1 Date Mechanism

The unique way Banksformer handles dates involves two parts – encoding and prediction. In BF, we create multiple features based on the timestamp to facilitate learning date-based patterns. Specifically, the day of the month (DoM), the number of days until the months' end (DTME), the day of the week (DoW), and the month of the transaction are each represented using two features. The two features are $f_1 = \sin(2\pi i/n_i)$ and $f_2 = \cos(2\pi i/n_i)$, where i is and ordering index and n_i is the number of possible indices (e.g., $i = 0$ and $n_i = 12$ when encoding the month of January). Additionally, BF also models a time delta (Δ_t) feature, as is done in TG and DG.

The way we have chosen to encode the date information helps BF learn date patterns; however, it also clearly contains redundancy. When generating data with BF, we first generate a probability distribution over the result for each date feature, and then create a distribution over the transaction date as

$$p(date) = \prod_{field \in \{DoM, DTME, DoW, month, \Delta_t\}} p_{field}(date[field]). \tag{2.1}$$

We implement this with the following approach. First, a maximum time between transactions is set to make the approach feasible. The distribution over the time delta feature is modeled with a truncated Gaussian distribution, covering the range from 0 to the maximum time. Typically Gaussian distributions are used for modeling continuous features in DGMs, and we use a truncated distribution to ensure the time delta is non-negative. BF outputs two features for the time delta, which are interpreted as the mean and variance to the truncated Gaussian. For each of the other features, BF outputs a categorical distribution over the options, which is created by a softmax layer. To compute the normalizing constant for the distribution, we sum the normalized probabilities of all dates between 0 and the maximum number of days from the current date. We then sample a date from this distribution, and then convert the selected date back into the separate date features.

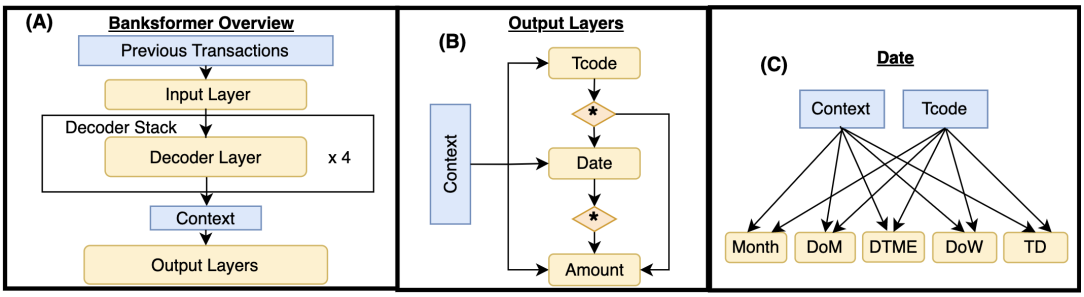

Figure 2.1: An illustration of BF. (A) An overview of BFs architecture. (B) A zoomed-in view of the output layers, showing how BF sequentially handles transaction parts. When generating data, the * boxes indicate a sampling operation that samples a value from the input distribution. During training, teacher forcing is used, and the * boxes indicate the true value which should have been produced by the input distribution. (C) A further zoomed-in view of the date layer, showing that each piece of date information is predicted independently from the context, which encodes the sequence of previous transactions and the true value of the tcode for the current transaction.

2.5.2 Architecture

Figure 2.1 outlines the architecture of BF, which is composed of 3 main parts. The input layer takes a sequence of multivariate transactions and maps it to a d_{model} dimensional sequence to which the positional encoding is added. The decoder stack then processes the encoded sequence and emits a context sequence that encodes predictions about the next element in the sequence. Finally, the output layers process each context and transform them into transaction predictions.

Input Layer. The input layer in BF is fully connected and simply maps the input data with dimension d_{input} to a representation with dimension d_{model}, which is used throughout the decoder stack.

Decoder Stack. After the input layer, BF contains a stack of 4 identical decoder layers, following a similar design as the decoder layers used in [171]. Each decoder layer is composed of two sub-layers. The first is a masked multi-head self-attention layer. This layer allows the network to attend to all sequence positions less than i when predicting the i^{th} element. This design follows the decoder stack in [96]. The final decoder layer emits a vector with size d_{model}. Our BF synthetic datasets were created using $d_{model} = 128$, which was chosen by cross validation.

Output Layer. In BF, the output contains multiple important pieces of information. This work focuses specifically on three: a categorical tcode, a transaction date, and a real-valued amount. The output layer of BF contains a conditional generation mechanism, which generates each of these values sequentially, and conditions each value on all previous ones. In the end, our model represents the probability distribution of the k^{th} transaction ($trans_k$) in a sequence as $p(trans_k|hist) = p(tcode_k|hist) \cdot p(date|hist, tcode_k) \cdot p(amount|hist, tcode_k, date)$, where $hist$ is the transaction history up to the k^{th} element of the sequence.

Loss Function.

The loss function used for training BF treats each piece of information within a transaction separately, with the overall loss a weighted sum of individual losses. Aside from the continuous time delta, all the other date-based features are treated as categorical variables, and BF outputs for each of these features are interpreted as probability distributions over the possible options. Mean-squared error is used as the loss function for continuous features and categorical cross-entropy for categorical features.

2.5.3 Generating data

BF generates synthetic data in the following way. The first element of the sequence contains the metadata, transformed into a vector with the same dimensionality as the feature dimension of the training sequences. In the *czech* data, this is the age of the main account holder, and in the *uk* data we simply use a vector of zeros to indicate the sequence start, as no metadata exists for these sequences. A sequence of l transactions is then iteratively generated. At each step, the current generated sequence is passed as input, and the next element in the sequence is output. This element is then concatenated to the existing generated sequence. When generating a transaction, BF generates each attribute in a predefined order, conditioning each attribute on all previous attributes. Each generated attribute is output by a unique, fully connected layer. For categorical attributes, the raw output from the associated layer is passed through a softmax function to create a probability distribution over possible values, and the generated value is randomly sampled from this distribution. For continuous attributes, BF outputs two features, which are treated as the mean

and variance of a normal distribution, and the generated value is sampled from this distribution. The transaction date is sampled from the distribution in Equation 2.1, following the method detailed in Section 2.5.1.

2.6 Results

In this section, we present a comparison of synthetic data generated by BF, TG, and DG on both the *czech* and *uk* datasets. The figures in this section focus on results from the *czech* dataset. For BF and TG, all synthetic sequences had an equal number of transactions (20 for *uk*, 80 for *czech*), and the start dates (plus ages for the *czech* data) were randomly sampled from the empirical distribution in the real datasets. In contrast, DG generates sequence lengths and meta-data along with the transaction sequences. To better understand the quality of our generated data, we use a set of metrics to evaluate multiple aspects of our synthetic data.

2.6.1 Univariate distributions

The most straightforward metrics are based on comparing univariate feature distributions for the continuous and categorical features. We use the Wasserstein-1 distance [133] for distributions of continuous variables and the Jensen Shannon divergence (JSD) [115] for discrete variables to quantify the difference in univariate distributions. These both measure distances of univariate probability distributions (for more information see [133] & [115]).

For continuous variables, we compare the distributions of transaction amounts and monthly cash flow. The monthly cash flow of an account is simply the sum of all credits and debits (positively and negatively-valued transaction amounts) in a given month. We are interested in cash flow distributions because, unlike the transaction amount, cash flow is not directly modeled as a variable in the training data. However, cash flow is still an important facet of bank data. If synthetic data can capture the cash flow patterns from the real training data, this will support the claim that the model is learning the actual data distribution.

As we can see from Figure 2.2, the synthetic data generated by BF best captured the monthly cash flow patterns from the real data. This is supported quantitatively as

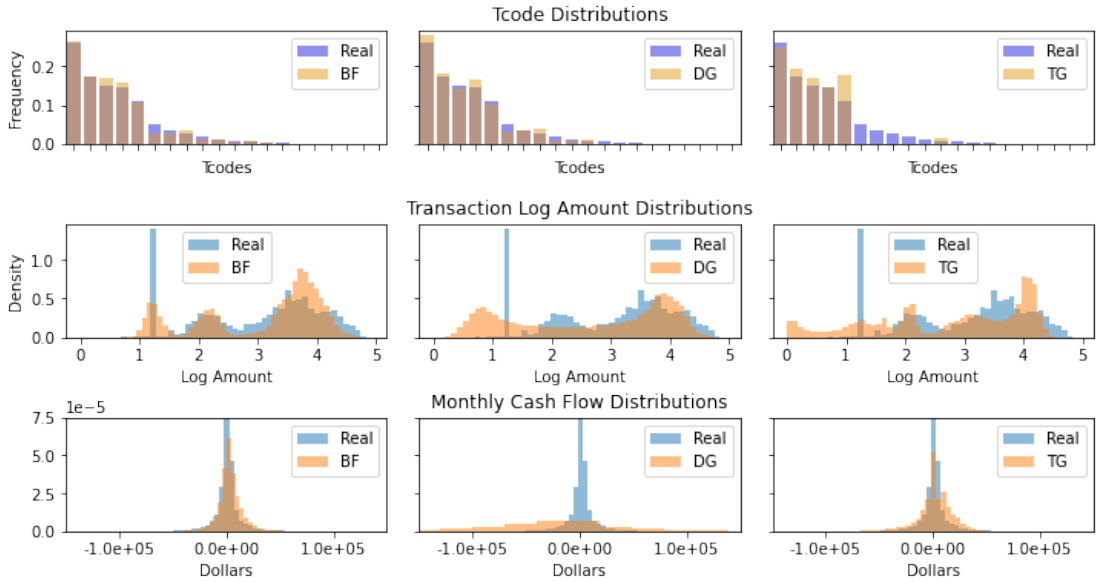

Figure 2.2: Comparison of univariate distributions in *czech* data. This figure shows a comparison of the distributions for the tcode (top), log amount (middle), and monthly cash flow (bottom) in the synthetic datasets produced by BF, DG, and TG.

well (Table 2.2). The amount distribution produced by BF was quantitatively worse than both TG and DG on the *czech* data (Table 2.2). However, when viewed on a log scale, BF appears to better capture the three modes of the real amount distribution 2.2. On the *uk* dataset, BFs' amount distribution was closest to the original data. The data generated by BF also performs best at emulating the tcode distribution in the *czech* data, which can be seen in Figure 2.2 and Table 2.2. DG does nearly as well as BF at capturing the tcode distribution on the *czech* data, and slightly better than BF on the *uk* data.

2.6.2 N-gram Distributions

An N-gram is a length N sequence of N symbols, and these symbols may represent categorical values, such as the tcode. Comparing N-gram distributions allows us to measure the models' ability to capture sequence orderings. Here we focus on 3-grams, and use the JSD to quantify differences in these distributions. We experimented with other values of N and the results did not change significantly. However, the JSD becomes harder to estimate as N increases because the empirical N-gram distributions

Table 2.2: Full Results Summary. The first 2 score columns are the Wasserstein-1 distances comparing the univariate amount (AMT) and monthly cash flow (CF) distributions respectively. The next two columns are JSD results comparing the univariate distributions of the tcode (Tcode) and transaction day of the month (DoM). The final columns are also JSD results. The Tcode 3G column show the JSD between the distributions of tcode 3-grams. And finally, the (Tcode, Date*) column compares the joint distributions of tcode and the most significant categorical date feature, which is DoM for the *czech* data, and DoW for the *uk* data. For each dataset, the bottom three rows show the results of ablation experiments (Section 2.6.4).

Data	Model	Amt	CF	Tcode	DoM	Tcode 3G	Tcode, Date*
czech	BF	2102	**2738**	**0.004**	0.011	**0.042**	**0.251**
	DG	1939	57800	0.007	0.090	0.132	0.660
	TG	**1931**	4980	0.075	0.059	0.337	0.638
	BF-ND	3705	4191	0.009	0.059	0.059	0.595
	BF-NC	3580	4775	0.158	**0.006**	0.411	0.542
	TF-V	4726	4138	0.185	0.059	0.445	0.674
uk	BF	**42.6**	**541.8**	0.015	**0.024**	0.156	**0.008**
	DG	179.0	1051	0.011	0.034	0.135	0.061
	TG	116.0	1460	0.237	0.087	0.622	0.077
	BF-ND	64.1	814.8	**0.002**	0.048	**0.112**	0.062
	BF-NC	258.0	1418	0.013	0.041	0.176	0.024
	TF-V	236.0	1402	0.004	0.047	0.139	0.063

become worse estimates of the true N-gram distributions due to the curse of dimensionality. We attempted to mitigate this with *additive smoothing* [103], however this did not significantly change the results, so the results we present are based solely on comparing empirical distributions.

Figure 2.3 compares the distributions of the most common N-grams, and shows both BF and DG produce more accurate N-gram distributions on the *czech* data than TG. This is supported by quantitative results in Table 2.2, which also show that BF outperforms DG in terms of the JSD metric on the *czech* dataset. This metric also shows DG performs slightly better than BF on the *uk* dataset.

2.6.3 Joint distributions

One limitation of the previous metrics is that they do not account for how well feature interactions are modeled in the synthetic data. To get a sense of the overall joint

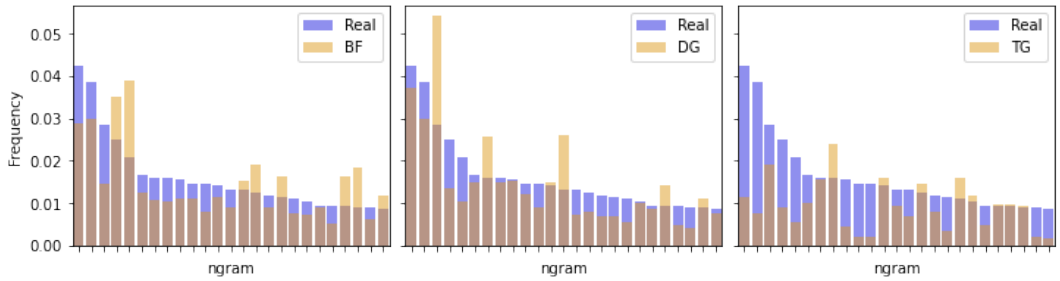

Figure 2.3: 3-gram frequency comparison. This figure compares the frequency of the 25 most commonly occurring 3-grams in the real *czech* data, for each of the synthetic datasets.

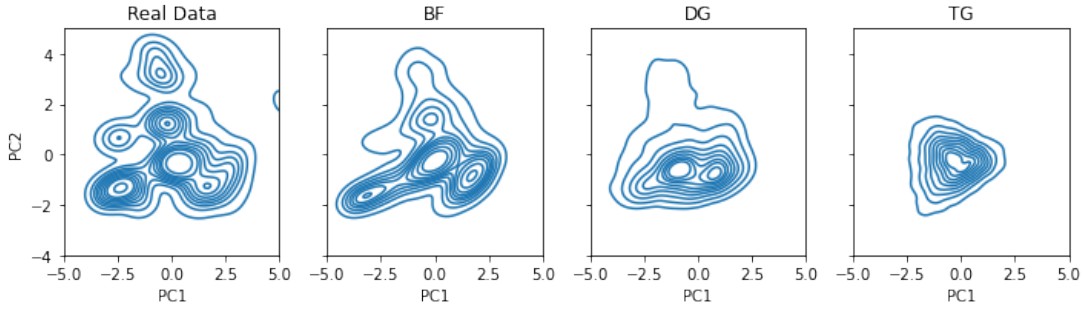

Figure 2.4: PCA visualization of *czech* data. The two principal components of the data distributions obained using PCA. The generated data are projected using the PCA model that was fit to the real data.

distribution, we can visually compare the distributions of two-dimensional projections of the datasets (Figure 2.4). To create this visualization, we follow the approach of [185], The sequences were first flattened along the temporal dimension and then a PCA model was fit to the real data. All data sets are projected into 2D using this PCA fit. Figure 2.4 shows that are multiple peaks in the real *czech* data, and that BF reproduces these peaks on the whole. DG only poorly reproduces the real data, yielding a bimodal distribution, and TG focuses on a single mode.

Figure 2.5 shows the distribution over the day of the month for two specific tcodes that only occur at specific times of the month. The top row is for interest credited to the account, which only happens on the last day of the month, and the bottom row is a type of debit transaction that only occurred between the 5^{th} and 14^{th} of the month. BF is the only model able to learn the date pattern associated with these tcodes. In particular, our model can correctly generate transactions at the end of the month, even though the last day of the month may occur on days 28 to 31.

The JSD may be used to quantify how well the relationship between tcodes and categorical date features were learned in general. Table 2.2 shows the JSD for the *czech* data, using the joint (tcode, DoM) distributions, and the *uk* data, using joint (tcode, DoW) distributions. For both data sets, BF significantly outperforms both DG and TG.

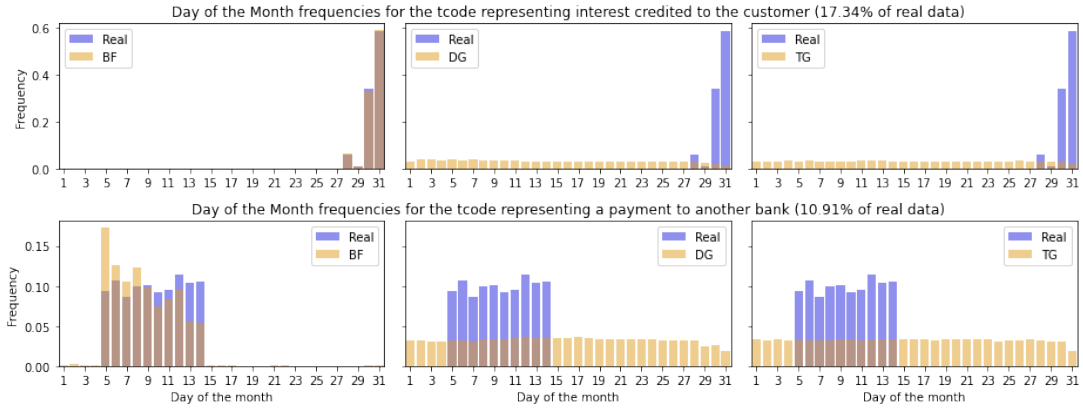

Figure 2.5: Date, tcode relationship in *czech* data. This figure shows the conditional distribution of the transaction day of the month, given the tcode, for two tcodes that are strongly related to the date. This figure shows that BF (left) is the only model which has learned the relationship between these tcodes and the date.

Different transaction types also have different associated amount distributions. In Figure 2.6, we compare the conditional amount distributions for the two most common tcodes in the *czech* dataset. In this figure, we can see that BF, TG and DG all appear to have approximately learned the relationship between amount and tcode. Additionally, this figure also shows qualitative differences in the conditional amount distributions produced by the different models. BF tends to produce narrower, symmetric distributions, which are centered near the mean of the real data; whereas both DG and TG tend to produce much wider, asymmetric distributions.

2.6.4 Ablation

To illustrate the impact of the innovations behind BF, we perform ablation experiments on conditional generation and date generation mechanisms. Specifically, we create the following three ablated versions of BF:

- A version without the date mechanism (BF-ND). In this implementation, we

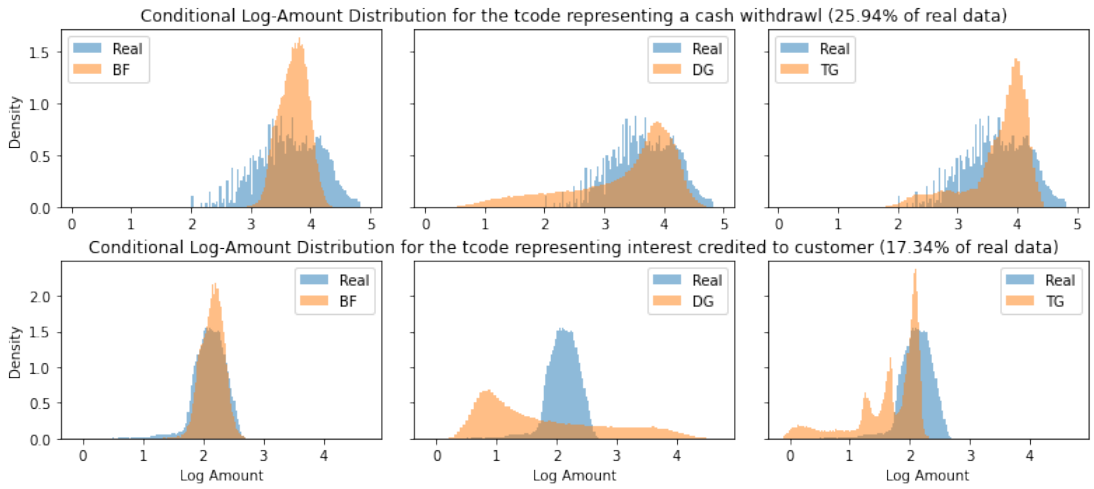

Figure 2.6: Amount, tcode relationship in *czech* data. This figure shows a comparison of the conditional distributions $p(\log(amount)|tcode)$ produced by BF, DG and TG, against real data for the two most common tcodes in the real data.

model the date using only the time delta feature, as is done in TG and DG.

- A version without conditional generation (BF-NC). In this implementation, we generate all transaction fields simultaneously.
- A basic transformer model with neither mechanism (TF-V).

The results from these experiments are shown in Table 2.2, which validate that both mechanisms introduced to the architecture of BF led to improved performance on most metrics. This was particularly true for the metrics that measured joint distributions, as well as metrics related to the amount, where BF scored much better than the ablated versions, and TF-V scored noticeably worse. For other metrics, different ablations had different impacts. The BF-NC version did worse than BF-ND on comparisons of both the tcode and tcode 3-gram distributions, with BF-NC being comparable to TF-V on these metrics. Similarly, BF-ND does worse than BF-NC and is comparable to TF-V on the DoM metric, which compares the distributions of transaction day of month. BF-ND does outperform BF on the tcode related metrics for the *uk* data, however this is not surprising, as dates are not needed to generate data which captures these patterns.

Overall, these results are in line with expectations. It is somewhat surprising that the conditional generation mechanism improved the distributions of tcodes and tcode

3-grams, as the tcode is the first feature produced when generating conditionally. It may be that without conditional generation, the other features become more difficult to model, causing the model to spend more effort learning those relationships, and less on the tcodes. We plan to investigate this further in future work.

2.7 Discussion

Our experiments show that the design of BF led to a clear improvement over TG and DG in modeling financial transactional sequences. Qualitatively, the most significant area of gain is in modeling the joint relationship between dates and transaction types, as only BF was able to learn these. Quantitatively, BF also created data that better matched the statistical properties of real data, according to the majority of the metrics we considered. Through ablation experiments, we demonstrated that both of BFs' innovations, the date mechanism and conditional generation for the individual transaction fields, improved synthetic data quality.

We believe a promising future direction for this work is to explore hybrid models and combine innovations from BF, TG, and DG. There are multiple approaches we have in mind to explore this idea, including adapting the date mechanism from BF to GAN models based on TG and DG, and adding an adversarial training step to BF.

Another critical area for future work is to examine the privacy implications of these models. One major motivation for studying synthetic banking transaction data is to minimize reliance on real private data. However, before these models can be used to generate synthetic data to replace real data with genuine privacy concerns, users must be aware of any potential information which could be leaked through synthetic datasets.

Chapter 3

Enhancing Banksformer: Additional Analysis, Ensemble Methods, and Differential Privacy

In this chapter we extend the work of the previous chapter by introducing new data quality metrics and visualizations, exploring privacy implications, and proposing an ensemble approach to further improve upon previous results.

3.1 Introduction

As we have discussed throughout the previous chapters of this thesis, synthetic data has many important use cases, particularly in privacy-sensitive domains such as personal finance. While the work presented in Chapter 2 represents an important step towards generating high-quality synthetic banking data which can replace real user data, our synthetic data is still not a perfect substitute for real data. Further, while we have succeeded at creating synthetic sequences which are not tied to any real customers, the synthetic data we produce does not have any formal guarantees of protecting user privacy.

In this chapter, we work towards three goals. First, we aim to better understand how the structure of our synthetic data differs from the real data on which we train. To this end, we conduct a qualitative analysis of the real *czech* dataset in Section 3.2,

and develop a quantitative evaluation framework in Section 3.3. Our second goal is to improve upon the quality of synthetic data we generate, by creating ensemble datasets which combine synthetic data from multiple BF models, which we describe in Section 3.4. And finally, in Section 3.5 we aim to determine the feasibility of creating a BF model which can give formal privacy guarantees on the synthetic data we generate.

3.2 Qualitative Analysis

In this section, we begin by performing an in-depth qualitative analysis of the real *czech* dataset, described in Section 2.2 of the previous chapter, and how it compares to the synthetic data we have generated. Based on this analysis, we then propose additional metrics to quantify the quality of synthetic data.

When generating synthetic images or text sequences, one important form of quality analysis is manually inspecting samples. Since most humans encounter a large amount of image and textual data, we are naturally suited to recognize real data from these domains. However, we observe that visually inspecting transaction sequences, to determine if they 'appear realistic', is a much more difficult task. In Figure 3.1, we show a comparison of snippets from real and synthetic sequences. While it is possible to pick out some differences visually between this pair – such as the fact that real transactions tend to have rounder amount values – visually inspecting accounts is of limited value.

One limitation of the metrics used in Chapter 2 is that they evaluate the data at the transaction level as opposed to the account level. The n-gram based metric described in Section 2.6.2 is the only one of these metrics which accounts for the sequential nature of the data; however, it only considers length three subsequences, and thus is of limited use for understanding if our synthetic data has the same structure as the real data. As a concrete example of this limitation, we describe the distribution of pension payment transactions in the real *czech* dataset described in Section 2.2. Overall, pension payments account for about 2.9% of transactions in the *czech* dataset. However, these pension payments are not evenly distributed among accounts. In fact, over 83% of accounts have zero pension payments, whereas pension payments constitute over 25% of the transactions from the remaining 16.4% of accounts with at least one pension payment. Ideally, we would like our synthetic data to have a

amount	type	k_symbol	operation	datetime
3679.0	CREDIT	nan	COLLECTION FROM ANOTHER BANK	1995-04-13
12600.0	CREDIT	nan	CREDIT IN CASH	1995-04-23
19.2	CREDIT	INTEREST CREDITED	nan	1995-04-30
3679.0	CREDIT	nan	COLLECTION FROM ANOTHER BANK	1995-05-13
2100.0	CREDIT	nan	CREDIT IN CASH	1995-05-23
79.0	CREDIT	INTEREST CREDITED	nan	1995-05-31
3679.0	CREDIT	nan	COLLECTION FROM ANOTHER BANK	1995-06-13
200.0	DEBIT	nan	CASH WITHDRAWAL	1995-06-22
100.6	CREDIT	INTEREST CREDITED	nan	1995-06-30
3679.0	CREDIT	nan	COLLECTION FROM ANOTHER BANK	1995-07-13

Example Real Sequence

amount	type	k_symbol	operation	datetime
9.26	DEBIT	PAYMENT ON STATEMENT	CASH WITHDRAWAL	1997-03-22
144.48	CREDIT	INTEREST CREDITED	nan	1997-03-31
4.00	DEBIT	PAYMENT ON STATEMENT	CASH WITHDRAWAL	1997-03-31
338.81	DEBIT		REMITTANCE TO ANOTHER BANK	1997-04-05
832.23	CREDIT	nan	CREDIT IN CASH	1997-04-06
83223.72	DEBIT	HOUSEHOLD	REMITTANCE TO ANOTHER BANK	1997-04-07
3449.65	DEBIT	nan	CASH WITHDRAWAL	1997-04-08
3337.93	CREDIT	nan	CREDIT IN CASH	1997-04-21
4668.89	CREDIT	INTEREST CREDITED	nan	1997-04-30
100.60	DEBIT	PAYMENT ON STATEMENT	CASH WITHDRAWAL	1997-04-30

Example Synthetic Sequence

Figure 3.1: Example transaction sequences from real and synthetic datasets.

similar distribution of pension payments. Unfortunately, our existing set of metrics only quantifies how well the overall tcode frequencies in synthetic data match the real data at the transaction level.

In order to get a better understanding of the data structure, we first devised an approach to visualize accounts behavior. As described in Section 2.2 of the previous chapter, transactions in the *czech* dataset have three categorical features that contain information about the transaction and can be combined to create the tcode. Though the tcode is the most informative categorical feature, there are 16 options, making it difficult to visualize. Therefore, we focus on the the *operation* feature for our visualization in this section, as this feature has only six options, but is still informative about the transaction type. Additionally, in Section 3.3.1 we propose a new metric based on the intuition developed in this section, which uses the tcode.

Figure 3.2 compares six groups of accounts, which are based on the six options of the operation variable. Specifically, each row contains the 50 accounts from each dataset with the highest proportion of each type of transaction, such as "cash withdrawal" or "collection from another bank". Our visualization is similar to a stacked bar chart. Each bar in our plot corresponds to an account, and we normalize the bar height, so each bar shows the proportion of different transaction types in the account.

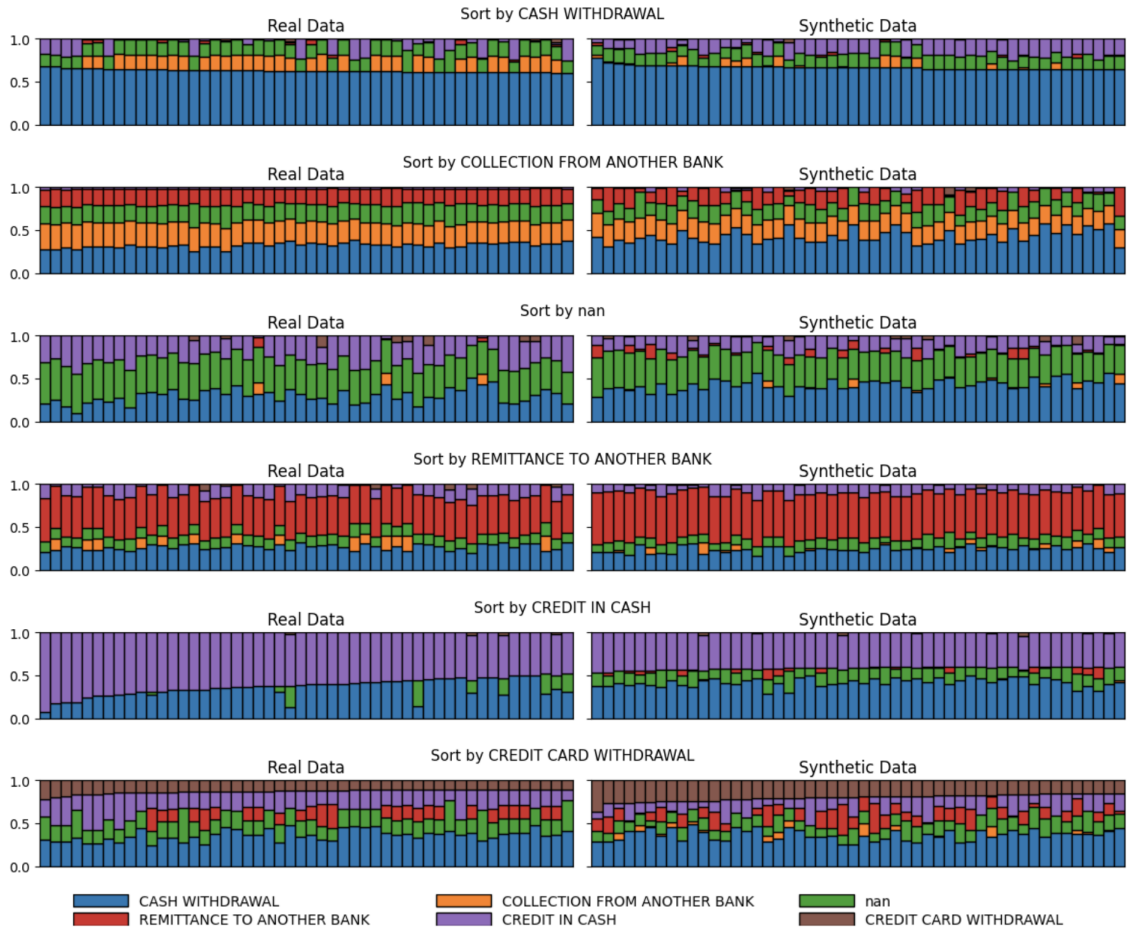

Figure 3.2: Visual comparison of real and synthetic accounts with respect to the *operation* feature. Each row is based on a different option for the *operation* and shows the 50 real and 50 synthetic accounts with the highest proportion of that option.

The goal of our approach is not to directly compare accounts, rather, it is to compare groups of accounts with similar behaviors.

Figure 3.2 shows that generally, the real and synthetic accounts appear to have similar structure. Particularly in the second and fourth rows, we see that the types of transactions and their relative proportions in accounts are very similar between both datasets. However, there are also noticeable differences between the real and synthetic data. The synthetic accounts tend to have more 'cash withdrawal' transactions than real accounts in most rows. Another difference is the synthetic accounts tend to have more unique transaction types than real accounts. This is particularly noticeable in the 5th row of Figure 3.2, which has the accounts sorted by 'credit in

cash'. Many of the real accounts only have two types of transactions ('credit in cash', 'credit withdrawal'; indicated by blue and purple respectively); whereas the synthetic accounts all have at least three types.

3.2.1 Behavior Patterns

Figure 3.2 suggests that it may be possible to group accounts by *behavior patterns*, as it appears as though each row shows sets of accounts which are used in different manners. In order to determine the significant *behavior patterns* which occur in the real data, we apply a method called Latent Dirichlet Allocation (LDA). LDA is a versatile method, which has been applied to a wide range of tasks, including language modeling [15], population genetics [131], and many more [91, 135]. Within the machine learning community, this method was popularized in the context of language modeling by [15], and therefore most of the literature on LDA uses terminology from this domain.

LDA is applied to a set of *documents*, each document is defined by a sequence of *words*, and LDA aims to find a set of underlying *topics* on which the documents are based. LDA assumes that each document is a mixture of a small number of topics, and each topic defines a distribution over words. Given a set of N documents, and a number k indicating the number of topics, LDA outputs a set of k topic prototypes and a $N \times k$ matrix indicating the topic distribution for each document. While each document is represented as a mixture of multiple topics, we say a document's *primary topic* is the topic that has the highest weight in the document. In the context of banking data, we apply LDA to a set of *accounts*. Each account is defined by a sequence of *transactions*, and we use LDA to find a set of underlying *behavior patterns* on which the account sequences are based. When applying LDA, we represent transactions with a single categorical value, such as the *operation* or *tcode*.

We begin by applying LDA to the real *czech* data with respect to *operation* feature. We use the *operation* here because it is easiest to visualize. An important preliminary step before applying LDA is determining the number of topics for the model to use. Using cross-validation, we determined that when considering the operation feature, $k = 4$ is the most likely number of behavior patterns in the real data.

To get a sense of these behavior patterns, we visualize 40 randomly selected accounts for each of the four primary behavior patterns in Figure 3.3. Visually, the

behaviors in the synthetic data are quite similar to those from the real data, with a few noticeable differences. For one, the synthetic data has a much lower proportion of sequences with pattern #1 as their primary pattern (11.38% vs. 35.00%) and a much higher proportion with pattern #3 (75.88% vs. 52.80%). The other main noticeable difference between real and synthetic data is that real sequences tend to have fewer unique types of transactions, as was the case in Figure 3.2.

Despite these minor differences, for each of the four distinct patterns shown in Figure 3.3, real and synthetic accounts tend to exhibit very similar behaviors. Accounts which fall under the first behavior pattern typically have about 40% of their transactions as 'cash withdrawal', and they also have significant proportions of 'collection from another bank', 'nan' and 'remittance to another bank' transactions. In pattern #2, there are a large proportions of both 'cash withdrawal' and 'remittance to another bank' transactions, and also significant proportions of 'nan' and 'credit in cash'. The third pattern captures accounts with significant proportions of 'cash withdrawal', 'credit in cash', and at least one of 'collection from another bank' or 'nan'. Finally, pattern #4 is similar to #3, with the difference that accounts falling under pattern #4 have significantly more 'credit card withdrawal' transactions. Overall, all four patterns have a significant amount of 'cash withdrawal' transactions, which makes sense, as 41% of all transactions are 'cash withdrawal'.

One limitation of LDA is that it is a randomized algorithm, and depends on the user specifying the number of topics. While using this method to visualize accounts provides us with insight into the different types of account behaviors that exist in the real data, we should recognize that it is based on a random algorithm, and the exact patterns found may be different if the algorithm is run multiple times with different random seeds. For a user who is interested in applying this visualization to understand account behaviors in a transaction sequence dataset, we would recommend running this procedure multiple times, and observed which patterns are typical. In our experiments, we found that if we re-ran the cross-validation process to choose the number of topics, we sometimes found different values, ranging from three to eight. However four was the most common value, and the results in Figure 3.3 are fairly typical. Overall, we primarily use this analysis to see if the behavior patterns in the synthetic data are similar to those in the real data, and not to determine a set of ground-truth patterns for describing the real data, so this randomness is not a major issue for our application. Further, in Section 3.3.1, we propose a novel metric for

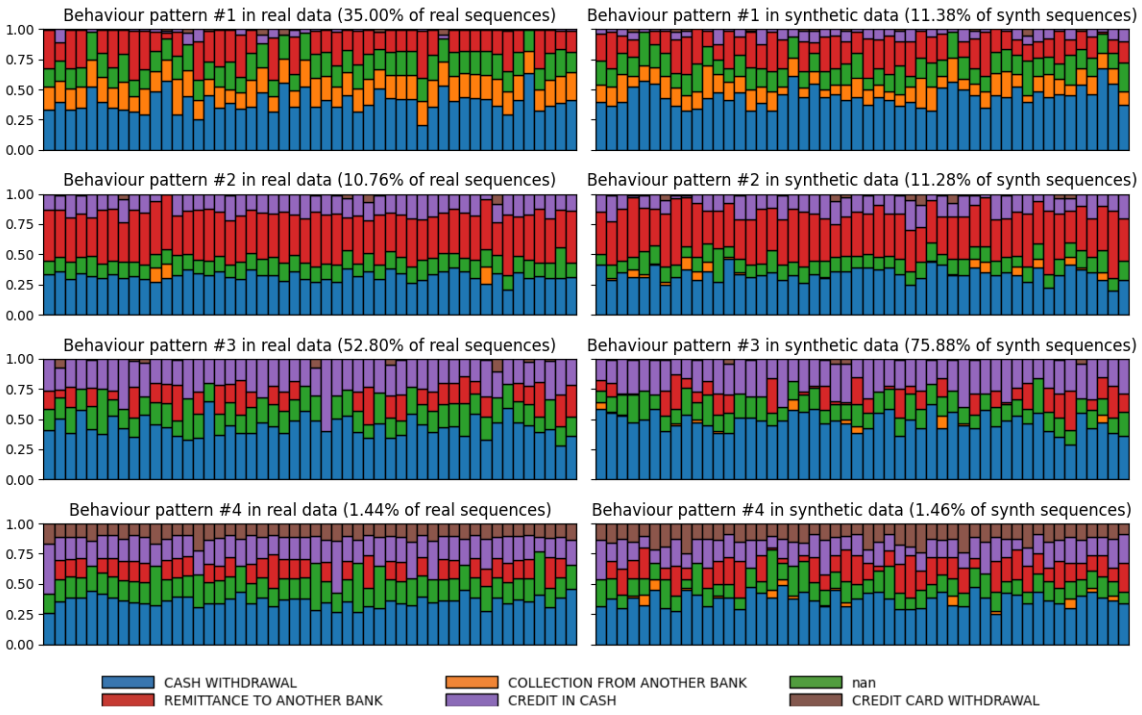

Figure 3.3: Visual comparison of real and synthetic accounts with respect to the *operation* feature. Each row shows a random 50 real and 50 synthetic with a different primary behavior pattern, as determined by a 4 topic LDA model fit to the real data.

synthetic data quality based on this intuition.

3.3 Quantitative Evaluation Framework

In Chapter 2, we relied on a useful, but imperfect, set of metrics to evaluate the quality of the synthetic data we produced. As was discussed, there is no single metric which can quantify 'how good' a synthetic dataset is; in fact has been argued that often the best metrics take into account the type of data and how it will be used [65, 159]. However, in Chapter 2, our metrics were all based on comparing feature distributions. While it is important that the feature distributions in synthetic data are similar to the distributions from the target real data, this is only one facet of data quality. In this section, we propose a suite of additional metrics, and describe the overall framework we use for comparing the quality of datasets.

3.3.1 LDA metric

One limitation of the visualization approach used above is that it is only useful for visualizing account behavior with respect to a categorical feature with a small number of options. In this section, we propose an LDA-based metric for evaluating the structure of synthetic banking data. The metric is still based on a single categorical feature, however it can be used with features with many options. In our experiments, we use the tcode feature when computing this metric, as the tcode contains information from the three unique fields. The idea behind our metric is that an LDA model fit to real account data should be a good model for synthetic accounts and vice versa. To this end, we are interested in the following two quantities:

$$score(lda_{synth}, real) - score(lda_{real}, real) \tag{3.1}$$

$$score(lda_{real}, synth) - score(lda_{synth}, synth) \tag{3.2}$$

While there are various methods for scoring an LDA model, we define $score(lda_X, Y)$ as the perplexity score of an LDA model fit to data X and evaluated on Y. The quantity in Equation 3.1 shows the increase in the perplexity score of encoding real data with a model fit to synthetic data, as opposed to a model fit to real data. Similarly, Equation 3.2 shows the increase in the perplexity score of encoding synthetic data with a model fit to real data, as opposed to a model fit to synthetic data. As we view each of these quantities as equally important, our LDA based metric is the mean of these values:

$$\frac{score(lda_{synth}, real) - score(lda_{real}, real) + score(lda_{real}, synth) - score(lda_{synth}, synth)}{2} \tag{3.3}$$

Figure 3.4 shows a typical example of LDA scores for models using real and synthetic data. The significant takeaway from this figure is that the test data has a much greater impact on the perplexity score than the training data. We see that within each column, the values are quite close, indicating that whether we train on real or synthetic data, the test performance will only change minimally. Conversely, within each row, the difference in values is much larger, with models tested on real data always achieving lower perplexity than models tested on synthetic data, regardless

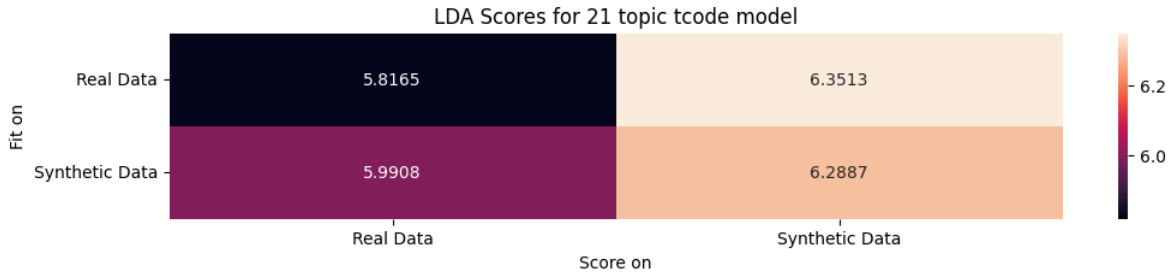

Figure 3.4: LDA score matrix example, showing the perplexity scores (lower is better). This figure demonstrates that LDA models trained on either real or synthetic data perform similarly, evidenced by the small difference between values within each column. The relatively larger differences within each row, and the fact that both models score worse on the synthetic data, imply that the synthetic data has higher entropy.

of whether the model was fit to real or synthetic data. The fact that the perplexity values when testing on synthetic data are always higher than the values when testing on real data implies that the synthetic data distribution has greater entropy than the real data. This is in line with our qualitative analysis, which found that in the real data many accounts only have a small number of transaction types; whereas, in the synthetic data, accounts typically perform more types of transactions, making their transaction sequences harder to predict. This can be seen in both Figure 3.2 and Figure 3.3.

3.3.2 Distinguishability

One critical aspect of synthetic data quality in many applications is that synthetic samples should be indistinguishable from samples from the target distribution [4]. In order to test this property, we perform a set of experiments using machine learning classifiers to try and distinguish between real and synthetic samples. We then conducted a set of distinguishability experiments using a fixed-sized account representation and various standard machine-learning classifiers.

To create the fixed size representation, we first apply the method described in Section 2.2 to encode a batch of sequences into a 3D tensor with shape $(n_{sequences}, n_{steps} + 1, dim_{features})$, with the first step encoding sequence meta-data. We then remove the meta-data and take the mean of the remaining data over the n_{steps} dimension to

create a $dim_{features}$ dimensional representation for each sequence. In the 3D tensor encoding, categorical features are represented with one-hot encodings, and continuous features are represented with a single value, scaled so each feature has a variance of 1. After flattening the sequence, the representation encodes the mean of continuous features, and the proportion of each option for categorical features. We experimented with concatenating the meta-data together with the flattened sequence to create a $2 \cdot dim_{features}$ dimensional representation, but found that on average this hurt the classifier performance. A possible reason for this is that we only have a single meta-data feature, the customers' age, which may not be useful for distinguishing between real and synthetic sequences.

For our distinguishability experiments, we create a binary classification problem by combining real and synthetic data into a merged dataset, and training binary classifiers to distinguish between real and synthetic samples, using the flattened representations. We consider a small set of binary classifiers with different types of decision boundaries and create a metric for each classifier using their area under the receiver operator curve (AUC) score on held-out data. Specifically, we employ the scikit-learn [125] implementations of Random Forest, Logistic Regression, and K-Nearest Neighbours classifiers, with the default parameters, as well as shallow ($depth \in \{1, 2\}$) Decision Trees.

3.3.3 Transferability

Another important property of good synthetic data is that inferences made from the synthetic data should also be valid for real data. For example, a machine learning classifier or regressor trained on synthetic data should perform similarly to a model trained on real data. In the context of the *czech* banking data, we devised a set of experiments that involve either predicting the transaction type (classification) or transaction amount (regression) from the other features.

To quantify this, we use a similar approach as the LDA-based metric. Specifically, we compute

$$\frac{score(RF_{synth}, real) - score(RF_{real}, real) + score(RF_{real}, synth) - score(RF_{synth}, synth)}{2}, \tag{3.4}$$

which is the average of the difference in test scores between models trained on real

versus synthetic data when scored on real data and the difference in test scores when scored on synthetic test data.

3.3.4 Coverage

Another aspect of good synthetic data is *coverage*. The general idea, as shown in Figure 3.5, is that if we have a dataset of N real samples and N synthetic samples, then the nearest k neighbors of any sample should be roughly half real and half synthetic. Based on this idea, we implement two related metrics to quantify how well our synthetic data cover the real data.

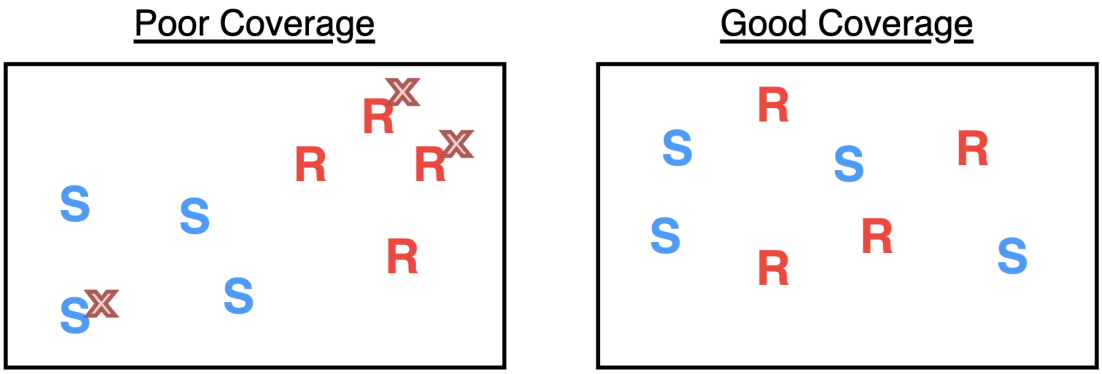

Figure 3.5: Coverage intuition. A synthetic dataset achieves good coverage with the synthetic samples (S) are interspersed among real samples (R). In the left panel, three of the eight samples shown have poor coverage with respect to their 3-neighbourhood. That is, all of their three nearest neighbors are from the same dataset. In the right panel, all samples have at least one real sample and one synthetic sample in their 3-neighbourhood, making this an example of a synthetic dataset with perfect coverage with respect to the 3-neighbourhood.

In both metrics, we define the neighborhood of a sequence as follows. First, we flatten sequences over the time dimension, using the same method used for the classifier metrics. Then, we compute the neighborhood of the flattened sequence using Euclidean distance. The first metric,

$$\frac{(\#\ samples\ with\ all\ 3-NN\ real) + (\#\ samples\ with\ all\ 3-NN\ synthetic)}{\#\ samples\ total}, \tag{3.5}$$

considers the 3-neighbourhood of samples in a combined real-synthetic dataset. Specifically, we first select N sequences from both the real and synthetic data, where $N = \min(n\_sequences_{real}, n\_sequences_{synthetic})$, and combine them into a combined dataset C. We then compute the three nearest neighbors in C for all sequences in C. The first metric is the proportion of samples from C with all 3-neighbors coming from the same dataset. The second metric,

$$\mathop{\mathbb{E}}_{x \sim S}[proportion\_kneighbours\_real(x)], \tag{3.6}$$

is also based on a balanced combined dataset C, however in this metric, we consider the 5-neighborhood of only the synthetic samples. Specifically, we take the average over synthetic samples of the proportion of real samples in the 5-neighborhood.

3.3.5 Non-memorization

An orthogonal aspect of data quality is that we do not want our model to simply generate copies of the real data. For many of the other metrics we consider, one way to create a high-scoring 'synthetic' dataset is to simply copy the real training data. This would create an indistinguishable dataset with the same distribution as the real data. However, the true value of synthetic data comes from being able to synthesize novel samples.

One way to check if the model has memorized the training data, is to look for 'identical' transactions occurring in both the real and synthetic data. In order to conduct this test, we first need to determine exactly what constitutes identical transactions. Suppose we use an overly stringent definition, such as 'identical transactions must have the same type, exact same amount, and be on the same date'. In that case, we find that identical transactions are quite rare, with only 0.09% of synthetic transactions being identical to real transactions. Simply considering this number by itself is not particularly informative about the quality of the synthetic data. If we want our synthetic data to not simply copy the training data, we might argue that synthetic data with fewer transactions identical to real transactions are better; however, since we want our synthetic data to *look like* real data, having some identical transactions may be desirable. For example, if we consider only transactions with

the tcode 'credit__nan__interest credited', which corresponds to transactions indicating interest payments, we see that 0.59% of synthetic transactions are exact duplicates of real transactions. While this is significantly higher than the overall proportion of duplicate transactions, closer examination reveals this has a reasonable cause. Transactions with this tcode account for 17% of the entire real dataset, typically have small amounts (median amount is $133.00), and only occur on the last day of the month. Therefore, it is expected that a large synthetic dataset that follows the real data distribution will have some duplicate transactions. By making assumptions about the distributions of amount and date values, we could compute an estimate of the expected number of duplications, and use this in evaluation criteria. However, we adopt a different approach, which makes no assumptions about feature distributions.

Our approach is based on [129]; the central idea is to use a validation set of real data to determine what number of duplications is suitable. Specifically, before training our BF model, we partition the real *czech* data into training and validation sets. We train BF using only the training set, then during evaluation we compare the number of duplications between the synthetic data and training data, with the number of duplications between the synthetic data and validation data. Because the training data typically contains more sequences than the validation data, we use a random sample of the training data, which is the same size as the validation data for this analysis. Overall, 0.09% of the transactions in the synthetic data are exact duplicates of samples from the real training data, and 0.09% of the transactions in the synthetic data are exact duplicates of samples from the real validation data. Therefore, it does not appear that the model memorizes the training data.

We consider two additional relaxed definitions for identical transactions. First, we use a definition that ignores the date features, and defines duplicates as transactions with the same amount and tcode. Then, we consider a further relaxed criteria, under which transactions are identical if they have the same tcode and their amounts are within 1% of each other. As we show in Table 3.1, the number of duplications increases with each relaxation of the criteria; however, there is always a similar level of duplication between the training data and validation data, suggesting we have not overfit the training data. Table 3.1 also shows a breakdown of the duplications by tcode. As discussed above, duplications may be more or less common for different tcodes. Across all tcodes, the number of duplications is similar for the training and validation data, strengthening the claim that we did not overfit the training data.

Table 3.1: Comparison of duplication percentage between training and validation data. Each row shows the percent of synthetic transactions which have a duplicate in the real data, according to different criteria. Each row shows the result based on a specific transaction type, with the last row showing the result over all types. We consider three different criteria for determining if transactions are duplicates, and for each criterion compare the duplication rate with training samples from the real data against the duplication rate with validation samples.

	Amount within 1%		Exact Amount		Exact Amount & Date	
Matches in	train	val	train	val	train	val
Tcode #1	67.11%	66.56%	0.08%	0.08%	0.00%	0.00%
Tcode #2	47.26%	47.40%	0.08%	0.08%	0.00%	0.00%
Tcode #3	59.34%	59.58%	5.77%	5.69%	0.57%	0.59%
Tcode #4	0.90%	0.90%	0.02%	0.02%	0.02%	0.02%
Tcode #5	42.01%	43.88%	0.04%	0.04%	0.00%	0.00%
Tcode #6	40.38%	40.39%	0.08%	0.09%	0.01%	0.00%
Tcode #7	24.39%	26.46%	0.00%	0.00%	0.00%	0.00%
Tcode #8	19.70%	19.70%	0.10%	0.09%	0.00%	0.01%
Tcode #9	22.32%	24.06%	0.01%	0.00%	0.01%	0.00%
Tcode #10	19.03%	20.19%	0.00%	0.00%	0.00%	0.00%
Tcode #11	8.71%	8.91%	0.00%	0.01%	0.00%	0.01%
Tcode #12	32.52%	31.74%	3.16%	3.25%	0.19%	0.09%
Tcode #13	6.97%	8.07%	0.03%	0.00%	0.00%	0.00%
Tcode #14	25.13%	27.62%	0.00%	0.00%	0.00%	0.00%
Tcode #15	3.98%	13.88%	0.00%	0.00%	0.00%	0.00%
Tcode #16	2.33%	0.00%	0.00%	0.00%	0.00%	0.00%
Overall	46.22%	46.37%	0.93%	0.92%	0.09%	0.09%

Holdout metric

The idea that the appropriate number of duplications can be determined using a validation set is powerful, and useful for more than just determining how many duplications to expect. Generally, for any metric we compute comparing a synthetic dataset against a real dataset, the value of the metric should be the same whether it is computed using the same real data which the synthetic data model was trained on, or on held-out data from the real distribution [129]. If the synthetic data scores significantly better when compared to the training data than when compared to the validation data, then this can be taken as evidence of overfitting. In Chapter 2, we computed all metrics based on comparing synthetic data to the full real dataset. However, in the interest of quantifying how much our model is overfitting the training

data, we now compute each metric with three different options for what we use as the real data: the full real dataset, the portion of the real dataset used to train our model, and the held-out portion of the real dataset not used to in training.

Computing these three values is straightforward for most metrics, such as those that compare distributions. We start with the split of the real data R into training data used for training BF, R_{tr}, and validation data R_{val}. Given a synthetic dataset S, we simply compute $metric(R, S), metric(R_{tr}, S)$, and $metric(R_{val}, S)$. However, for metrics that involve training predictive models, such as the classifier-based metrics, there is added complexity to computing these three values because we also need to consider what data is used to train and evaluate the classifier. We adopt the following approach. First, we partition the synthetic data S into training S_{tr} and validation S_{val} sets. To compute the metric based on the full real dataset, we use samples from $\{S_{tr}, R_{tr}\}$ to fit the models and samples from $\{S_{cv}, R_{cv}\}$ for scoring. To compute metrics based on training and validation sets, we further partition the data used to train BF (R_{tr}) into ($R_{tr,tr}$), ($R_{tr,val}$). When computing the training value, we use samples from $\{S_{tr}, R_{tr,tr}\}$ to fit the models and samples from $\{S_{cv}, R_{tr,cv}\}$ for scoring (only real samples from R_{tr}). For the validation value, we use the same samples to fit the models and samples from $\{S_{cv}, R_{cv}\}$ for scoring. Our *holdout metric* is defined as the mean percentage difference between the training and validation scores, averaged over all metrics.

3.3.6 Distribution metrics

The final set of metrics are based on the idea of comparing feature distributions between real and synthetic datasets. These metrics were described in detail in Chapter 2. Specifically, we use the metrics comparing univariate distributions for the amount, tcode and day of the month feature, described in Section 2.6.1, the N-gram metric from Section 2.6.2, and the joint (tcode, day of the month) distribution metric from Section 2.6.3.

3.3.7 Overall Framework

As has been made clear, the overall quality of a synthetic dataset is not a well-defined concept, and it depends on how the data will be used. However, when trying to

determine the effect of a modeling decision – such as the number of models to include in an ensemble – having some sense of overall quality is crucial. Because of this, we define a flexible framework for determining the overall quality of a synthetic dataset relative to other synthetic datasets based on the same real data. The basic idea behind our framework is to aggregate the results from the various metrics described throughout this section.

Specifically, given a set of synthetic datasets, and a set of metrics, we compute a ranking of the datasets according to each metric, and use the mean rank over metrics as the overall quality score. The decision to base the aggregation on ranks was made for two reasons. First, some metrics have vastly different scales, so using ranks allows us to translate all metrics to a common scale. Second, unlike linear scaling, using a ranking minimizes the impact of outliers. Depending on the application, the second point may or may not be desirable. It depends upon the answer to the question, 'If a dataset is extremely bad on a single metric but very good on all other metrics, how should it be ranked?' With a ranking approach, the dataset will likely be scored quite well, despite any extreme outliers. Conversely, if a single score was significantly poor, then a linear scaling approach would cause the synthetic dataset to receive a very bad score. Our framework can easily be adapted to various specific applications by adding, removing, or reweighing metrics based on their relevance to the application, as well as changing the aggregation method. In Section 3.4.2, we experiment with using different subsets of the metrics to create datasets tailored to different purposes.

3.4 Ensembles

Recent work on GAN-based generative modeling has shown that ensembles of GANs tend to be better data generators than non-ensembled GANs [174, 162, 56, 101, 38]. There are various ways in which GAN ensembles can be implemented. The most straightforward approach is to train multiple copies of the same model, using different random seeds [174]. Other approaches include bagging [38], boosting [162, 56], and training multiple copies of a model on different subsets of the training data [174, 38]. In the context of GANs, experimental evidence suggests that the optimal number of models to include in an ensemble depends on the problem and model design [38]. Inspired by this success, we conduct a series of experiments using various methods to

construct ensembles of BF models, with the aim of improving our synthetic data.

In these experiments, we adopt a similar approach as [174], and build our ensembles with multiple copies of Banksformer, which were trained using different random seeds. Specifically, we begin by training 30 randomly initialized copies of BF, and then use each trained model to generate a *single-model* synthetic dataset of 10554 sequences. We create ensemble datasets by combining sequences from two or more single-model datasets. To generate a synthetic dataset of N sequences from an ensemble of k models, we select $\lfloor \frac{N}{k} \rfloor$ sequences from each model's dataset, and if needed select an additional sequence from $N - k \cdot \lfloor \frac{N}{k} \rfloor$ randomly selected models. Using this framework for ensemble dataset construction, we experiment with different *selection criteria* for selecting which k models to include. As a baseline, we consider random selection, where we select which models to include in the ensemble randomly. This random selection criteria is logically equivalent to using no selection, as we simply create the ensemble from k randomly chosen models. In Section 3.4.2, we also consider six *elitist selection criteria*, which select the k models deterministically, according to different criteria.

3.4.1 Number of models

First, we are interested in the impact of the ensemble size on synthetic data quality. After some preliminary experiments, we decided to focus on ensembles containing between 2 to 20 models, as considering bigger ensembles did not lead to improvements. To study the impact of the number of models, we randomly created and evaluated 10 ensemble datasets for each ensemble size in this range, according to the random selection criteria. We compare these ensemble datasets with the 30 single model datasets.

In Figure 3.6, we show the overall impact of the number of models included in the ensemble. The results in Figure 3.6 are somewhat mixed. If we look at the overall mean rank across metrics, there doesn't appear to be a strong main effect of the number of models. However, there does appear to be some reduction in variance as the number of models increases. This is likely due to the two-step method for constructing ensemble datasets; first selecting the models to include and then selecting the samples. If there are a small number of models which each contribute a large number of samples, then there will be higher variance than if a small number of samples where chosen

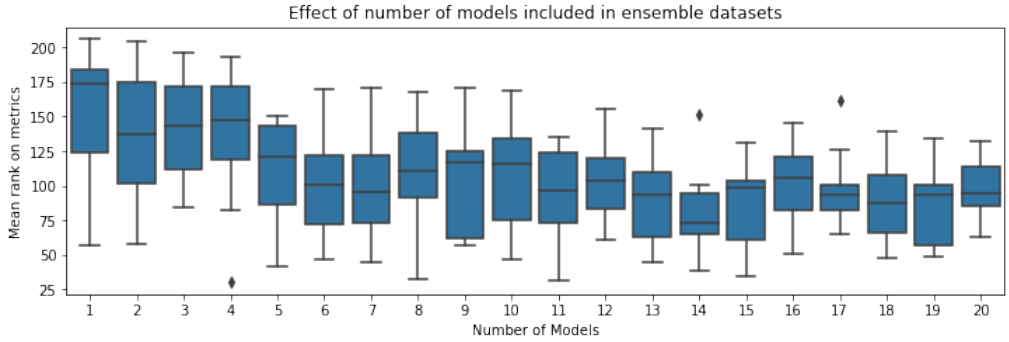

Figure 3.6: Effect of the number of models in ensemble dataset. As the number of models k used to create the ensemble dataset increases, there is a slight improvement in the average quality of a dataset created with a k model ensemble, up to 14. The variance in dataset quality decreases with k. The best dataset was created by a four models ensemble and achieved mean rank of 30.4 across all metrics. The worst dataset was created by a single model ensemble and achieved mean rank of 206.0 across all metrics

from a larger number of models.

Another interesting finding from Figure 3.6 is that the best dataset comes from an ensemble of four models; however on average, datasets produced by larger ensembles scored better than those produced by ensembles of four of fewer models. This led us to consider *elitist* methods for creating ensemble datasets, which select the k models to include according to a *selection criteria*.

3.4.2 Elitist Ensembles

In these experiments, we consider six potential selection criteria for deterministically selecting k of m models ($k < m$) to include in ensembles. The first criterion we considered is to choose the k models which achived the best validation loss during training, as it is a generic indicator of the model quality and not tied to any particular metric. We also consider the overall quality metric, which was defined in Section 3.3.7. While these two criteria are meant to be generic, we also consider four specialized criteria, which aggregate different subsets of related metrics. In Table 3.2, we detail the individual metrics included in each criteria.

In Figure 3.7, we show the interaction between the selection criteria and the number of models included in the ensemble. This figure shows ensembles constructed

Elitist Criteria	Included Metrics
Distribution	amount-wasser, td-wasser, tcode-jsd, tcode-3g-jsd, day-jsd, tcode_day-jsd
Classifier	RF_auc, DT-1_auc, DT-2_auc, LR_auc, KN3_auc, KN7_auc
Coverage	cover3, cover5
Holdout Metric	holdout_val

Table 3.2: Specialized Elitist Selection Criteria. This table describes the set of individual metrics used in each of the elitist selection criteria.

using the mean rank on classifier metrics, or the mean rank on all metrics tend to be the best. Again, there does not appear to be an obvious optimal number of models to include; however, datasets constructed from more than 12 models tend to outperform those with 12 of fewer models.

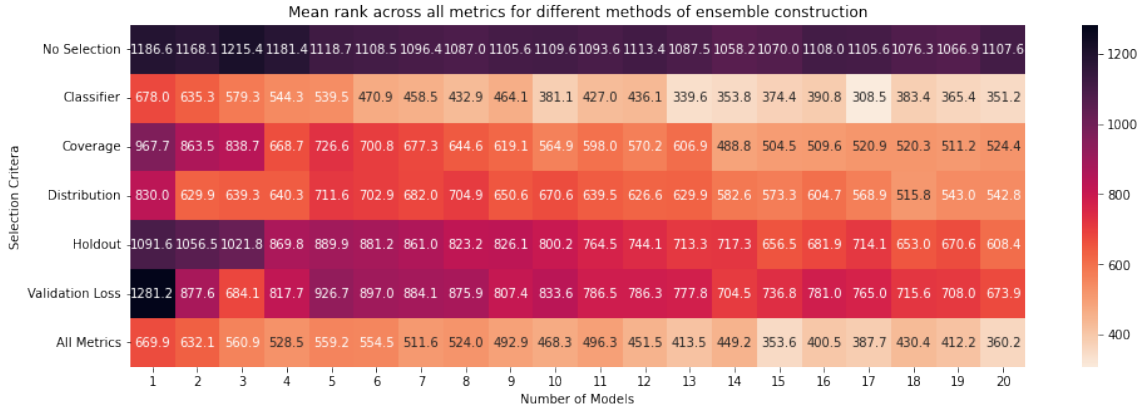

Figure 3.7: Effect of selection criteria and number of models on mean rank. Using the classifier selection criteria with a 17 model ensemble produced the best results, achieving a mean rank of 308.5.

One limitation of Figure 3.7 is that we only consider the overall data quality when comparing methods. This raises a question: does selecting models based on their performance on a subset of metrics (such as classification metrics), create ensembles that will perform well on those metrics? We investigate this in Figure 3.8, where we visualize the the impact of selection criteria on different metric subsets. Overall, the classifier-based selection produces the best data judged by all metrics, which is consistent with what we saw in Figure 3.7. Unsurprisingly, for each of the specialized metric sets (Classifier, Coverage, Distribution, Holdout), the best ensemble dataset

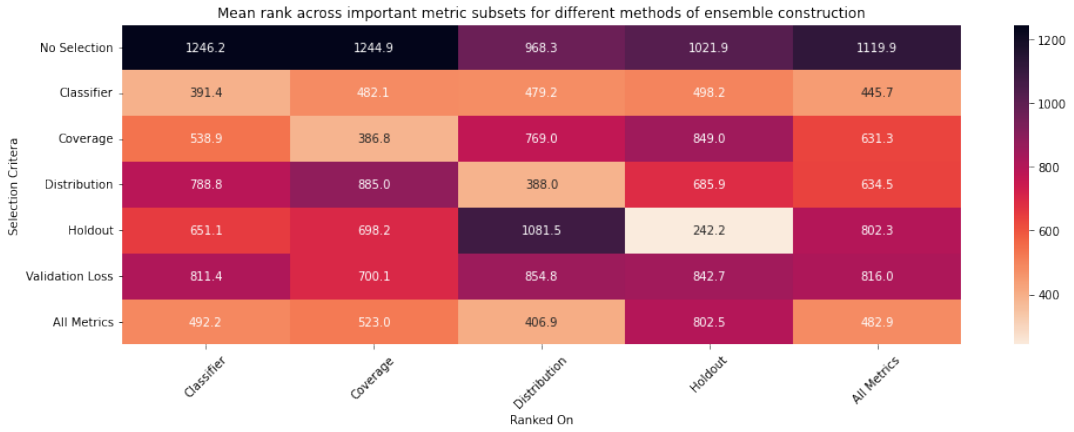

Figure 3.8: Effect of selection criteria and on metric subsets. Each column shows the quality score computed on a different subset of the metrics, while each row shows results for different ensemble selection criteria. Overall, the classifier based selection criteria performs quite well across a broad range of metric subsets. However, when judged using specific metric subsets, such as coverage or distribution metrics, using the corresponding selection criteria produces the best results.

according to that set was created using the corresponding selection criteria. Interestingly, classifier-based selection also performs quite favorably across most metric subsets, and is the best or second-best on all metric subsets besides the distribution metrics. The ensembles created randomly, using no selection criteria, performed worse than ensembles created with any of the other selection criteria, which is also consistent with Figure 3.7.

3.4.3 Comparison to previous results

Throughout this section, we have explored various methods for constructing BF ensembles, with the goal improving the quality of the synthetic banking data we generate. Table 3.3 compares the best datasets from this chapter against the best datasets produced by Banksformer and DoppelGANger in Chapter 2. In Table 3.3 we see the data produced by ensemble methods is better than the best Banksformer dataset from Chapter 2, across all metrics. Table 3.3 also shows that while both elitist and non-elitist ensembles improve on prior results, the elitist ensemble performs best on significantly more metrics. However, none of the Banksformer based approaches were able to outperform DoppelGANger with respect the *amount-wasser* metric. This may

suggest that DoppelGANger is best for modeling single features without dependencies, as the *amount-wasser* metric is based on the *amount* feature and is independent of other features.

Table 3.3: Quality comparison of synthetic banking data generated by various methods. The first two columns show the comparison between data generated by DoppelGANger (DG) [93], and the first iteration of Banksformer (BF), which was detailed in Chapter 2. The 3rd column shows results from the best ensemble model, created by our experiments in this chapter.
*The holdout metric was not computed for the DG and BF datasets, as we do not have information on the training/validation split used for these earlier models.

	DG	BF	ensemble-r	ensemble-e
amount-wasser	**1544**	3004	2622	2495
td-wasser	5.275	0.8514	0.7261	**0.6649**
tcode-jsd	0.0103	0.0116	**0.0016**	0.0019
day-jsd	0.1614	0.0159	0.0052	**0.0043**
tcode_day-jsd	0.4111	0.0435	0.0164	**0.0153**
RF_auc	1.0	0.993	0.9287	**0.9023**
DT-1_auc	1.0	0.6483	0.5988	**0.5808**
LR_auc	1.0	0.8197	0.7499	**0.7059**
KN7_auc	1.0	0.9474	0.8645	**0.7974**
lstm_auc	0.9998	0.9932	0.9786	**0.8667**
tcode_num_consist	0.2035	0.1991	0.1461	**0.1335**
log_amount_sc_consist	0.6914	0.3528	0.3235	**0.2961**
cover3	1.0	0.8402	0.6128	**0.5522**
cover5	1.0	0.8466	0.7014	**0.6716**
lda	0.2081	0.13	**0.0475**	0.0594
tcode-3g-jsd	0.19	0.0954	**0.0186**	0.0194
holdout_val	NA*	NA*	2.031	**0.5254**

3.5 Privacy and Synthetic Data

One of the major motivations for working on synthetic data is to facilitate data sharing while minimizing the risks of exposing private information. It might appear that if we share synthetic data which are not tied to any specific customer, then we do not need to worry about exposing private information about the customers; however, this is not the case. In fact, there are many historical examples (such as [95, 109, 12]) of

'anonymized' data being released and then later being used by others to infer private information about the original data. There are many different formal definitions of privacy, such as: ϵ-identifiability [184], k-anonymity [12], and (ϵ, δ)-differential privacy [36].

Differential privacy (DP) is a common privacy criteria within the machine learning community. As opposed to many other notions of privacy, (ϵ, δ)-DP is a property of a randomized algorithm, not only a dataset. There are various ways (ϵ, δ)-DP can be used to create private synthetic data, however, methods generally follow the same procedure. First, learn an approximate model of the data using an (ϵ, δ)-DP algorithm, then use the approximate model to generate synthetic data. One useful property of (ϵ, δ)-DP is that it is robust to post-processing, meaning that any data generated from a model trained with a method that is (ϵ, δ)-DP, will also be (ϵ, δ)-DP.

Intuitively, an algorithm is (ϵ, δ)-DP if and only if there is a high probability that changing any single record in the input data will not significantly change the algorithm's output. The parameter ϵ indicates the strength of the guarantee to 'not significantly change the algorithm's output', and δ indicates the probability that this guarantee will be violated. Formally, a randomized algorithm A which maps datasets to an output space O is (ϵ, δ)-DP if and only if, for any pair of input datasets D_1, D_2 which differ on only a single sample:

$$\forall S \in O, p(A(D_1) \in S) \leq e^{\epsilon} p(A(D_2) \in S) + \delta$$

.

3.5.1 Differentially Private Banksformer

In this section, we create a differentially private version of Banksformer, by using a modified training algorithm, based on a differentially private version of stochastic gradient descent (DP-SGD) [1]. Essentially, this approach modifies the training procedure by using gradient clipping to limit the sensitivity of the gradients and adding noise to the gradients.

DP-SGD has many parameters which affect both the privacy guarantee and the algorithm's performance, including the amount of noise added to gradients, the strength

of gradient clipping, the batch size, and the total number of batches used during training. Generally, it is expected that adding stronger privacy guarantees will cause some degradation in performance [1, 8, 67], however in some cases adding more privacy may act as a regularizer and improve performance [123]. For a specific privacy level, there are trade-offs between the other parameters that impact the algorithm's performance. Our original goal was to study the trade-offs between these other parameters and the trade-off between stronger privacy guarantees and the quality of synthetic data.

3.5.2 Privacy results

After extensive experimentation with DP-SGD, and other similar tools provided by the TensorFlow Privacy library [53], we found that adding even an extremely weak privacy guarantee caused a substantial degradation in the quality of the synthetic data produced. Typically, values of $\epsilon > 10$ are not considered to offer significant privacy protection. We present results of a $(10^{19}, 10^{-5})$-DP version of Banksformer, that is, with $\epsilon = 10^{19}$ and $\delta = 10^{-5}$. While $\delta = 10^{-5}$ is an appropriate setting, choosing $\epsilon = 10^{19}$ essentially offers zero privacy guarantee. We will demonstrate that even this absurdly low level of privacy protection significantly degrades the quality of the synthetic data generated. In our experiments, we found that with stronger privacy guarantees, the model fails to learn both date-based patterns and appropriate amount distributions.

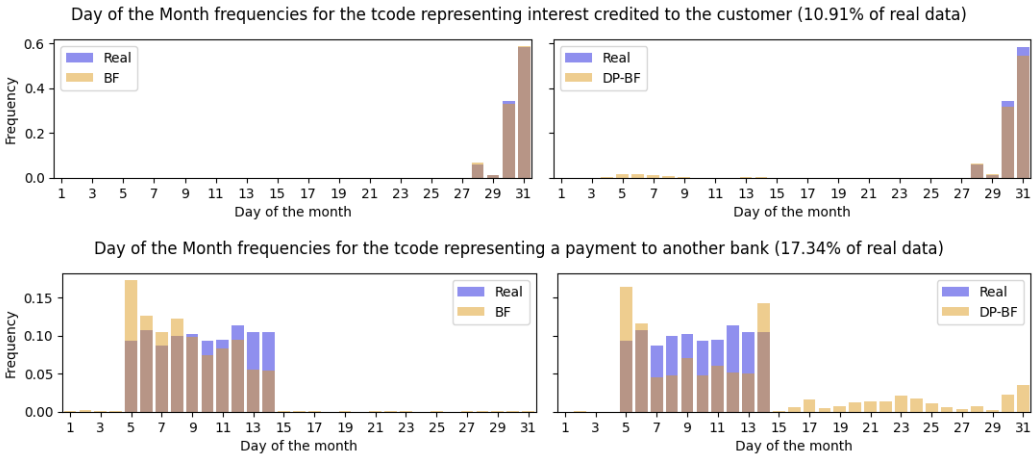

Figure 3.9: Date, tcode relationship in BF and DP-BF data versus real *czech* data. Each row shows a different tcode with a date-based pattern. We include the BF results from the previous chapter on the left side, and show DP-BF results on the right.

Figure 3.9 shows a comparison of the day of the month frequencies for two tcodes with date-based patterns. For reference, we also include the BF results from Figure 2.5 of the previous chapter. From Figure 3.9, we see that while our DP-BF model is able to learn date-based patterns significantly better than the GAN models considered in Chapter 2, it is significantly worse than the non-private BF. The results for the tcode in the top row are similar between BF and DP-BF, however the DP model creates more transactions on the 5th-8th days of the month than it should. The DP-BF model makes further errors on the tcode shown in the second row, producing a significant proportion of these transactions after the 15th of the month, which does not occur in real data or data generated by the non-private BF.

Figure 3.10 shows the synthetic data contains some transactions with amount values orders of magnitude larger than are appropriate based on their tcode. If we compare this to the result from the previous chapter (Figure 2.6), we see that DP-BF is significantly worse than the GAN models, as well as the basic BF implementation.

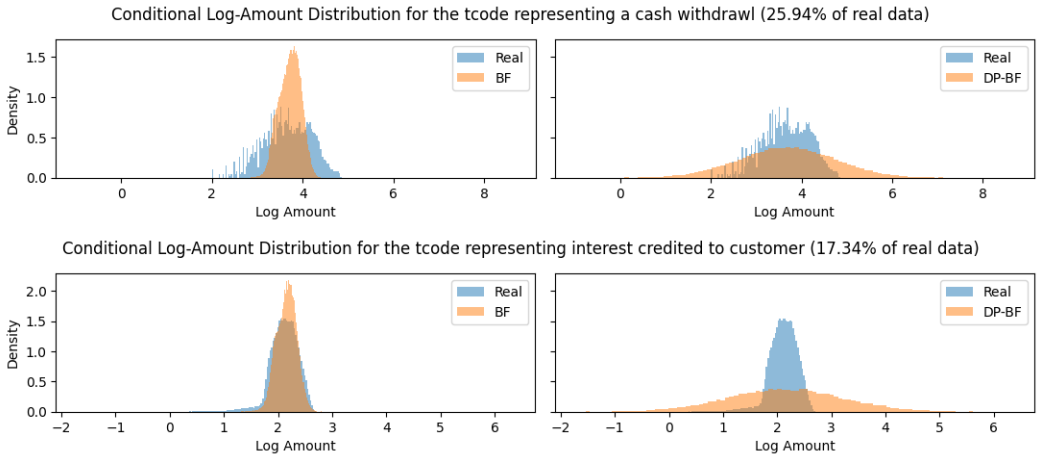

Figure 3.10: Conditional amount distributions for DP-BF. This figure shows a comparison of the conditional distributions $p(\log(amount)|tcode)$ produced by BF against real data, for the two most common tcodes in the real data.

These results with DP-SGD are in line with previous research [93], which also found that generative models trained with DP-SGD failed to capture important data patterns in time-series data. Given that using $\epsilon = 10^{19}$ yields a noticeable degradation in synthetic data quality, and this provides virtually no privacy protection, we conclude that any level of usable privacy ($\epsilon \approx 10$) would make the generated data unusable in practice. This is further supported by our preliminary experiments with

lower ϵ values which yielded very poor quality synthetic data. However, experiments with other types of data have found that for particular datasets, such as MNIST [83], it is possible to achieve a reasonable level of privacy and synthetic data quality simultaneously [181].

3.6 Discussion

The tools we developed in this chapter have allowed us to better understand the quality of our synthetic data, both through qualitative visualizations and quantitative metrics. By visualizing account behavior in Figures 3.2 and 3.3, we found that overall the behaviors are similar between real and synthetic accounts; however, synthetic accounts tend to have more unique types of transactions than real accounts. The additional metrics were then used in Section 3.4, to evaluate different approaches to ensemble construction.

Results from Section 3.4 demonstrate that adopting an ensemble approach with our Banksformer model produces better synthetic data than relying on a single model. Further, in Section 3.4.2, we investigated different methodologies for selecting which models to include when creating ensembles. We found that selecting models based on their performance on classifier-based metrics lead to the best quality data overall. However, our results from Figure 3.8 also suggest that if a certain facet of data quality is particularly important, such as the feature distributions being accurate, then selecting models based on that facet can lead to better results.

The key takeaway from our privacy experiments is that adding even a minute amount of privacy to BF via DP-SGD severely corrupts the quality of generated data. However, our analysis in Section 3.3.5 does suggest that we are not memorizing training data, as the synthetic data we produced was not significantly more similar to the data used to train BF than it was to held-out data, which was not used in training. While this is not a substitute for a formal privacy guarantee, it is still an encouraging finding. Further, throughout our experiments, we have gained a better understanding of the limitations of DP-SGD, and have identified some potential directions of future research, which could add privacy to Banksformer.

One promising future direction would be to modify the DP-SGD framework to allow users to designate certain aspects of the training data as *public information*,

which is not privacy sensitive. For instance, in the *czech* banking data, account fees are always paid on the last day of the month, and over 90% of transactions corresponding to account fees have an amount value of $14.60. If we could designate this information as non-sensitive, we may be able to learn these patterns better and under stronger privacy guarantees. Our work is also limited by data availability; the *czech* dataset only contains 4500 accounts. Because DP quantifies the impact of changing a single training sample, increasing the amount of training data would likely improve results.

Another potential direction is to explore other methods for creating differentially private data without DP-SGD, such as private aggregation of teacher ensembles (PATE) [123], which has been successfully applied in the context of GANs [67]. The PATE approach requires partitioning the training data into k disjoint subsets, training a unique *teacher* model on each subset, and then training a final *student* model by making differentially private queries to the teacher models. We did not explore this method extensively due to the small size of our dataset. We conducted some preliminary experiments training copies of BF on disjoint subsets of the training data. However, we found that with even two subsets, the reduction in training data led to significantly worse synthetic data being produced, before adding any privacy mechanism. While PATE was not useful in our work, it is a promising direction for researchers with access to larger training datasets.

3.7 Conclusion

Throughout this chapter we introduced novel qualitative and quantitative approaches for evaluating synthetic banking data, and in Section 3.3.5 we proposed a flexible framework for evaluating the overall quality of the synthetic data we produce. Using this framework, we explored multiple approaches to ensemble construction, and determined that generally elitist approaches to ensemble construction outperformed random ensembling. While there are different notions of what constitutes 'high quality' synthetic data, we demonstrate that for a wide range of quality metrics, we can create ensemble datasets that perform well on these metrics by using the metrics to pre-filter which models to include in the ensemble. Finally, we demonstrated that adopting an ensemble approach allows us to generate better synthetic banking data

than our best results from Chapter 2.

Chapter 4

Creating Diverse Ensembles for Classification with Genetic Programming and Neuro-MAP-Elites

In this chapter, we explore evolutionary computing methods for constructing diverse ensembles. Our focus in this chapter is on classification tasks, as opposed to synthetic data generation, as ensembles have been better studied in the context of classification. In Chapter 5, we employ a diversifying evolutionary algorithm based on this work to further boost the quality of synthetic banking data we create.

This work was published and presented at the 2022 EuroGP conference, a CORE-B ranked conference in the field of evolutionary computing [113]. It built upon the work, which was the basis for a short paper and poster presented at the 2021 Genetic and Evolutionary Computation Conference, a CORE-A ranked conference in the field of evolutionary computing [112].

4.1 Introduction

Ensemble classifiers, which make predictions by combining multiple simple models, outperform single model classifiers for many supervised learning tasks. The importance of diversity in ensemble classifiers is well documented, however in general it can be challenging to create diverse ensemble classifiers in a principled way. While there may be various notions of diversity, for ensemble classifiers it is specifically *error diversity* - meaning that individual classifiers make errors on different samples - which matters most [20].

To see the benefit of error diversity, consider a toy problem consisting of only three samples and a set of limited classifiers, each of which can only classify two of the three samples. In this scenario, we can construct an optimal ensemble from three limited classifiers, as long as they are maximally diverse with respect to the errors they make. Conversely, if even two of the three classifiers make the same error, then the ensemble will not be correct in all cases (see Figure 4.1).

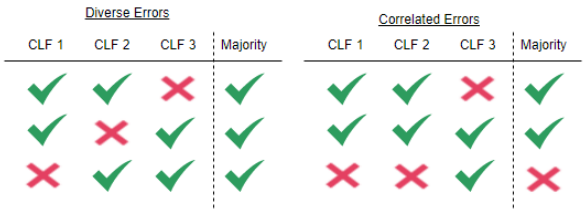

Figure 4.1: Benefit of ensemble diversity. In this toy example, each classifier can only achieve a maximum accuracy of 67%. However, as long as there is diversity in the errors made, a majority vote classifier constructed from these imperfect classifiers will achieve perfect accuracy.

The fact that evolutionary algorithms (EAs), such as genetic programming, produce populations of solutions makes them seemingly ideal for ensemble creation. However, many standard EAs for evolving classifiers lose diversity as evolution proceeds, causing the predictions of the individuals in the final population to be strongly correlated. This limits our ability to create an effective ensemble classifier directly from a final population. In practice, we can run an EA many times and create a diverse ensemble by selecting the best individual from each run to be in the final ensemble, however there are drawbacks to this method. In addition to being wasteful by not using all models from each run, there are other issues with this approach. For example,

it is possible that if the standard algorithm is run too long that independent trails will converge to the same or similar solutions, and if it is not run long enough, the individuals will not be fit enough to be useful in an ensemble.

Within the field of evolutionary computing, there are many examples of the utility of creating diverse sets of solutions as opposed to focusing on a single best solution, such as evolving adaptable robot controllers [29], playing card games [23, 42], and generating video game content [55]. Many of these results rely specifically on the MAP-Elites algorithm, a niching EA, which represents the population as a discrete grid of cells and assigns solutions to cells based on their behaviors [108].

The MAP-Elites algorithm essentially provides a structured population, which aims to increase population diversity in a user defined *behaviour space*. In MAP-Elites, candidate solutions only compete with other candidates assigned to the same cell, which allows the population to maintain diversity as evolution proceeds. The mapping from solutions to cells is facilitated by *behavior descriptors*, which maps solutions to a low dimensional behavior space. The behavior space is then partitioned into cells, and solutions are assigned to the cells in which their behavior descriptors lie. This partitioning of the behavior space helps prevent MAP-Elites populations from converging to a set of highly similar solutions by preserving diversity in the behavior space [108].

Defining effective behavior descriptors is an essential aspect of MAP-Elites, as they determine the sorts of diversity which will be produced. As mentioned above, when creating ensemble classifiers, we are particularly concerned about diversity with respect to the classification errors made. Given a fixed set of samples to be classified, error information can be represented naturally as a high dimensional binary indicator vector, with length equal to the number of samples. Our method relies on a variational autoencoder (VAE) to learn a low-dimensional representation of the high dimensional error vectors.

The main contribution of this work is Neuro-MAP-Elites (NME), a novel framework based on MAP-Elites, for evolving ensembles of classifiers with greater error diversity. Using linear genetic programs (LGPs) as our classification model, we show that NME can be used to create diverse populations of classifiers and further that these more diverse populations make more effective ensemble classifiers.

4.2 Background

In this section, we provide some brief background on the techniques upon which NME is based.

4.2.1 Linear Genetic Programming

In general, genetic programming (GP) seeks to evolve computational models for making predictions [18]. There are many variants of GP, based on different representations for the computational models being evolved, such as tree GP [77], Cartesian GP [104] and linear GP (LGP) [18]. In this work, we use LGP for our genetic programming model, as it provides good performance on a wide range of classification tasks [18, 147]; however, any GP variant can be used with our method. In this section, we give a general overview of LGP, and in Section 4.3, we provide more specific details on the specific LGP implementation used in this work.

Simply put, LGP is a type of GP where computational models (i.e., programs) are represented as sequences of instructions, often resembling imperative programs [18]. Generally, programs have access to a number ($n_{registers}$) of writable *computation registers* used when executing their instruction sequences. Programs also have some registers designated as *input registers*, which, as their name suggests, contain input values to the program. In the context of classification tasks, these input values are the features of the samples that we are classifying. The input registers may be implemented by designating a subset of the writable computation registers as input registers or as a separate set of read-only registers. Instructions typically involve mathematical operations on values stored in registers and result in either information being written to registers, or changes to the program execution (such as skipping instructions). Programs perform classification of samples as follows: first, the samples' features are loaded into the input registers, next the instructions are executed, and finally the final state of the computation registers is then transformed to a prediction. The transformation of the final state of the computation registers may be done in many ways and is dependent on the problem. In the simple case of binary classification tasks, a common approach is to designate a single computation register as the *output register*, and then compare the value of the final register to a predefined threshold to obtain the prediction (i.e., if $output > threshold$ predict class 1, else

predict class 2). In this work, we combine LGP with the MAP-Elites algorithm to help create diverse populations.

4.2.2 Map-Elites

Map-Elites is an EA designed to evolve diverse populations of solutions in a single run [108]. More specifically, Map-Elites falls under the umbrella of *quality diversity* algorithms, which are EAs designed to evolve populations of high-performing solutions which are also diverse. The way Map-Elites works is fairly straightforward – instead of using an unstructured population, as is typical in EAs, MAP-Elites used a structured grid population to maintain diversity. The structured grid contains a number of cells equal to the maximum population size. Each cell begins empty and may contain no more than one solution during the execution of the algorithm. When a new candidate solution is created, there is a two-step process to determine if the candidate solution is added to the population. First, a *behavior function* maps the candidate solution to a single cell. The candidate solution then competes with the current solution in this cell, or if no solution currently occupies the cell, the candidate solution is automatically added. The simplest way to implement the competition is by simply selecting the solution with the highest fitness and breaking ties randomly.

An essential part of MAP-Elites is determining the mapping from solutions to cells. To this end, when evaluating a possible solution, a *behavior descriptor* is produced, in addition to a fitness score. The *behavior space* is a low-dimensional vector, typically 2D, used to represent a solution in the *behavior space*. The behavior space is partitioned into (usually equal sized) cells so that a program is assigned to the cell in the behavior space in which its descriptor lies. The reason for using low-dimensional descriptors is that higher dimensional descriptors lead to problems stemming from the curse of dimensionality. Specifically, the problem is that if we divide each dimension into k regions (boundaries for the cell), then the total number of cells for a d-dimensional descriptor will be k^d. If we instead try to fix the max population size at N, then there can only be $\lfloor \log_d N \rfloor$ regions per dimension. In practice, we typically want at least ten regions per dimension, so we must stick to low-dimensional descriptors or use other variants of MAP-Elites, such as MAP-Elites-CVT [170].

4.2.3 Variational Autoencoders (VAEs)

VAEs are a powerful probabilistic modeling framework for representation learning [74]. The VAE framework assumes that high dimensional observed data are generated by a random process acting on unobserved latent factors. VAEs are composed of two feed-forward neural net models, often called the encoder and decoder networks. The encoder network learns to infer a distribution over low-dimensional representations of the high dimensional data, and the decoder learns a generative model from latent factors to the samples from the original high dimensional space.

To train a VAE, the weights of both the encoder and decoder are optimized together, using an unsupervised objective. During training, the high-dimensional original data x are passed through the encoder, which outputs a probability distribution over encodings of x. Specifically, this is done by outputting a mean vector μ_x, and variance vector σ_x, which are interpreted as parameters to a Normal distribution with a diagonal only co-variance matrix. A sample is then drawn from this distribution, and the sample is fed into the decoder, which aims to reconstruct the original sample x by outputting $\hat{x}$. The loss function used to train a standard VAE is known as the evidence lower bound (ELBO), and contains two terms. The first is the reconstruction error $||x - \hat{x}||^2$, which measures the reconstruction quality. In the loss function, this term incentivizes the VAE towards learning low-dimensional encodings, which are informative about the high dimensional representation. The other term is the KL divergence between the distribution output by the encoder, and a standard Normal distribution $D_{KL}(N(\mu_x, \sigma_x)||N(0, \mathrm{I}))$. This term encourages representations that are approximately normally distributed.

While it may seem beneficial to only focus on the first objective — which is essentially what is done in basic autoencoders [62] — the second objective provides regularization and encourages the encoder to learn smoother encodings. This means small changes in the latent features produce small changes in the high dimensional space, and similar high dimensional vectors are encoded to nearby locations in the low dimensional space. Further, it also produces latent spaces in which the samples are approximately normally distributed, which is useful for some downstream tasks.

There have been many modifications made to VAEs since their original proposal. Here we briefly mention a VAE variant called β-VAE [61], which is used in this work. β-VAE follows the original VAE design but employs a modified version of the ELBO

loss function. In β-VAE, the loss contains a constant β, which controls the relative weight of the two terms in the standard VAE loss. The benefit of β-VAE is that it allows us to control the trade-off between the two terms in the ELBO. Specifically, the β-VAE loss function is: $loss = ||x - \hat{x}||^2 + \beta D_{KL}(N(\mu_x, \sigma_x)||N(0, I))$.

4.2.4 Ensemble Classifiers

Traditionally, GP approaches to classification rely on outputting the single best program as the final predictive model. However, it has been argued that since these algorithms produce populations of programs that are all adapted to the target task, it is logical to make use of the entire population in the final model [47, 140]. One major issue which must be addressed when evolving ensembles is the maintenance of diversity amongst the individual solutions, as predictive diversity has been shown to be crucial in ensemble creation [20].

In work from the GP community on evolving ensembles, there are generally two approaches; *offline* approaches, which construct the ensemble directly from a final population, and *online* approaches, which gradually build the ensemble during evolution [47]. With offline approaches, it is necessary to actively design the population structure to encourage diversity and prevent convergence to a population of identical or similar individuals. In online approaches, while not strictly necessary, diversity preserving populations can be beneficial [140].

More recently, others have used EAs with explicit diversity mechanisms to create ensemble classifiers. Boisvert and Sheppard [16] use an approach based on novelty search [84], to evolve diverse ensembles of decision trees. In this work, the population is represented as a variable-sized *archive*, and new candidate solutions are added to the archive if they meet a criterion based on novelty and fitness. Cardoso et al. [25] also used novelty search in the space of neural network architectures to create diverse ensembles. In their work, the novelty metrics were based explicitly on error diversity. To the best of our knowledge, our work is the first to apply MAP-Elites and GP to create ensemble diversity.

4.3 Our LGP implementation

In this section we describe the specific details of the LGP implementation used in this work. In our implementation of LGP, all instructions have the format

$$destination = source1 \ < op > \ source2$$

where op is one of the functions in the operator set {ADD, SUBT, MULT, PDIV, SNIG, QUIT}, and *destination*, *source*1, *source*2 refer to registers which store floating point values (see Table 4.1 for information on the operators).

Table 4.1: Operators from LGP system. *dest* refers the destination register and $src1$, $src2$ refer to source registers.

Operators	Description
ADD	$dest = src1 + src2$
SUBT	$dest = src1 - src2$
MULT	$dest = src1 * src2$
PDIV	if $scr1 \neq 0$ $dest = src1/src2$
	else $dest = src1$
SNIG	if $src1 > src2$, skip next instruction
	(skip next if greater)
QUIT	do not execute any more instructions

In our system, all registers are writable, including the input registers. Programs have access to a total of $10+n_{features}$ registers, where $n_{features}$ is the number of features in the dataset. This setting was determined empirically and worked well with the variety of datasets we tested. Others have advocated using a constant multiple of the number of features [18], however we found that for datasets with many features, this method provides too many registers, and evolution takes much longer to find good solutions. For initialization, the first register is set to 0.0, the following nine are set to constant values, and the remaining $n_{features}$ registers are set to the feature values of the sample which is being classified. Any binary features are represented using 1.0 to represent true and 0.0 for false. A program's prediction is based on the final value of the first register; values greater than 0 are interpreted as a prediction that the class label is "1", values less than 0 are interpreted as a prediction that the class label is "0". Programs that end execution with 0 in the first register are interpreted

as not having made a prediction; when computing a program's fitness, no prediction is always scored as an incorrect prediction.

Variation

In our LGP algorithm, we produce variation in programs through both micro and macro mutations. Our macro mutation operator replaces a randomly selected instruction with a new randomly generated instruction, and our micro mutation operator randomly changes either the operator, destination register, or one of the source registers of a randomly selected operation.

When generating random instructions, new instructions obey the following rules, which allows for efficient evolution and ensures all mutations result in legal programs. The first rule simply ensures all instructions refer to valid registers and operators by enforcing bounds on their range. This second rule is that when generating a random value for a destination register of a program, the maximum value is one greater than the current maximum value of a destination register in that program. This rule is inspired by the progressive complexification proposed in NEAT [152] and is designed to encourage mutations to have a higher chance of being effective.

Table 4.2: Rules for random instruction creation. Rule 1 ensures that generated instructions are always legal and meaningful. Rule 2 encourages mutations to have a higher change of being effective. In rule 2, max_dest refers to the largest integer representing a destination register in the program for which the instruction is being generated, and is taken to be 0 in the case of a program with no instructions.

Rule	Description
Rule 1	$op \in \{0, 1, ..., 5\}$
	$src1, src2, dest \in \{0, 1, ..., 9 + n_{features}\}$
Rule 2	$dest \in \{0, ..., max\_dest + 1\}$

Fitness

A program's fitness is its balanced accuracy score on the training data, which is computed from the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN): $fitness = \frac{0.5TP}{TP+FN} + \frac{0.5TN}{TN+FP}$. The balanced accuracy

score provides a good dataset agnostic measure of the accuracy of a classifier, as it is normalized to account for imbalanced classes [118].

Behaviors

To use MAP-Elites with LGP, we must define suitable behavior descriptors for the LGP programs. Previously there has been only limited work using MAP-Elites together with LGP, such as [33]. In our experiments using basic MAP-Elites with LGP, we use three behavior descriptors to categorize program behaviors: the number of features, instructions, and registers used by the program. The number of instructions counts only instructions that affect the final output of the program, and the number of registers counts the number of unique destination registers used in effective instructions, both of which of indicative of a programs complexity (see Table 4.3 for information on the descriptors used with each variant).

Table 4.3: Behavior descriptors used with each variant of MAP-Elites.

ME Type	# of features	# of registers	# of instructions	VAE encoder
Basic ME 0	X	X		
Basic ME 1		X	X	
Basic ME 2	X		X	
Neuro ME				X

4.4 Neuro MAP-Elites

This section details the Neuro MAP-Elites (NME) algorithm, an offline approach to evolving diverse GP populations, which can be used as accurate predictive ensembles. We begin by outlining an ideal high-dimensional behavior descriptor appropriate for any classifier, particularly linear genetic programs. We then propose a low-dimensional approximation that can be used with MAP-Elites to create accurate ensemble classifiers.

The primary motivation behind our behavior descriptors is that classifiers are designed to make predictions on samples coming from some distribution. When we describe the behavior of a classifier, what really matters is how it behaves on samples

from this target distribution. This is similar to the idea of program *semantics*, which has previously been studied in the context of GP [10, 11, 78, 106, 111]. The main innovation in this work is proposing a methodology for creating a low-dimensional encoding that captures this notion of behavior.

If we consider a simplified case in which the target distribution is uniform over a finite dataset, we can create a high dimensional descriptor of a classifier behavior by recording its predictions on all samples. In this case, the high dimensional descriptor gives a complete description of the classifier behavior. The question of whether it is possible to obtain a complete low dimensional description of the behavior is equivalent to the question of whether it is possible to compress the high dimensional descriptors to a fixed low dimension without loss - which in general is not possible. This means that for any low dimensional behavior descriptor, some programs with different prediction behaviors will be mapped to the same point in the low dimensional space. The goal of our method is to provide a low dimensional approximation to the ideal high dimensional descriptor. To this end, we employ a VAE which learns to compress the high dimensional ideal descriptors into low dimensional approximations.

The proposed NME algorithm can be divided into 3 phases; mining solutions, VAE training, and a final MAP-Elites run using the encoder to generate behavior descriptors (see Figure 4.2 for an overview of the algorithm.)

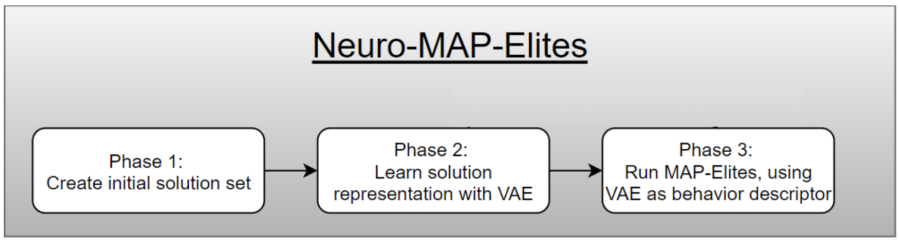

Figure 4.2: Overview of Neuro-MAP-Elites. (Phase 1) Create an archive of classifiers and record their predictions on training samples. In our implementation, we create this archive by running MAP-Elites multiple times using various simple descriptors. (Phase 2) A VAE is trained to compress the prediction vectors (inputs) to a two-dimensional descriptor (outputs) which can be used to map programs to cells in MAP-Elites. (Phase 3) Run MAP-Elites, using the encoder network to produce behavior descriptors when mapping programs to cells.

4.4.1 Mine Solutions

The goal of the first phase of NME is to draw a sample from the distribution of errors produced by 'good' programs, which can be used in phase two to train the VAE. In general, and depending on the specific problem, what constitutes a 'good' solution may vary. In this work, we consider any solution which is present in a final population and obtains a balanced accuracy score greater than 0.5 on the training data. To create the sample, we run a basic version of MAP-Elites using various descriptors (Basic ME0, Basic ME1, Basic ME2), and record the predictions made by each individual in the final populations on all training samples.

4.4.2 VAE Training

The goal of the second phase is to learn a low-dimensional representation of samples from the distribution of errors produced by 'good' programs. To this end, we use the prediction vectors from solutions produced in the first phase as training data and train a β-VAE to encode these vectors into a 2-dimensional representation to be used as a behavior descriptor in MAP-Elites.

Initially, when training the VAE, we used cross-validation to select the optimal β values for each dataset from values in the range $\{0.1, 0.2, ..., 1.5\}$. We found for all datasets that the optimal β was in $\{0.2, 0.3, 0.4\}$, so when running final experiments, we reduced the range of β values tested to only those ≤ 1.0. For each β, we repeat fitting the VAE five times, using different random seeds, as we found that the random initialization impacted the VAEs' ability to learn good representations. The VAE that created the highest entropy distribution in the latent space was selected from these five fitting trials. We experimented with using two and three dimensions for the latent space; however, all experiments in this work are based on two dimensions, as we did not find better performance from increasing the latent space to three dimensions.

The VAE that created the highest entropy distribution in the latent space was selected from these five fitting trials.

4.4.3 MAP Elites with encoder

In the final phase, we rerun MAP-Elites, however instead of using the basic descriptors (as in phase 1), we now use the encoder network of the VAE (trained in phase 2) to produce the descriptors. The encoder takes as input a binary prediction vector made by a program on the training data and outputs a 2-dimensional real-valued encoding of the prediction vector. As the encoder was trained as part of a VAE, the distribution of encodings should be approximately a unit normal distribution. We take advantage of this fact when partitioning the latent space into bins to use with MAP-Elites, and set the bin boundaries so that each bin has equal probability mass under a normal distribution.

4.5 Experiment Setup

To test the efficacy of our method, we compared the predictive accuracy of our method against other standard supervised learning techniques across a diverse set of datasets.

4.5.1 Dataset Selection

In this work, we use a subset of datasets from the Penn Machine Learning Benchmarks (PMLB) repository [118] which contains a large collection of curated datasets for machine learning evaluation and comparison. PMLB contains a wide assortment of datasets suitable for various machine learning tasks. In this work, we considered datasets that contain a binary classification task. As our algorithm produces an ensemble classifier, we are particularly interested in how it performs relative to other ensemble classifiers. To this end, we selected the datasets based on the performance of a standard ensemble classifier: random forest [19]. Specifically, we selected two datasets where random forest performs much better than other standard classifiers, two datasets where random forest performs much worse, and finally two datasets where all the tested standard algorithms do poorly (See Table 4.4).

For all classifiers, the datasets were first partitioned into a "full training' and a test set (75%-25% split). The "full training' set is further partitioned into training and validation sets (75%-25% split). The training sets are used to fit the models, the

validation sets are used to determine model hyperparameters, and the test sets are used to compute the final metrics.

Table 4.4: Datasets used for classifier comparison. See [118] for dataset details.

Dataset	Difficult for RF	Difficult for other ML
Breast		X
Monk2		X
HV_without_noise	X	
HV_with_noise	X	
GAMETES_Epistasis	X	X
Parity5+5	X	X

To demonstrate the efficacy of our method, we compare the performance achieved by classifiers generated with NME against the performance of standard machine learning classifiers, as well as genetic programming classifiers evolved using a classic version of MAP-Elites.

4.5.2 Standard Machine Learning Classifiers

All experiments using standard classifiers used the classifier implementations from *scikit-learn* [125], as well as scikit-learn utilities for tuning their parameters[1]. Specifically, we considered the following five standard machine learning classifiers (scikit-learn names in parentheses); random forest (RandomForestClassifier) [19], k-nearest neighbors (KNeighborsClassifier), logistic regression (LogisticRegressionClassifier) [13] , Gaussian naive Bayes (GaussianNB) [59] and support vector machine classifiers (SVC) [128].

4.5.3 Map-Elites Classifiers

For both basic MAP-Elites and NME, we tested two methods of creating a final classifier from the final population. In the first method, the final classifier is simply the single program with the best fitness from the final population. If multiple programs are tied for the best fitness, the program with the fewest effective instructions is

[1] see github.com/BigTuna08/nme for the code to tune parameters of all models

selected (if there is still a tie, the winner is chosen randomly). In the second method, an ensemble is created by combining all programs in the final population with fitness above a threshold t (t is chosen to maximize the accuracy of the majority vote classifier on the training data). The ensemble classifier outputs the prediction of the majority of the programs, with ties broken randomly. We also considered other methods of combining programs to create the final classifier, such as weighting the programs by their fitness, but this did not improve results.

All runs of MAP-Elites used a 20×20 grid for the population and ran for 1 million evaluations.

4.6 Results

In this section we present results comparing NME with the variants of basic MAP-Elites described in Table 4.3. In addition to our results on ensemble accuracy (Section 4.6.3), we show results supporting our methods in Sections 4.6.1 and 4.6.2.

4.6.1 VAE Efficacy

One possible concern with this methodology is how well the learned encoding captures the information from the original high dimensional descriptors. To investigate this, we examine the decoders' ability to reconstruct the original high dimensional descriptor from the 2-dimensional encoding. Averaged across all datasets, the decoders were able to correctly reconstruct over 88% of the predictions, with the reconstruction ability varying from just below 80% to nearly 100% on the various datasets (see Table 4.5).

4.6.2 Diversity Comparison

Here, we compare the predictive diversity of ensembles of LGP programs created by basic variants of MAP-Elites, as well as our proposed NME method. We define the predictive diversity as the average Euclidean distance between prediction vectors within the ensemble. To obtain these results, we conducted 50 independent runs of each MAP-Elites variant (Table 4.3).

The results from comparing predictive diversity (Table 4.6) support our claim

Table 4.5: Accuracy of VAE at reconstructing predictions. Percent scores indicate the percentage of bits in the error vectors correctly reconstructed by the VAE. Higher scores indicate that the learned descriptors are more informative about the ideal high dimensional descriptors.

Dataset	VAE prediction reconstruction
Breast	98.88%
HV_without_noise	90.25%
HV_with_noise	89.66%
Monk2	89.19%
Parity5+5	84.94%
GAMETES_Epistasis	79.67%
Mean	88.76%

that NME produces more diverse populations. We found that across all datasets, the average diversity produced by NME in a single run was greater than both (1) the diversity of an ensemble created by combining all final populations created from the three variants of basic MAP-Elites and (2) the diversity of the single best run of basic MAP-Elites. Further, these results show that for four of the six datasets tested, the least diverse single run of NME was still more diverse than the most diverse run of basic MAP-Elites (Table 4.6).

4.6.3 Ensemble Accuracy

Measured across all datasets, our method compares favorably against both the traditional machine learning classifiers and LGP classifiers evolved using MAP-Elites with basic LGP descriptors.

However, no method is a clear winner across all of the datasets (see Table 4.7). On the two datasets which were selected for being easy for random forest (Breast and Monk2), random forest was the most accurate. On one of the datasets, which was significantly harder for random forest than other standard classifiers (HV_without_noise), multiple methods achieved perfect accuracy, including NME partial vote; on the other (HV_with_noise), multiple standard methods outperform all evolved classifiers. Finally, on the datasets which were difficult for standard methods (Parity5+5 and GAMETES_Epistasis), evolved classifiers outperformed the standard ones, although in

Table 4.6: Comparison of LGP prediction diversity with top 2 scores in bold. Here we show the diversity scores obtained from running basic ME 120 times and combining the final populations (Multi), the highest diversity score obtained in a single run (Best Single), the average diversity score from a single run (Mean Single), and the lowest diversity score from a single run (Worst Single). Across all datasets, NME produced populations with the the most diversity. For many datasets, the least diverse single run of NME was more diverse than all variants using basic MAP-Elites.

	Basic ME			NME	
Dataset	Multi	Best Single	Mean Single	Mean Single	Worst Single
GAMETES	4.67	5.74	3.85	**7.64**	**5.78**
HV_noise	7.62	7.54	6.27	**8.69**	**7.93**
HV	4.83	5.18	4.26	**7.11**	**5.66**
Breast	3.28	**3.74**	2.97	**4.29**	3.58
Monk2	3.51	**4.21**	3.29	**4.64**	4.11
Parity5+5	3.98	4.64	3.76	**6.85**	**5.34**

only one of these (GAMETES_Epistasis) was the NME classifier the best.

Another finding evident from Table 4.7 is that of the methods tested that evolve ensembles, those created with NME significantly outperform ensembles created with the basic variants of MAP-Elites. This is despite the fact that basic MAP-Elites can produce high-quality single solutions, sometimes better than the single solutions produced by NME. This result supports our hypothesis that NME generates populations with more meaningful diversity and that this diversity is beneficial for creating ensembles.

Table 4.7: Comparison of balanced accuracy scores of common machine learning classifiers and evolved classifiers. Best scores for each dataset are indicated in bold.

	Classic ML					MAP-Elites		NME	
Dataset	RF	KNN	LR	GNB	SVC	ME-Best	ME-Vote	NME-Best	NME-Vote
Breast	**.950**	.564	.500	.949	.500	.939	.579	.913	.943
Monk2	**.987**	.709	.447	.456	.697	.729	.502	.642	.785
HV	.646	.637	**1.00**	.523	.949	**1.00**	.426	.986	**1.00**
HV_noise	.575	.520	**.974**	.526	.854	.687	.495	.709	.802
GAMETES	.499	.515	.486	.513	.486	.495	.516	.510	**.569**
Parity5+5	.594	.601	.463	.463	.473	**1.00**	.494	.891	.986
Mean	.709	.591	.645	.572	.660	.808	.502	.775	**.848**

4.7 Discussion

Using representation learning techniques in combination with quality diversity methods provides a promising avenue for creating diverse classifier ensembles. In this work, we have provided a method for extending MAP-Elites for use with high dimensional descriptors by learning low dimensional approximations to the true descriptors. Here, we focused on diversity with respect to the errors made, but there are other possible high dimensional descriptors that could be used with our method.

One in particular, is information about features used by each classifier. In a similar manner as the high dimensional binary error vectors, information about the features used by a classifier can naturally be represented as a binary indicator vector, which could be compressed by a VAE. There are two main reasons why it may be beneficial to use this methodology with feature information; the first is for working with data with missing or noisy features, and the second is to improve our understanding of what features tend to co-occur in effective classifiers. This second use is perhaps the most interesting and relies on the generative model portion of the VAE (sometimes called the decoder), as opposed to the encoder portion used in NME. If we use a simple generative model, such as a single-layer neural network, we can inspect the model to gain insight into the relationships amongst features.

Another possible direction for this work is to explore other methods for compressing the ideal high-dimensional descriptor outlined in this work. The design of VAE models has recently received much attention from the representation learning community, and it is possible that novel architectures may be more suitable for use in our algorithm. In our experiments, we limited the VAE latent dimension to two, as we found that was sufficient for encoding most of the information in the prediction vectors for most datasets. Clearly, this may be limiting, particularly when the number of samples used for creating the prediction vectors is large. In future work, we plan to examine the relationship between the number of samples included in the high dimensional descriptor, the dimensionality of the encoding, and the performance of NME.

One limitation of our current work is NME requires greater computational resources than traditional GP methods, as it requires us to create an initial set of solutions and train a VAE, before evolving the final population with MAP-Elites. In

our current implementation, we create the initial set of solutions by running basic variants of MAP-Elites multiple times, which is a major cause of inefficiency. In future work, we plan to experiment with more efficient ways to create the initial solution set; in particular by using more efficient machine learning classifiers in this step. We also plan to implement a cyclical version of NME, which can continuously alternate between phase two (training VAE) and phase three (MAP-Elites with encoder), until a stopping condition is met. This will hopefully reduce the amount of effort needed in phase 1, as the solution set used to train the VAE will be updated as NME proceeds. By improving the efficiency of NME, we can make it more practically competitive with non-evolutionary classifiers and lead to greater adoption of GP by the larger machine learning community.

Chapter 5

Improving Synthetic Banking Data with MAP-Elites and Banksformer

In this chapter, we return to our core objective of generating high-quality synthetic banking data. Our previous findings in Chapter 3 revealed that ensembles of Banksformer models produce superior data quality when compared to datasets generated by a single model. Chapter 4 explored the idea of using a diversity promoting evolution algorithm to evolve a population of models which can be used as an ensemble. In this chapter, we implement a diversity-promoting method to evolve a diverse ensemble of Banksformer models, which achieves further improvements in data quality.

The contents of this chapter are being prepared for submission to the 2024 EuroGP conference

5.1 Introduction

Over the past decade, there has been an explosion of work on generative models in the machine learning community, which was in part fueled by novel neural network-based generative models, such as variational autoencoders [74] and GANs [52]. More recent generative models, including Transformers [171] and diffusion-based models [141], are currently able to generate high-quality synthetic data in many domains.

Creating systems to automate content generation has long been an interest of the evolutionary computing community. Evolutionary algorithms have been used

successfully in the past to generate art [146], music [63], video game levels [107], and many more types of content. In this work, we combine innovations from evolutionary computing with recent deep learning architecture innovations to create high-quality synthetic banking data.

Banking data is an area where there is a massive amount of data collected; however, due to privacy concerns, banking data is very rarely publicly available. Synthetic data is a potential method for increasing the availability of sensitive data. Generating realistic bank transaction sequences is challenging; there are many different types of customer behavior, and each transaction contains an amount value, a categorical code indicating the type of transaction, and a timestamp. For synthetic data to appear realistic, both the sequence ordering and the fields of each transaction must follow similar patterns as the real data.

In addition to each sequence appearing realistic, good synthetic data should be as diverse as real data from the target distribution. In this chapter, we implement a MAP-Elites [108] based evolutionary algorithm, inspired by the work of the previous chapter, to improve upon the quality of synthetic banking data we generate.

5.2 Synthetic Banking Data

The specific type of banking data we use for our experiments are sequences of transactions from non-business customer accounts. The basis for our synthetic data is a dataset consisting of real bank transactions from the Czech Republic [127], which was also used in Chapters 2 and 3 of this thesis. This dataset contains records of transactions from 4500 accounts, over a period of 5 years, spanning 1993 to 1998. Each transaction record has an amount value, a date, and three categorical fields which provide information about the transaction type.

5.2.1 Banksformer

Throughout this chapter, we use Banksformer [114] as the base model for generating synthetic banking data. Banksformer is based on the recently proposed *transformer* neural network architecture, and has been shown to be state-of-the-art for generating synthetic data based on the *czech* dataset used in this work. The Banksformer model

has many hyper-parameters, which can impact its performance; however, the relationship between hyper-parameters and model performance is complex and non-linear. Because it takes significant computational effort to fit a single copy of Banksformer to a dataset, and the relationship between hyper-parameters and model performance is stochastic, it is not feasible to exhaustively search the hyper-parameter space. Therefore, we adopt an evolutionary approach for parameter tuning, allowing us to explore the parameter space more efficiently. Inspired by our results from the previous chapter, we propose using a diversifying EA to create a high-quality ensemble of BF models. However, in this chapter we aim to diversify with respect to notions of data quality, as opposed to classifier behavior.

5.2.2 Quality of Banking Data

As we have discussed throughout the previous chapters, it is difficult to quantify the quality of synthetic data as a single number. In fact, it is generally agreed that there are multiple independent factors that affect synthetic data quality [159, 4]. Ideally, we want to produce synthetic data which is good with respect to all factors; however, there can sometimes be a tension between factors. For instance, we want our synthetic data to 'look like real data', but we do not want it to create exact copies of the training data. The approach we take in this chapter is to evolve an ensemble of Banksformer models, where diversity is defined with respect to a set of quality metrics, which we detail in Section 5.4.1.

5.3 Evolutionary Algorithm

The basis of our method is the MAP-Elites algorithm [108], which we described in Section 4.2.2 of the previous chapter. In Chapter 4, we built upon MAP-Elites, by introducing a variational autoencoder (VAE) for mapping raw high-dimensional descriptors to a two-dimensional space. As we discuss below, this exact method is not suitable for evolving a population of Banksformer models, due to amount of computational effort required to evaluate a single BF model. In this section, we propose a novel MAP-Elites based algorithm, tailored to evolving Banksformer ensembles, which is inspired by our approach in the previous chapter.

Population structure

We structure our population as a 3×3 grid-shaped archive, following MAP-Elites [108]. We are restricted to a significantly smaller population than we used in Chapter 4, because the time to evaluate a single BF individual is many orders of magnitude greater than the evaluation time in Chapter 4 (seconds vs hours). To initialize the population, we first create and evaluate an initial set of 90 individuals. After the initial selection step we are left with a maximum population size of 9 because of the grid structure. After initialization, we run evolution for 29 additional epochs, where each epoch involves creating and evaluating nine new individuals, by applying the *variation* to randomly selected individuals from the current population. In preliminary experiments, we experimented with up to 50 epochs, but found that the results stabilized by the 20th epoch.

Encoding of individuals

In our EA, each individual defines a set of hyper-parameters for a Banksformer model. We represent each individual with a dictionary, which encodes the value for each hyper-parameter. There are four types of hyper-parameters; categorical, integer, integer_2, and float. The integer_2, type is an integer that must be a power of 2. In Table 5.1, we describe the set of hyper-parameters which define an individual.

Fitness

The fitness of an individual should indicate the quality of a Banksformer model trained using the hyper-parameters defined by the individual. As we have discussed extensively, there are many ways in which this can be measured. Initially, we conducted experiments using the validation loss during training of the model as the fitness value, but found this did not produce better data than the ensemble experiments from Chapter 3. We found that instead, using the mean rank of a synthetic dataset produced by the model led to significantly better results.

One significant difference between our method here and our method from Chapter 4 is the computational effort required for evaluating an individual. The time to train and evaluate a Banksformer model varies based on the specific hyper-parameters used,

Table 5.1: Hyper-parameters defining an individual. In this table, we list the parameters which define an individual in our evolutionary algorithm, and give a brief description of each.

Parameter	Type	Initial Range	Description
learning_rate	float	$[10^{-5}, 0.09]$	Controls the learning rate of the optimizer
clipnorm	float	[0.75, 4.0]	Controls the strength of gradient clipping
dropout_rate	float	[0.1, 0.75]	Controls the dropout rate of Banksformer during training
num_layers_dec	integer	[1, 7]	Number of decoder layers
batch_size	integer_2	[8, 128]	Number of samples to use per batch
num_heads	integer_2	$\{1, 2, 4, 8\}$	Number of attention heads
d_model	integer_2	[32, 512]	Number of hidden units per layer
opt_name	categorical	$\{adam, rms\}$	Name of the optimizer used for training, either Adam [71] or RMSProp [54]
use_clip	categorical	$\{true, false\}$	Controls if gradient clipping is used. If false the clipnorm parameter is ignored
use_custom_lr	categorical	$\{true, false\}$	Controls if custom learning rate schedule is used. If true, the learning_rate parameter is ignored

but typically takes at least 4 hours on a cloud computing node using 2 CPUs with 12GB of memory per CPU. We set a maximum time limit of 12 hours for evaluation, and if no dataset was produced within this time, then the individual is discarded. This is many orders of magnitude slower than the evaluation in Chapter 2, which took under a second per individual. However, because we use back-propagation – as opposed to evolution – to learn the models' weights, we should be able to find high-performing solutions much faster.

Variation

Variation operators are used to create new individuals from existing individuals in the current population. We use three variation operators in our EA: *mutation*, *arithmetic crossover*, and *discrete crossover*. The exact implementation of each depends on the type of parameter. Generally, the mutation operator changes a parameter to a similar value, and the crossover operators combine two individuals into a new individual. In arithmetic crossover, the individuals are combined by averaging the values of the numeric parameters. In discrete crossover, for each parameter, we randomly select one parent, and use that parent's value. Unlike the other variation methods, discrete crossover does not depend on the parameter type, as it always randomly chooses a value from the parents.

For categorical parameters, variation is straightforward. The mutation operator simply randomly selects a new value from the possible options. Both crossover operators are the same for categorical parameters; they simply randomly choose between the parents' values.

The mutation operator changes floating point values by a random amount, between -15% and $+15\%$. The arithmetic crossover takes the mean of the parent's values, and discrete crossover randomly chooses a single parent's value.

For integers, mutation adds or subtracts 1 or 2, and a condition prevents integers from mutating to values less than 1. The arithmetic crossover takes the average of the two parents and randomly rounds the result if it is not an integer.

With integer_2 values, mutation will either randomly double or half the value of these parameters, ensuring it does not go below 1. For arithmetic crossover, we first compute the $\log_2$ of the values for each parent, and then average these values, randomly rounding to an integer if necessary. The new value is 2 to the power of this integer.

5.4 Behavior Space

When using the MAP-Elites algorithm, the behavior space defines the type of diversity we produce. In Chapter 4, our goal was to create error diversity with respect to samples. That is, we aimed to create a set of classifiers with high predictive accuracy,

which tend to make errors on different samples.

Our goal in this chapter is to create synthetic data which is deemed *high quality*, with respect to a set of quality metrics. As we have a large set of metrics, we adopt an approach similar to the Neuro-MAP-Elites method employed in the previous chapter, and begin with a high-dimensional raw descriptor which is then compressed into a two-dimensional descriptor. However, the heightened demands on our evaluation process necessitated reconsidering our approach to compressing raw behavior descriptors. Previously, our Neuro-MAP-Elites method employed a variational autoencoder (VAE) for this purpose, given its capability to learn complex, non-linear relationships. Unfortunately, the inherent flexibility of VAEs can result in overfitting when there is a shortage of training data [50]. Given our constraint of evaluating significantly fewer individuals when evolving BF models, we opted to employ Principal Component Analysis (PCA) to reduce the dimensionality of the raw descriptors, as opposed to a VAE. PCA is deterministic, and performs dimensionality reduction via a linear mapping, making it much less prone to overfitting [50].

Another motivation for using PCA is that it is well suited for data with strong linear correlations among features. This is beneficial, as our set of raw metrics contains some highly correlated metrics. As we discuss below, there are a few aspects of data quality we are particularly interested in; however, for each of these intuitive aspects, there are various ways we could create concrete metrics. We aim to take a balanced approach, including multiple metrics for each criteria, while still keeping the total number of metrics to a reasonable level. We hypothesize that many of our metrics will exhibit strong linear correlations, and therefore PCA is well suited to compressing the raw descriptors. The following section details the complete set of metrics that define the raw behavior space.

5.4.1 Metrics for raw behavior descriptors

The individual metrics used to define the behavior descriptor can be grouped based on what facet of data quality they are designed to measure. The number of metrics in each category varies, as some categories contain many highly correlated metrics. Because we use PCA-based dimensionality reduction, including additional metrics which are highly correlated with existing metrics will only have a minor impact on the compressed behavior descriptor.

Closeness of marginal distributions

The univariate and bi-variate distribution of individual features should be similar between the synthetic data and the original data used for training. To measure the closeness of the marginal distributions, we use the metrics defined in Chapter 2, which compare univariate and bi-variate distributions. Specifically, we use the metrics which compare the univariate distributions for the *amount*, *tcode*, *day of the month* features defined in Section 2.6.1, and the joint *tcode, day of the month* distribution, defined in Section 2.6.3.

Sequence structure

One limitation of comparing the marginal distributions is they do not capture any information about the sequence structure. We employ two approaches for quantifying how well the sequence structure of the synthetic data conforms to the structure of real sequences. The first is with the 3-gram metric described in Section 2.6.2, which compares the frequencies of length three tcode sequences, to ensure transaction ordering is similar between both datasets. The second is the LDA metric described in Section 3.3.1, which compares the behavior patterns between real and synthetic sequences.

Indistinguishability from real data

Given a mixed dataset, containing real and synthetic samples, it should be difficult to distinguish if any given sample is real or synthetic. There are many potential ways to implement a metric based on the notion of indistinguishability. The two components required for an indistinguishability metric are (1) a binary classifier, and (2) a scoring criteria. We use the same set of binary classification algorithms described in Section 3.3.2, and additionally, we include a single-layer LSTM neural network. We added the LSTM because it can classify variable sequences directly, without needing to convert them into fixed-sized representations. For each classifier, we consider two scoring criteria; the accuracy and the area under the receiver operator curve (AUC). Accuracy is an intuitive score, as it simply reports the proportion of samples correctly classified. AUC is a more robust but less intuitive measure, which factors in both the true positive and false positive rates [41]. In total, we have 12 metrics (6 classifiers $\times$ 2 scoring criteria) that measure distinguishability.

Consistency

We also want the relationships between all features to be consistent between our synthetic data and data from the real distribution. The approach we take to measuring this is to create a set of supervised learning tasks, where each task involves predicting a different transaction feature based on all other features. We consider two such tasks, namely, predicting the tcode (classification) and predicting the amount (regression). For each task, we compute the metric described in Section 3.3.3 and defined in Equation 3.4, which captures how much performance at these tasks decreases when the model is trained on one dataset (either real or synthetic), and evaluated on the other.

Coverage

Given a mixed dataset, containing an equal number of real and synthetic samples, the nearest neighbors of each sample should be roughly half real and half synthetic. This is measured with the coverage metrics, described in Section 3.3.4.

Non-memorization

The synthetic data we produce should be an unbiased sample from the real data distribution, and not simply a copy of the real data we trained our model on. We measure non-memorization using the holdout metric described in Section 3.3.5, which quantifies the difference between how close synthetic data is to the training data, and how close it is to held-out data from the same distribution.

5.5 Results

Table 5.2 compares the results from the best single-model (evo-s) and multi-model (evo-mm) datasets produced by our evolutionary algorithm against the best datasets produced in previous chapters. Specifically, we include the Banksformer and DoppelGANger results from Chapter 2 and results from the best ensemble dataset produced in Chapter 3. Table 5.2 shows that while our evolutionary algorithm does not strictly outperform the elite ensemble approach from Chapter 3, it does create better data according to some metrics. One reason for the limited improvement provided by our

EA versus the ensemble results from Chapter 3, is that in Chapter 3 we used hyper-parameters which we spent a significant amount of effort tuning while developing BF for use on the *czech* dataset in Chapter 2. A significant value of our evolutionary algorithm is its ability to search for good hyper-parameter settings during the course of the algorithm, while working towards producing a final ensemble dataset. Another takeaway from Table 5.2, is that datasets created by ensembles – that is, the *ensemble-e* dataset from Chapter 3 and the *evo-mm* dataset from this chapter – tend to outperform single model datasets. However, despite this general trend, there are some metrics on which the single model dataset produced by our evolutionary outperform the best ensemble datasets. Interestingly, there is no clear connection between the four metrics the single-model dataset performs best on, they include metrics based on distributional closeness, classifiers and non-memorization.

Table 5.2: Quality comparison of synthetic banking data generated by different methods. Results are shown for DoppelGANger (DG), Banksformer (BF) as detailed in Chapter 2, best performing Banksformer ensemble (ensemb) from Chapter 3, and best single-model (evo-s) and multi-model (evo-mm) evolutionary methods.
*The holdout metric was not computed for the DG and BF datasets, as we do not have information on the training/validation split used for these earlier models.

	DG	BF	ensemble-e	evo-s	evo-mm
amount-wasser	**1544**	3004	2495	2392	2397
td-wasser	5.275	0.8514	0.6649	**0.6627**	0.7698
tcode-jsd	0.0103	0.0116	0.0019	0.0033	**0.0003**
day-jsd	0.1614	0.0159	**0.0043**	0.0066	**0.0043**
tcode_day-jsd	0.4111	0.0435	**0.0153**	0.0199	0.0199
RF_auc	1.0	0.993	**0.9023**	0.9798	0.9667
DT-1_auc	1.0	0.6483	0.5808	**0.5734**	0.5885
LR_auc	1.0	0.8197	0.7059	**0.6424**	0.6766
KN7_auc	1.0	0.9474	**0.7974**	0.9154	0.8922
lstm_auc	0.9998	0.9932	**0.8667**	0.9874	0.9857
tcode_num_consist	0.2035	0.1991	0.1335	0.1462	**0.1248**
log_amount_sc_consist	0.6914	0.3528	**0.2961**	0.3011	0.4033
cover3	1.0	0.8402	**0.5522**	0.708	0.7054
cover5	1.0	0.8466	**0.6716**	0.7384	0.7239
lda	0.2081	0.13	0.0594	0.0699	**0.0582**
tcode-3g-jsd	0.19	0.0954	**0.0194**	0.0339	0.035
holdout_val	NA*	NA*	0.5254	**-0.8044**	0.388

A likely explanation for the single-model dataset performing best at these metrics

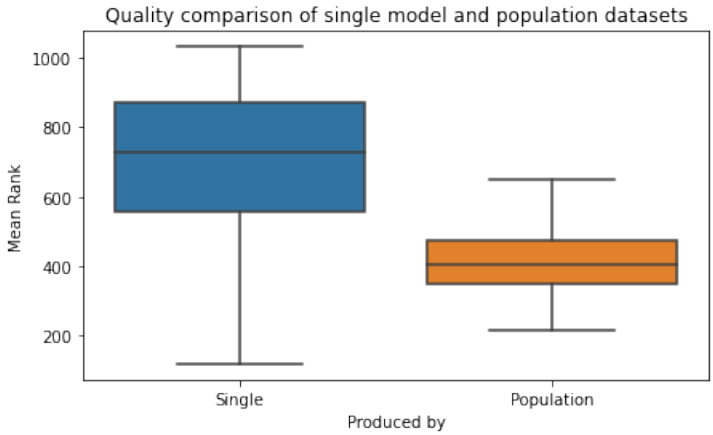

Figure 5.1: Comparison of datasets produced by single models and population of models. This figure shows that typically, datasets generated with a population outperform datasets generated by a single model. However, despite this average trend, the best datasets are produced by single models, due to the large variance in the quality of single-model datasets.

is the fact that, as we discussed in Chapter 3, the single model datasets have a much larger variance in quality. In Figure 5.1 we compare the overall quality of all the single model datasets created while running our EA against that of the population datasets. This figure shows that while on average the overall quality of multi-model datasets is better than single-model datasets, the best single-model datasets are better than the best multi-model datasets. This is due to the much higher variance in the quality of single-model datasets compared to multi-model.

In order to study how the quality of synthetic data evolves over the course of

Figure 5.2: Quality of synthetic dataset produced by the population during evolution. At the end of each evolutionary epoch, we generate an ensemble dataset with the procedure defined in Section 3.4 of the previous chapter, using the current population as the ensemble.

our EA, at the end of each evolutionary epoch we create 10 ensemble datasets from the current population. As we show in Figure 5.2, there was minimal improvement during the first 18 epochs, after which a significant improvement occurred. However, after this improvement, we found the data quality stabilized, and did not significantly improve further. Between the 10th to 18th epoch, there is an increase in the variability of data quality, however no lasting gains are made until the 18th epoch. While developing this method, we ran the evolutionary algorithm multiple times, and found the results generally followed this pattern. That is, minimal improvement during the first 10-20 epochs, followed by a significant improvement slightly before the 20th epoch. In these preliminary runs we experimented with up to 50 epochs, but found that after a significant improvement between epoch 10 and 20 there were no further clear improvements.

5.6 Conclusion

In this chapter, we have applied a variant of the quality-diversity (evolutionary) algorithm developed in Chapter 4, to the task of generating synthetic banking data, improving on our previous best results from Chapter 3. While this approach does lead to improvements in data quality, it also introduces extra computational effort. Whether the improvements are significant enough to justify the added computation will depend on the specific use case of the synthetic data. For scenarios where having the best possible synthetic data is crucial, the methods from this chapter provide a pathway to improved results.

Another scenario where this method can be helpful is if a user plans to apply Banksformer to a new dataset, and decides to tune the parameters based on the target dataset. In this case, the effort to tune the parameters may be comparable to running our evolutionary algorithm to optimize the hyper-parameters while simultaneously constructing ensemble datasets. The models in the ensembles from Chapter 3 all used the same hyper-parameter values, which were determined when creating the original Banksformer model in Chapter 2.

Chapter 6

Discussion

Throughout this thesis, we have made many contributions which have improved upon state-of-the-art synthetic transaction sequence generation. In Chapter 2, we proposed Banksformer, which is the basic model used in our synthetic data generation framework. Banksformer is based on the transformer neural network architecture, and it employs special conditional generation and date generation mechanisms, which were designed to facilitate learning date-based patterns. In Chapter 2 we compared our newly proposed Banksformer model against existing GAN models which could be used to generate banking data, and found Banksformer to be a superior approach. We also conducted a series of ablation experiments to demonstrate the benefits of both the conditional generation and date generation mechanisms in Banksformer.

The work in Chapter 2 relies on a set of metrics based solely on comparing feature distributions to evaluate our synthetic data. While these metrics do provide valuable insights to data quality, they do not provide a complete picture. In Chapter 3 we build upon the metrics from Chapter 2 and propose additional metrics to measure other aspects of data quality missed by the previous metrics, as well as qualitative analysis to compare the account structures between real and synthetic dataset.

Equipped with this enhanced evaluation framework, Chapter 3 aims to improve upon the quality of synthetic data by creating ensemble datasets which combine samples from multiple copies of BF. Throughout Section 3.4, we explored different approaches to ensemble construction. We found that adopting an elitist approach, where we trained N models and selected only k ($k < N$) of the models based on a scoring criteria, led to the best ensemble results. Additionally, if there is a specific subset of

metrics of interest, such as metrics which compare feature distributions, it is possible to create a scoring criteria tailed to those metrics.

In Chapter 3 we also developed a differentially private version of BF, based on differentially private stochastic gradient descent. While this model was able to generate synthetic data with a formal privacy guarantee, we found that adding even an extremely weak guarantee substantially degraded the quality of synthetic data. Despite not being able to produce high quality private data, it is possible that our DP-BF model may be useful in some limited scenarios where data quality does not matter. In Section 3.3.5 we compared the rate of identical transactions between our synthetic data and the real data used to train our model with the rate of identical transactions between our synthetic data and held-out real data not used to train the model. This analysis strongly suggested that our model does not simply memorize the training data. However, while encouraging, it does not provide any formal guarantees.

Chapter 4 developed a quality-diversity evolutionary algorithm to improve ensemble creation. We began Chapter 4 by discussing the benefits of diversity in ensembles. Inspired by recent work in the field of evolutionary computing on quality-diversity algorithms, which can create populations of models that are high-performing and qualitatively diverse, we believed these algorithms could be powerful tools for ensemble construction. In Chapter 4, we focused on a classification task, and therefore we sought to create a population which was diverse with respect to the errors made by individual models. Because of constraints on the existing MAP-Elites quality-diversity algorithm, it could not be directly applied to create error-diverse populations. This lead us to propose Neuro-MAP-Elites, a novel quality-diversity algorithm which extends the original MAP-Elites algorithm. In Section 4.6, we demonstrated that Neuro-MAP-Elites performs favorably against other methods on a selection of binary classification problems.

Finally, in Chapter 5, we adapted the Neuro-MAP-Elites algorithm proposed in Chapter 4 to create a variant which can be used with Banksformer. In Chapter 5, we were interested in creating diversity with respect to a set of metrics, as opposed to the error diversity we sought in Chapter 4. Another significant difference between these chapters, is that in Chapter 5, evaluation of an individual in the evolutionary algorithm took many orders of magnitude longer than in Chapter 4. This led us to create a significantly more efficient evolutionary algorithm, which was still able to

create diverse populations of high-performing individuals while evaluating significantly fewer individuals. As discussed in Chapter 5, it is possible to significantly reduce the number of evaluations because we only used evolution to optimize the hyper-parameters, while still relying on back propagation to optimize the model parameters.

We concluded Chapter 5 by comparing the quality of synthetic banking data produced by our evolutionary algorithm against the best synthetic datasets from Chapters' 2 and 4. Our analysis found mixed results, with the overall quality of the synthetic data produced by evolution similar to that produced by the ensemble methods in Chapter 3. However, as discussed in Section 3.4, one reason why we only saw limited gains from our algorithm is that the ensemble results from Chapter 3 were obtained with parameters which had been tuned extensively in prior work.

6.1 Discussion of Impacts

Throughout this thesis, we have created a novel approach to synthetic banking data generation and evaluation, which creates higher quality data than previous approaches. The synthetic data we produced with the ensemble methods detailed in Chapters' 3 and 5 outperforms GAN generated data across a broad spectrum of metrics.

One of the main motivations for creating synthetic banking data is to allow more researchers and organizations to access banking data, without compromising the privacy of individuals and their transaction histories. Our framework is able to create high quality synthetic transaction sequence data. Further, we have extensively reviewed the synthetic data we generate, and find no obvious leaks of user information from the real data we trained our model on. However, the high quality data we generate is not *guaranteed* to protect user privacy. In Chapter 3, we created a variant of our model which does offer a formal privacy guarantee, however that model produces much lower quality data. It is worth reflecting on the progress made in this thesis, the limitations of our work, and the ways these limitations can be overcome in the future.

While in its current form, our framework is not able to create synthetic banking data from real customer data which could be freely distributed to researchers and others without fear of violating user privacy, it does represent significant progress towards this goal. Further, our method in its current form can still be used to increase

customer data privacy at institutions which work with financial transaction sequences. As an example, currently there are private institutions, such as large banks, which have access to large volumes of financial data, which they may use for research and development. This involves the institution authorizing person(s) to access the private data, under an agreement to not reveal any private user information. While this approach often works perfectly well to protect user privacy, it depends on the person(s) with data access abiding by the terms of their agreement. With our framework, the institution could further protect the privacy of its customers, by providing persons(s) who need to access the data from research and development with synthetic data, as opposed to real customer data. As we have discussed, though we cannot formally prove our synthetic data protects privacy, it is highly non-trivial (maybe not possible) to try and make inferences about which customers or transactions were included in the real data used for training. This approach adds to the customer data privacy, because it eliminates the chance that a researcher will 'accidentally' learn private information. Further, if a malicious researcher does want to try and reveal private information, it will greatly increase the amount of effort they need to expend to obtain this information.

6.2 Future Work

While our work represents an improvement over prior work, there is still further work which can be done to improve upon both the quality and privacy of synthetic banking data. One clear indication that data quality can be improved is the scores on the classifier-based metrics shown in the final results Table 5.2 shows that various classifiers are able to distinguish between real and synthetic samples significantly better than by chance.

There are multiple directions which could be pursued to further improve data quality. One promising direction could be considering other types of generative models. Since we began work on Banksformer, there has been a significant amount of work on novel types of generative models, such as diffusion models [183], and combining different types of generative models, such as training transformers using GAN methods [64]. Our work only considers GAN and transformer based models, which were considered state-of-the-art when we began this research. Future work should consider

other types of generative models as a possible way to improve data quality.

A related pathway to improvement is considering other approaches to ensemble construction, such as boosting [145]. As we have argued throughout this thesis for the benefits of ensemble diversity, future work should consider ensemble approaches that are capable of incorporating models with different architectures. The elite ensemble methods from Chapter 3 could be easily adapted to incorporate different architectures; however the evolutionary approach in Chapter 5 would need to adapted, as many hyper-parameters are model specific.

Privacy is important aspect of synthetic data, particularly when working with sensitive data such as financial records. While our work developing a DP-BF model in Chapter 3 was not able to produce high quality private data, in Section 3.5 we discuss that creating a DP version of BF based on PATE might be a more promising path to improvement. The best approach to creating private synthetic data depends on why privacy is desired, as there are many different notions of privacy. For instance, while differentially private methods such as DP-SGD and PATE are able to provide formal mathematical guarantees which characterize the privacy level, these mathematical frameworks often do not align with legal requirements around data privacy, which exist in some domains.

Overall, the work in this thesis has created a state-of-the-art synthetic transaction sequence generator, explored methods for boosting the performance of our generator by combining multiple models, and provided a framework for evaluating synthetic transaction sequences. Our focus was on creating synthetic banking data, and we believe our work can be used to help reduce the need for real customer data when developing tools to analyze and predict such data. As discussed, being able to produce synthetic data which is both high quality and differentially private is still an open challenge to be pursued in future work. While the high quality data we produce is not mathematically guaranteed to be private, our analysis in Chapter 3 suggest that we are not memorizing training data. One concrete application for our model is to create synthetic data of financial transactions for internal use by financial organizations. Employees of the organizations can work with synthetic data to develop or design new tools, instead of real, private data. This improves the security of the private data by limiting who has access or reducing the period of time access is needed. In the long term, we hope that this work can be extended to create high quality banking data with

formal privacy guarantees, which could facilitate data sharing between organizations and with academics.

Code availability

- https://github.com/BigTuna08/Banksformer_ecml_2022 - Code for experiments from Chapter 2.
- https://github.com/BigTuna08/banksformer-multi-model - Code for experiments from Chapters 3 & 5.
- https://github.com/BigTuna08/nme - Code for experiments from Chapters 4.

Bibliography

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] T. Adel, Z. Ghahramani, and A. Weller. Discovering interpretable representations for both deep generative and discriminative models. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 50–59. PMLR, 10–15 Jul 2018.

[3] J. Adler and S. Lunz. Banach wasserstein gan. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6754–6763. Curran Associates, Inc., Montreal, 2018.

[4] A. M. Alaa, B. van Breugel, E. Saveliev, and M. van der Schaar. How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models. *CoRR*, abs/2102.08921, 2021.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

[6] S. Assefa, D. Dervovic, M. Mahfouz, T. Balch, P. Reddy, and M. Veloso. Generating synthetic data in finance: Opportunities, challenges and pitfalls. *InfoSciRN: Data Protection (Topic)*, 2020.

[7] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[8] E. Bagdasaryan, O. Poursaeed, and V. Shmatikov. Differential privacy has disparate impact on model accuracy. *Advances in neural information processing systems*, 32, 2019.

[9] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[10] L. Beadle and C. Johnson. Semantically driven crossover in genetic programming. pages 111 – 116, 07 2008.

[11] L. Beadle and C. Johnson. Semantically driven mutation in genetic programming. pages 1336–1342, 05 2009.

[12] S. M. Bellovin, P. K. Dutta, and N. Reitinger. Privacy and synthetic datasets. *Stan. Tech. L. Rev.*, 22:1, 2019.

[13] J. Berkson. Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227):357–365, 1944.

[14] C. M. Bishop. Novelty detection and neural network validation, 1994.

[15] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.

[16] S. Boisvert and J. W. Sheppard. Quality diversity genetic programming for learning decision tree ensembles. In *Genetic Programming*, pages 3–18, Cham, 2021. Springer International Publishing.

[17] D. Borrajo, M. Veloso, and S. Shah. Simulating and classifying behavior in adversarial environments based on action-state traces: an application to money laundering. *ArXiv*, abs/2011.01826, 2020.

[18] M. F. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[19] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

[20] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5 – 20, 2005. Diversity in Multiple Classifier Systems.

[21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[22] D. Byrd, M. Hybinette, and T. H. Balch. Abides: Towards high-fidelity market simulation for ai research. *ArXiv*, abs/1904.12066, 2019.

[23] R. Canaan, J. Togelius, A. Nealen, and S. Menzel. Diverse agents for ad-hoc cooperation in hanabi. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019.

[24] S. Candemir, X. V. Nguyen, L. R. Folio, and L. M. Prevedello. Training strategies for radiology deep learning models in data-limited scenarios. *Radiology: Artificial Intelligence*, 3(6):e210014, 2021.

[25] R. P. Cardoso, E. Hart, D. B. Kurka, and J. V. Pitt. Using novelty search to explicitly create diversity in ensembles of classifiers. GECCO '21, page 849–857, New York, NY, USA, 2021. ACM.

[26] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *SSST@EMNLP*, 2014.

[27] R. Cont. Empirical properties of asset returns: Stylized facts and statistical issues. *Quantitative Finance*, 1:223–236, 01 2001.

[28] E. Creager, D. Madras, J.-H. Jacobsen, M. Weis, K. Swersky, T. Pitassi, and R. Zemel. Flexibly fair representation learning by disentanglement. In *International conference on machine learning*, pages 1436–1445. PMLR, 2019.

[29] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521:503 EP –, 05 2015.

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

[31] E. Dikici, M. Bigelow, R. D. White, B. S. Erdal, and L. M. Prevedello. Constrained generative adversarial network ensembles for sharable synthetic medical images. *Journal of Medical Imaging*, 8(2):024004, 2021.

[32] J. Ding, A. Condon, and S. P. Shah. Interpretable dimensionality reduction of single cell transcriptome data with deep generative models. *Nature communications*, 9(1):1–13, 2018.

[33] E. Dolson, A. Lalejini, and C. Ofria. Exploring genetic programming systems with map-elites, 08 2018.

[34] G. Douzas and F. Bacao. Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Syst. Appl.*, 91(C):464–471, jan 2018.

[35] K. Doya. Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, 1993.

[36] C. Dwork. Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming*, ICALP'06, pages 1–12, Berlin, Heidelberg, 2006. Springer-Verlag.

[37] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[38] G. Eilertsen, A. Tsirikoglou, C. Lundström, and J. Unger. Ensembles of gans for synthetic training data generation. *arXiv preprint arXiv:2104.11797*, 2021.

[39] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

[40] R. M. Farsani and E. Pazouki. A transformer self-attention model for time series forecasting. *Journal of Electrical and Computer Engineering Innovations (JECEI)*, 9(1):1–10, 2021.

[41] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[42] M. C. Fontaine, S. Lee, L. B. Soros, F. De Mesentier Silva, J. Togelius, and A. K. Hoover. Mapping hearthstone deck spaces through map-elites with sliding boundaries. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 161–169, New York, NY, USA, 2019. Association for Computing Machinery.

[43] V. Fortuin, D. Baranchuk, G. Rätsch, and S. Mandt. Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*, pages 1651–1661. PMLR, 2020.

[44] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[45] M. Frid-Adar, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. Synthetic data augmentation using gan for improved liver lesion classification. In *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pages 289–293. IEEE, 2018.

[46] T. Gaber, A. Tharwat, A. Ibrahim, and A. Hassanien. Linear discriminant analysis : a detailed tutorial. *AI Communications*, 30(2):169–190, 2017.

[47] C. Gagné, M. Sebag, M. Schoenauer, and M. Tomassini. Ensemble learning for free with evolutionary algorithms? In *Proceedings of the 9th Annual Conference*

on Genetic and Evolutionary Computation, GECCO '07, page 1782–1789, New York, NY, USA, 2007. Association for Computing Machinery.

[48] M. Germain, K. Gregor, I. Murray, and H. Larochelle. Made: Masked autoencoder for distribution estimation. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR.

[49] A. Ghorbani, V. Natarajan, D. Coz, and Y. Liu. Dermgan: Synthetic generation of clinical skin images with pathology. In *Machine Learning for Health Workshop*, pages 155–170. PMLR, 2020.

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[51] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. *ArXiv*, abs/1406.2661, 2014.

[53] Google. Tensorflow privacy.

[54] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[55] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis. Procedural content generation through quality diversity. *CoRR*, abs/1907.04053, 2019.

[56] A. Grover and S. Ermon. Boosted generative models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[57] J. T. Guibas, T. S. Virdi, and P. S. Li. Synthetic medical images from dual generative adversarial networks. *arXiv preprint arXiv:1709.01872*, 2017.

[58] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., California, 2017.

[59] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition.* Springer series in statistics. Springer, 2009.

[60] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[61] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *ICLR*, 2017.

[62] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[63] A. Horner and D. E. Goldberg. *Genetic algorithms and computer-assisted music composition*, volume 51. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 1991.

[64] Y. Jiang, S. Chang, and Z. Wang. Transgan: Two pure transformers can make one strong gan, and that can scale up, 2021.

[65] J. Jordon, D. Jarrett, E. Saveliev, J. Yoon, P. Elbers, P. Thoral, A. Ercole, C. Zhang, D. Belgrave, and M. van der Schaar. Hide-and-seek privacy challenge: Synthetic data generation vs. patient re-identification. In H. J. Escalante and K. Hofmann, editors, *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 206–215. PMLR, 06–12 Dec 2021.

[66] J. Jordon, J. Yoon, and M. van der Schaar. Measuring the quality of synthetic data for use in competitions, 2018.

[67] J. Jordon, J. Yoon, and M. Van Der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. In *International conference on learning representations*, 2019.

[68] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[69] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. *CoRR*, abs/1912.04958, 2019.

[70] N. Killoran, L. J. Lee, A. Delong, D. K. Duvenaud, and B. J. Frey. Generating and designing dna with deep generative models. *ArXiv*, abs/1712.06148, 2017.

[71] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[72] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[73] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[74] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2014.

[75] A. Klushyn, N. Chen, R. Kurle, B. Cseke, and P. van der Smagt. Learning hierarchical priors in vaes. 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.

[76] A. Koshiyama, N. Firoozye, and P. Treleaven. Generative adversarial networks for financial trading strategies fine-tuning and combination. *Quantitative Finance*, 21(5):797–813, 2021.

[77] J. R. Koza. Genetic programming: Automatic programming of computers. *EvoNews*, 1(3):4–7, Mar. 1997.

[78] K. Krawiec and P. Lichocki. Approximating geometric crossover in semantic space. GECCO '09, page 987–994, New York, NY, USA, 2009. ACM.

[79] D. V. Kute, B. Pradhan, N. Shukla, and A. Alamri. Deep learning and explainable artificial intelligence techniques applied for detecting money laundering–a critical review. *IEEE Access*, 9:82300–82317, 2021.

[80] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 29–37, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[81] B. LeBaron. Agent-based computational finance. *Handbook of computational economics*, 2:1187–1233, 2006.

[82] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436 EP –, 05 2015.

[83] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[84] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[85] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218, 2011.

[86] J. Lehman and K. O. Stanley. Evolvability is inevitable: Increasing evolvability without the pressure to adapt. *PLOS ONE*, 8(4):1–9, 04 2013.

[87] J. Li, X. Wang, Y. Lin, A. Sinha, and M. P. Wellman. Generating realistic stock market order streams. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 727–734. AAAI Press, 2020.

[88] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. 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.

[89] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto. Diffusion-lm improves controllable text generation. *arXiv preprint arXiv:2205.14217*, 2022.

[90] Z. Li, T. Xia, X. Lou, K. Xu, S. Wang, and J. Xiao. Adversarial discrete sequence generation without explicit neuralnetworks as discriminators. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 3089–3098. PMLR, 16–18 Apr 2019.

[91] R. Lieck, F. C. Moss, and M. Rohrmeier. The tonal diffusion model. *Transactions of the International Society for Music Information Retrieval*, 3(1), 2020.

[92] B. Lim, S. Arık, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[93] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 464–483, New York, NY, USA, 2020. Association for Computing Machinery.

[94] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 464–483, New York, NY, USA, 2020. Association for Computing Machinery.

[95] Z. Lin, A. B. Owen, and R. B. Altman. Genomic research and human subject privacy, 2004.

[96] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. M. Shazeer. Generating wikipedia by summarizing long sequences. *ArXiv*, abs/1801.10198, 2018.

[97] E. Lopez-Rojas. *Applying Simulation to the Problem of Detecting Financial Fraud*. PhD thesis, 2016.

[98] E. Lopez-Rojas, A. Elmir, and S. Axelsson. Paysim: A financial mobile money simulator for fraud detection. In *28th European Modeling and Simulation Symposium, EMSS, Larnaca*, pages 249–255. Dime University of Genoa, 2016.

[99] E. A. Lopez-Rojas and S. Axelsson. Banksim: A bank payment simulation for fraud detection research. 09 2014.

[100] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics.

[101] L. Luzi, R. Balestriero, and R. G. Baraniuk. Ensembles of generative adversarial networks for disconnected data. *arXiv preprint arXiv:2006.14600*, 2020.

[102] D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2003.

[103] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[104] J. Miller. *Cartesian Genetic Programming*, volume 43. 06 2003.

[105] M. Mirza and S. Osindero. Conditional generative adversarial nets, 2014.

[106] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In *PPSN XII*, pages 21–31, Berlin, Heidelberg, 2012. Springer.

[107] F. Mourato, M. P. dos Santos, and F. Birra. Automatic level generation for platform videogames using genetic algorithms. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, pages 1–8, 2011.

[108] J. Mouret and J. Clune. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909, 2015.

[109] A. Narayanan and V. Shmatikov. How to break anonymity of the netflix prize dataset. *arXiv preprint cs/0610105*, 2006.

[110] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, Cambridge MA, 2002.

[111] Q. U. Nguyen, N. Hoai, M. O'Neill, R. McKay, and E. Galván-López. Semantically-based crossover in genetic programming: Application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12:91–119, 06 2011.

[112] K. Nickerson and T. Hu. Principled quality diversity for ensemble classifiers using map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 259–260, 2021.

[113] K. Nickerson, A. Kolokolova, and T. Hu. Creating diverse ensembles for classification with genetic programming and neuro-map-elites. In E. Medvet, G. Pappa, and B. Xue, editors, *Genetic Programming*, pages 212–227, Cham, 2022. Springer International Publishing.

[114] K. L. Nickerson, T. S. Tricco, A. Kolokolova, F. Shoeleh, C. Robertson, J. A. Hawkin, and T. Hu. Banksformer: A deep generative model for synthetic transaction sequences. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2022, Grenoble, France*, 2022.

[115] F. Nielsen. On the jensen–shannon symmetrization of distances relying on abstract means. *Entropy*, 21(5):485, 2019.

[116] Z. Niu, G. Zhong, and H. Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[117] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2642–2651. JMLR.org, 2017.

[118] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017.

[119] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. cite arxiv:1609.03499.

[120] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4797–4805, USA, 2016. Curran Associates Inc.

[121] A. Pal and V. N. Balasubramanian. Adversarial data programming: Using gans to relax the bottleneck of curated labeled data. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1556–1565, 2018.

[122] E. Panayi, M. Harman, and A. Wetherilt. Agent-based modelling of stock markets using existing order book data. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 101–114. Springer, 2012.

[123] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*, 2018.

[124] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[125] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[126] Petr Berka and Marta Sochorova. PKKD '99 Discovery Challenge. https://web.archive.org/web/19991010204337/http://lisp.vse.cz/pkdd99/berka.htm, 1999. Accessed: 2022-04-01.

[127] L. Petrocelli. Some translated/reformatted czech banking data. retrieved from: https://data.world/lpetrocelli/some-translatedreformatted-czech-banking-data/access.

[128] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[129] M. Platzer and T. Reutterer. Holdout-based empirical assessment of mixed-type synthetic data. *Frontiers in big Data*, 4:679939, 2021.

[130] R. J. Prenger, R. Valle, and B. Catanzaro. Waveglow: A flow-based generative network for speech synthesis. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621, 2019.

[131] J. K. Pritchard, M. Stephens, and P. Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, 2000.

[132] J. K. Pugh, L. B. Soros, and K. O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[133] A. Ramdas, N. García Trillos, and M. Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2):47, 2017.

[134] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[135] N. Rasiwasia and N. Vasconcelos. Latent dirichlet allocation models for image classification. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2665–2679, 2013.

[136] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1060–1069, New York, New York, USA, 20–22 Jun 2016. PMLR.

[137] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France, 07–09 Jul 2015. PMLR.

[138] A. J. Riesselman, J. B. Ingraham, and D. S. Marks. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods*, 15(10):816–822, Oct 2018.

[139] I. Rish. An empirical study of the naïve bayes classifier. *IJCAI 2001 Work Empir Methods Artif Intell*, 3, 01 2001.

[140] N. M. Rodrigues, J. E. Batista, and S. Silva. Ensemble genetic programming. In T. Hu, N. Lourenço, E. Medvet, and F. Divina, editors, *Genetic Programming*, pages 151–166, Cham, 2020. Springer International Publishing.

[141] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10684–10695, 2022.

[142] D. Rubin. Discussion: Statistical disclosure limitation. *Journal of Official Statistics*, 9(2):461–468, 1993.

[143] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[144] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. 1998.

[145] R. E. Schapire et al. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406. Citeseer, 1999.

[146] J. Secretan, N. Beato, D. B. D Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1759–1768, 2008.

[147] C. Sha, M. Cuperlovic-Culf, and T. Hu. Smile: systems metabolomics using interpretable learning and evolution. *BMC Bioinformatics*, 22(1):284, May 2021.

[148] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, Jul 2019.

[149] B. D. Silva and S. S. Shi. Style transfer with time series: Generating synthetic financial data, 2019.

[150] B. D. Silva and S. S. Shi. Towards improved generalization in financial markets with synthetic data generation, 2019.

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

[152] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

[153] R. L. Streit and R. L. Streit. *The poisson point process*. Springer, 2010.

[154] J. Su and G. Wu. f-VAEs: Improve VAEs with Conditional Flows. *ArXiv*, abs/1809.05861, 2018.

[155] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[156] T. Suzumura and H. Kanezashi. Anti-Money Laundering Datasets: InPlusLab anti-money laundering datadatasets. http://github.com/IBM/AMLSim/, 2021.

[157] S. Takahashi, Y. Chen, and K. Tanaka-Ishii. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527:121261, 04 2019.

[158] A. Tharwat. Linear vs. quadratic discriminant analysis classifier: a tutorial. *International Journal of Applied Pattern Recognition*, 3:145, 01 2016.

[159] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models, 2016.

[160] S. Thudumu, P. Branch, J. Jin, and J. J. Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7(1):42, Jul 2020.

[161] I. O. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf. Wasserstein auto-encoders. *CoRR*, abs/1711.01558, 2017.

[162] I. O. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf. Adagan: Boosting generative models. *Advances in neural information processing systems*, 30, 2017.

[163] J. Tomczak and M. Welling. Vae with a vampprior. In A. Storkey and F. Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1214–1223. PMLR, 09–11 Apr 2018.

[164] J. M. Tomczak and M. Welling. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

[165] J. M. Tomczak and M. Welling. Improving variational auto-encoders using convex combination linear inverse autoregressive flow. *arXiv: Machine Learning*, 2017.

[166] B. Uria, I. Murray, and H. Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[167] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 467–475, Bejing, China, 22–24 Jun 2014. PMLR.

[168] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1747–1756, New York, 2016. JMLR.org.

[169] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. van den Driessche, E. Lockhart, L. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, and D. Hassabis. Parallel WaveNet: Fast high-fidelity speech synthesis. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3918–3926. PMLR, 10–15 Jul 2018.

[170] V. Vassiliades, K. Chatzilygeroudis, and J. Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630, Aug 2018.

[171] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[172] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010.

[173] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[174] Y. Wang, L. Zhang, and J. van de Weijer. Ensembles of generative adversarial networks, corr abs/1612.00991. *arXiv preprint arXiv:1612.00991*, 2016.

[175] Z. Wang, B. Dai, D. Wipf, and J. Zhu. Further analysis of outlier detection with deep generative models. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8982–8992. Curran Associates, Inc., 2020.

[176] M. Weber, J. Chen, T. Suzumura, A. Pareja, T. Ma, H. Kanezashi, T. Kaler, C. E. Leiserson, and T. B. Schardl. Scalable graph learning for anti-money laundering: A first look. *arXiv preprint arXiv:1812.00076*, 2018.

[177] M. Wiese, L. Bai, B. Wood, and H. Buehler. Deep hedging: Learning to simulate equity option markets, 2019.

[178] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer. Quant gans: deep generation of financial time series. *Quantitative Finance*, 20(9):1419–1440, 2020.

[179] N. Wu, B. Green, X. Ben, and S. O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *CoRR*, abs/2001.08317, 2020.

[180] Z. Wu, K. E. Johnston, F. H. Arnold, and K. K. Yang. Protein sequence design with deep generative models. *Current Opinion in Chemical Biology*, 65:18–27, 2021. Mechanistic Biology * Machine Learning in Chemical Biology.

[181] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.

[182] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR.

[183] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, Y. Shao, W. Zhang, B. Cui, and M.-H. Yang. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796*, 2022.

[184] J. Yoon, L. N. Drumright, and M. van der Schaar. Anonymization through data synthesis using generative adversarial networks (ads-gan). *IEEE Journal of Biomedical and Health Informatics*, 24(8):2378–2388, 2020.

[185] J. Yoon, D. Jarrett, and M. 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.

[186] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2852–2858. AAAI Press, 2017.

[187] B. Zhu and C.-W. Ngo. Cookgan: Causality based text-to-image synthesis. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5518–5526, 2020.