Synthsonic: Fast, Probabilistic modeling and Synthesis of Tabular Data

Max Baak
ING Bank, University of Amsterdam

Simon Brugman
ING Bank

Lorraine D'almeida
ING Bank

Ilan Fridman Rojas
ING Bank

Jean-Baptiste Oger
ING Bank

Ralph Urlus
ING Bank

Abstract

The creation of realistic, synthetic datasets has several purposes with growing demand in recent times, *e.g.* privacy protection and other cases where real data cannot be easily shared. A multitude of primarily neural networks (NNs), *e.g.* Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), or Bayesian Network (BN) approaches have been created to tackle this problem, however these require extensive compute resources, lack interpretability, and in some instances lack replication fidelity as well. We propose a hybrid, probabilistic approach for synthesizing pairwise independent tabular data, called Synthsonic. A sequence of well-understood, invertible statistical transformations removes first-order correlations, then a Bayesian Network jointly models continuous and categorical variables, and a calibrated discriminative learner captures the remaining dependencies. Replication studies on MIT's SDGym benchmark show marginally or significantly better performance than all prior BN-based approaches, while being competitive with NN-based approaches (first place in 10 out of 13 benchmark datasets). The computational time required to learn the data distribution is at least one order of magnitude lower than the NN methods. Furthermore, inspecting intermediate results during the synthetic data generation allows easy diagnostics and tailored corrections. We believe the combination of out-of-the-box performance, speed and interpretability make this method a significant addition to the synthetic data generation toolbox.

Proceedings of the 25[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

1 INTRODUCTION

Existing approaches for tabular data synthesis fall broadly into four categories: GANs, VAEs, Copula methods, and BNs. These differ in the core model that fits the distributions and associations in the observed data. Relevant examples of each of these approaches are: `CTGAN` (Xu et al., 2019), `TVAE` (Xu et al., 2019), `CLBN` (Chow and Liu, 1968) and `PrivBN` (Zhang et al., 2017). We compare these using benchmark datasets in the SDGym replication tests package (Xu et al., 2019).

An important distinction between these methods is their treatment of datasets with both continuous and discrete features. Deep generative models traditionally perform well on continuous features and BNs on categorical ones. The method presented here differs in its treatment of mixed continuous-discrete features, where low-order correlations between both types are modeled with a BN by discretizing the continuous variables and joining these with the categorical ones. A calibrated, standard discriminator is used to model higher-order feature dependencies, using both the categorical and (untouched) continuous variables. Another factor where approaches differ is their computational training cost, with GANs and VAEs representing some of the most computationally expensive approaches to learn the data distribution, and BNs and Copula methods being generally more computationally efficient.

In this work, we present a novel, hybrid, probabilistic approach – called Synthsonic – which combines elements of Copula methods and BNs along with a calibrated, standard discriminator. Using a discriminator to model association in Copula space has not been studied before, as far as we are aware. The guid-

ing principle is to remove all obvious structure before modelling the remaining non-linear correlations. Synthsonic ultimately produces results competitive with or outperforming the best of all methods, on both continuous and discrete feature sets, at a computational cost to fit the model slightly higher than BNs alone but significantly lower than that of GANs or VAEs.

An additional benefit of our method is that, unlike GANs, every intermediate step can be visualized and debugged where necessary, making the overall procedure interpretable throughout.

2 BACKGROUND & RELATED WORK

The creation of synthetic datasets which resemble observed datasets is an endeavour that dates back at least as far as work on Statistical Disclosure Limitation and Statistical Disclosure Control, with the use of fully synthetic data first proposed in Rubin (1993) (see Matthews and Harel (2011) for a review). The objective of this work was to satisfy privacy requirements by using alterations, aggregations, or total replacement of an observed dataset, analogous to the current day efforts in differential privacy, for example as used in the US 2020 Census (Abowd et al., 2020).

Current day high-capacity models which can mimic realistic image data, such as GANs (*e.g.* Karras et al. (2019)), along with the ever-growing need for datasets that may be privacy-sensitive or difficult to move or copy, creates opportunity to further expand on this line of work. The approach of generating fully synthetic datasets has been advocated for in multiple fields of study, including finance (Assefa, 2020; Da Silva and Shi, 2019), medicine (Goncalves et al., 2020; Yale et al., 2019; Choi et al., 2017), and economics (Koenecke and Varian, 2020) to name a few.

Another well-known use for synthetic data is data augmentation to supplement observed training data. This can be both in the form of using synthetic data generation to oversample minority class examples in imbalanced class problems (Camino et al., 2020), or to augment the training data more broadly (Meyer et al., 2021).

Conceptually, the two recent, existing methods closest to work presented here are the Copula Flows (Kamthe et al., 2021) and the pre-transformation Variational Autoencoder (`PTVAE`) (Farhadyar et al., 2021) approaches. Similar to our approach, the former relies on probability integral transform methods for part of the training. The latter makes of transformations to obtain approximately normally-distributed data prior to learning finer aspects of the data. For this later stage, a VAE is used, whereas we rely on a BN and calibrated discriminator.

Lastly, extracting a probability density model from a classifier has recently been studied in Grathwohl et al. (2020), however our concept to model the Copula space is rather different.

3 SYNTHSONIC DATA MODEL

Synthsonic builds a probabilistic model describing any pairwise independent tabular input dataset X. X can consist of both categorical and continuous features[1], and is split beforehand into categorical and continuous sets: $X = (X_{\text{cat}}, X_{\text{num}})$.

The modeling takes place in four steps:

1. Continuous features are transformed into uniformly distributed ones, using invertible steps, from which the first-order correlations have been removed.
2. Categorical and discretized continuous features are jointly modeled with a Bayesian network.
3. A discriminative learner is trained to model the residual discrepancies observed between the input data and the Bayesian network, using the (not discretized) continuous and categorical features.
4. The discriminative learner is calibrated to reweight synthetic data from the Bayesian network.

Details on each building step are provided in the following subsections. This section concludes with a description of the joint probabilistic model and sampling procedure.

An optional feature selection step can occur before the discriminative learning. This can be beneficial when dealing with large numbers of features that not all need to be considered by the learner. The procedure is described in the supplementary material in Section C.

3.1 Transformations of continuous features

The continuous features are transformed into uniform distributions in three invertible steps, frequently encountered in Copula methods (Durante and Sempi, 2016; Jaworski et al., 2010). Fig. 1a-d illustrates the transformation steps.

1. Each individual feature gets transformed into a normal distribution using a quantile transformation (Pedregosa et al., 2011), resulting in dataset X_{normal}. Per feature non-unique continuous values,

[1] Synthsonic treats ordinal features as categorical.

e.g. often present in integer-based features, get whitened beforehand by adding minuscule white noise, ensuring a smooth quantile transformation[2].

The motivation for this transformation is that the PCA rotation we apply (next step) works best for normal distributions (Pearson, 1901). The inverse Jacobian of this transformation (Durante and Sempi, 2016) is:

$$\prod_j \frac{f_j(X_{\text{num}}[j])}{G(X_{\text{normal}}[j])}\,, \tag{1}$$

where the product runs over all continuous features j, $G(X)$ is a normal density distribution, and $f_j(X_{\text{num}}[j])$ is the marginalized probability density function of feature $X_{\text{num}}[j]$, as can be extracted from its quantile transformation.

2. Using PCA (Pearson, 1901; Pedregosa et al., 2011) the linear correlations between the normalized features are removed by rotating these to the principal component's frame of their covariance matrix, leading to X_{pc}. Note that the principal components do not follow normal distributions exactly when non-linear dependencies are present. The Jacobian of the rotation equals 1.

3. Each individual feature in X_{pc} gets transformed into a uniform distribution with range $(0, 1)$, using another quantile transformation, giving X_{uniform}. The inverse Jacobian of this transformation is:

$$\prod_j k_j(X_{\text{pc}}[j])\,, \tag{2}$$

where $k_i(X_{\text{pc}}[i])$ is the marginalized probability density function of feature $X_{\text{pc}}[j]$, again obtained from the quantile transformation.

Any non-linear dependencies between the continuous features in X_{num} are still present in X_{uniform} after these transformations. There are no tunable parameters for these steps, besides the number of quantiles (500 by default).

3.2 Bayesian network model

Each uniform, continuous feature get discretized into n equal-width bins in the range $[0, 1]$ and assigned a bin index (we use $n = 30$). This results in a dataset X_{discrete}, which gets joined with the categorical features as $X_{\text{bn}} = (X_{\text{cat}}, X_{\text{discrete}})$. X_{bn} is fitted using a standard, tree-based Bayesian network, which serves to describe the largest dependencies between the categorical features themselves, the categorical and continuous features, and residual dependencies between the continuous features. Fig. 1e illustrates the learned weights on the ring dataset, using only 10 bins to improve readability.

Our framework is not restricted to a single structure learner. In our experiments, for reasons of speed, the structure learning is kept simple using the well-established Chow-Liu (CL) algorithm as described in Chow and Liu (1968), with the Tree Augmented Naive Bayes (TAN) extension described in Friedman et al. (1997). The CL algorithm computes the mutual information of all feature pairs as edge weights for a complete graph, from which the maximum-weight spanning tree is computed (Chow and Liu, 1968). The TAN extension augments the structure with edges from a designated class node to all other nodes (Friedman et al., 1997) for more robust estimation of that class node. This class node can be provided, or detected automatically from the aforementioned edge weights, *e.g.* by using the node with the highest average edge weights.

As the CL algorithm only considers bivariate relations, in general the structure is less prone to overfitting than those that consider higher-order dependencies.

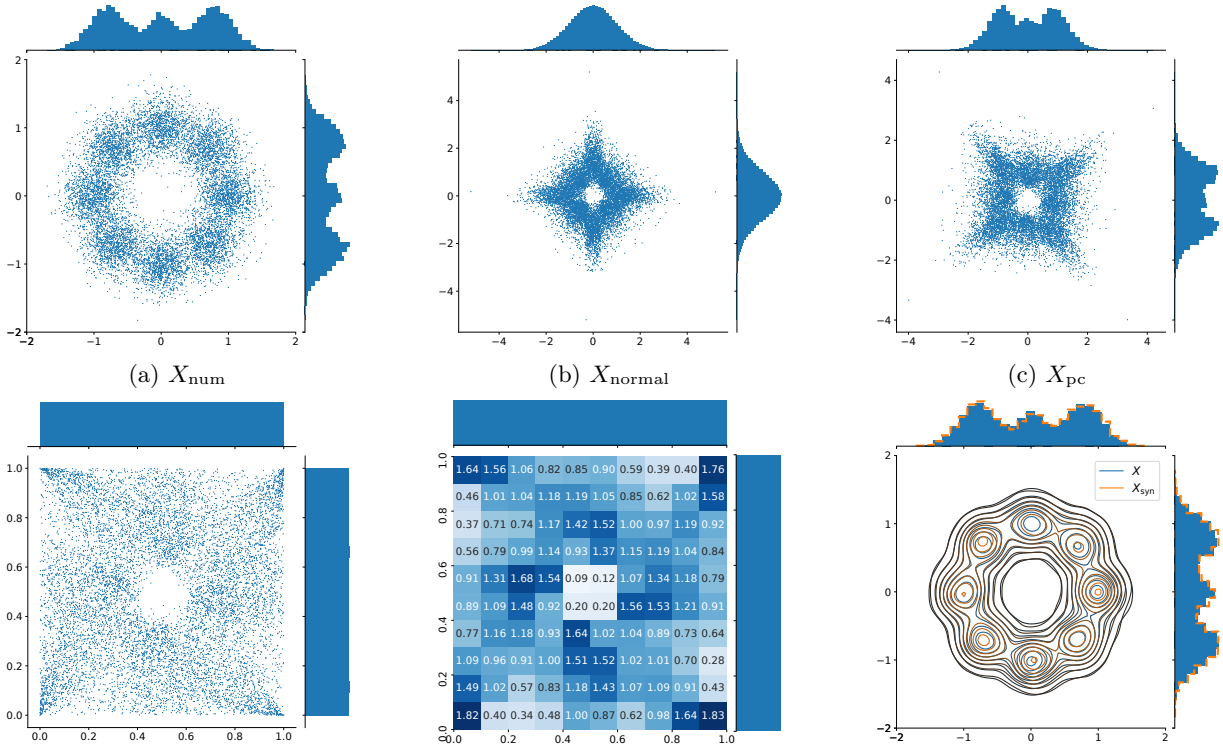
Because of the combination of the transformation steps applied to the continuous features, plus the Bayesian network used for both feature types, we call this the Copula Bayesian network.

Note that the information available in the continuous features is not discarded in the discretization step: it is used actively in the feature transformations and the discriminative learning stage below.

3.3 Discriminative learning

We define $X_{\text{trans}} = (X_{\text{cat}}, X_{\text{uniform}})$. Using a fixed random seed, X_{trans} is divided into training and calibration datasets, having a 70/30% split by default, and a synthetic dataset is generated from the Bayesian network model, having the same size as the training set. Unlike X_{trans}, the synthetic dataset contains categorical and discrete continuous features only. Each discretized, continuous feature gets converted to a uniform distribution by adding uniformly distributed random values – in the range $(0, \frac{1}{30})$ in case of 30 bins – to the bin indices. This undiscretization step is logical for the synthetic dataset as by construction all continuous features in X_{trans} also have uniform, marginalized distributions. At this point, X_{trans} and the synthetic sample, X_{syn}, are in a comparable state, although X_{trans} may contain higher-order feature dependencies not modeled by the Copula BN.

[2]The variance of the white noise is the range of a feature scaled with a (configurable) factor of 10^{-5}. Integer values can later be recovered by rounding the transformed features.



(a) X_{num} (b) X_{normal} (c) X_{pc}

(d) X_{uniform} (e) BN weights X_{discrete} (f) X_{num} and X_{syn}

Figure 1: Numerical transformations example on the ring dataset (a-d), (e) the Copula Bayesian network's weights for discretized X_{uniform} and (f) a synthetic sample.

Class label $y = 1$ is assigned to all data points in X_{trans} and $y = 0$ to those in X_{syn}. A standard non-linear, binary classifier is trained to discriminate between data points from the two training samples. By default, we use the gradient boosted tree classifier from `XGBoost` (Chen and Guestrin, 2016). If the Copula BN has managed to model the input data well, then the classifier should find only a slight separation between the two datasets. If non-linear and/or higher-order feature dependencies are not fully captured, the classifier should zoom in on precisely those differences. Note that a non-linear classifier is recommended here, as the linear correlations have already been modeled by now.

Synthsonic relies on being able to describe the data reasonably well with the Copula BN, such that the discriminative learner finds sufficient overlap between X_{trans} and X_{syn}. If the two datasets are perfectly separable by the classifier, a better initial model is needed before proceeding further. When some overlap is found however, a better model can be formed.

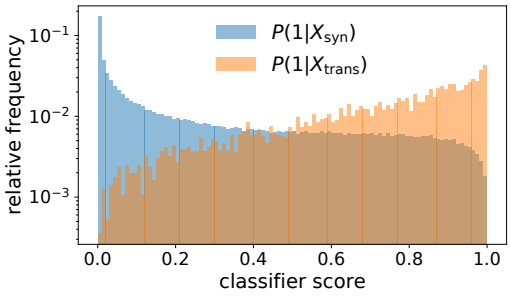
The discriminative learner is used to improve the Copula BN by weighting data points as a function of their classifier scores. Low classifier scores generally indicating non-similarity with the input data are to be suppressed, and high classifier scores, indicating (strong) compatibility, are to be encouraged. If the classifier is well-calibrated (see Sec. 3.4), its scores can be used to form a weight function that transforms data points from X_{syn} into X_{trans}:

$$w(x) = \frac{P(y=1|x)}{P(y=0|x)}\,, \tag{3}$$

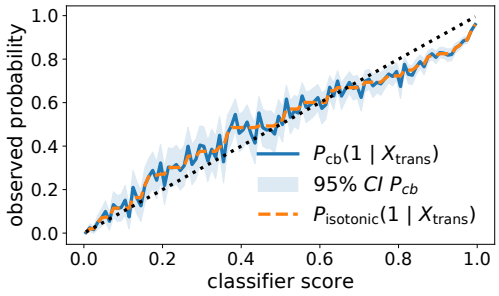
where $P(y = 1|x)$ is the predicted probability of a data point x to belong to the input data, and $P(y = 0|x) = 1 - P(y = 1|x)$ the probability that it originated from the synthetic data in our binary setup.

3.4 Calibration procedure

The weight function in Eqn. 3 can be interpreted as a probability density function describing the Copula space (Durante and Sempi, 2016; Jaworski et al., 2010). It needs probabilities as input, but out-of-the-box classifier scores (*e.g.* $P(y = 1|x)$) cannot be interpreted as such. Besides the possibility of overfitting the trained classifier, there is a second reason to calibrate it. Fig. 2a shows the two classifier score distributions of an example benchmark dataset, as obtained on the hold-out calibration dataset and another generated synthetic dataset. (This synthetic dataset can be arbitrarily large, by default 250k entries.) Fig. 2b shows the



(a) distribution of classifier scores



(b) calibration curve

Figure 2: Calibration score distributions on the Adult dataset (Xu et al., 2019). $P(1|X_{\text{trans}})$, $P(1|X_{\text{syn}})$ are the distributions obtained over all data points in, respectively, X_{trans} and X_{syn}.

classifier's calibration curve, P_{cb}, obtained from the normalized distributions in Fig. 2a, defined as the ratio of class 1 over the sum of both classes (Zadrozny and Elkan, 2002). The calibration curve represents the observed probability, as a function of classifier score, for any data point to belong to the input data. A perfectly calibrated classifier has a diagonal calibration curve, *i.e.* the classifier score and observed probability agree. If there is little to no separation between both classes, the calibration curve will be (nearly) flat over the domain. For example, a dataset containing only linear correlations results in a flat calibration curve at 0.5, as the Copula BN can model such first-order correlations.

The general assumption is made that higher classifier scores correspond to higher probabilities, and that any deviations from this are caused by statistical fluctuations, *e.g.* as visible in Fig. 2b. We therefore model the calibration curve with a monotonically rising step-function, specifically isotonic regression (Chakravarti, 1989). The input calibration curve is obtained from normalized classifier-score histograms. By default, the number of bins used is the maximum of Sturges' formula (Sturges, 1926) and the Freedman Diaconis Estimator (Freedman and Diaconis, 1981; Harris et al., 2020). We assign a sample weight to each bin that reflects the statistical, binomial uncertainty on each observed probability. Isotonic regression groups adjacent bins with statistically compatible probabilities, and assigns an average probability to each group.

As probability function in Eqn. 3 we use:

$$P_{\text{isotonic}}(1|x) = \texttt{isotonic\_regressor}(\texttt{classifier\_score}(x))\,. \tag{4}$$

Note that a flat probability is assigned across any bin, irrespective of the precise classifier score within that bin, and no additional (*e.g.* linear) interpolation is applied between different bins. In particular, this regulates the highest bin and tends to keep its assigned probability smaller than 1, keeping the corresponding weight value finite. For example, the synthetic data points in Fig. 2b get assigned a maximum weight of 20.8.

3.5 Joint probabilistic model

The steps in Sections 3.1-3.4 are combined here to form a joint probabilistic model describing input dataset X.

Using Sklar's theorem (Sklar, 1959) and the inverse Jacobians in Eqns. 1&2, the probability density function of only the continuous features is written as:

$$f(X_{\text{num}}) = \left[\prod_j f_j(X_{\text{num}}[j]) \cdot \frac{k_j(X_{\text{pc}}[j])}{G(X_{\text{normal}}[j])}\right] \cdot c(X_{\text{uniform}})\,, \tag{5}$$

where $c(X_{\text{uniform}})$ is the probability density function describing the Copula space. If no more dependencies are left after the initial feature transformations then $c(X_{\text{uniform}}) = 1$.

$c(X_{\text{uniform}})$ is modeled by both the Bayesian network and the weight function of the discriminator in Eqn. 3. Decompose the probability mass function (p.m.f.) of the Bayesian network as:

$$P_{\text{bn}}(X_{\text{bn}}) = P(X_{\text{cat}}|X_{\text{discrete}}) \cdot P(X_{\text{discrete}})\,, \tag{6}$$

where $P(X_{\text{discrete}})$ is the p.m.f. of only X_{discrete}. Considering only the Bayesian network, then:

$$\begin{aligned} c(X_{\text{uniform}}) &= \frac{P(X_{\text{discrete}})}{P_{\text{none}}(X_{\text{discrete}})} \\ &= P(X_{\text{discrete}}) \cdot n_{\text{bins}}^{N_{\text{num}}}\,, \end{aligned} \tag{7}$$

where n_{bins} is the number of bins per discretized, continuous feature, and $P_{\text{none}}(X_{\text{discrete}}) = n_{\text{bins}}^{-N_{\text{num}}}$ is the p.m.f. in case of no dependencies between any of the N_{num} continuous features.

The classifier weight applies to both categorical and continuous features. Putting all components together

then results in the joint probabilistic model:

$$p(X) = \left[\prod_j f_j(X_{\text{num}}[j]) \cdot \frac{k_j(X_{\text{pc}}[j])}{G(X_{\text{normal}}[j])}\right] \cdot n_{\text{bins}}^{N_{\text{num}}} \cdot P_{\text{bn}}(X_{\text{bn}}) \cdot w(X_{\text{trans}}), \quad (8)$$

where the categorical (continuous) features are modeled with a mass (density) function.

3.6 Sampling procedure

A five-step sampling procedure is used to produce synthetic datasets X_{syn}:

1. Synthetic data are generated from the Bayesian network model, using standard, forward sampling over the topological structure of the network tree (Guo and Hsu, 2002). This results in a dataset similar to X_{bn}.
2. Each discretized, continuous feature gets converted to a uniform distribution by adding uniformly distributed random values – in the range $(0, \frac{1}{30})$ for 30 bins – to the bin indices. This gives a dataset comparable to X_{trans}.
3. Classifier weights are assigned to all data points using Eqn. 3. Extremely high weights are problematic, as they are used for oversampling (next step). The maximum weight is capped at a (configurable) value of 5000. Minimum weights do not need to be capped.
4. We have chosen a fast weighting method to reshape the synthesized dataset: weighted sampling with replacement (Wong and Easton, 1980). In short, generated samples are dropped or duplicated to match the assigned weights.
5. The inverse transformations of the ones described in Section 3.1 are applied to all continuous features. At this stage, the synthetic dataset resembles the original dataset X, with the same boundaries per feature.

In practice, we find the level of duplication from weighted sampling to be relatively small, although the value differs per dataset. For weight-based sampling we can alternatively use the accept-reject method, without replacement, but in practice this can be slow when large maximum weight values are present.

4 INTERPRETABILITY & VALIDATION

The transformations of X_{num} to X_{uniform} in Copula space are well-studied (Sklar, 1959; Jaworski et al., 2010; Durante and Sempi, 2016) and can be inspected using goodness-of-fit tests (Berg, 2013) or visually such as in Fig. 1a-d. Similarly, multiple methods have been developed for the interpretation of BNs (Timmer et al., 2017; Chubarian and Turán, 2020), and the learned dependencies can be visualized as in Fig. 1e. The interpretability of the discriminative learning step is dependent on the choice of classifier. For the purpose of data synthesis, the classifier score distributions and calibration curve in Fig. 2 provide a reasonable estimate if the classifier is fit for the purpose.

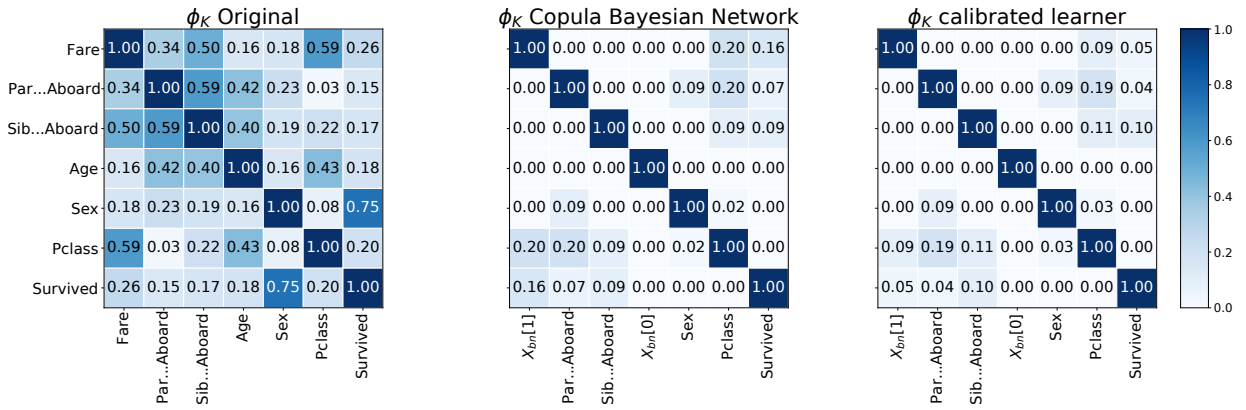
In order to understand how well non-linear correlations between variables are modeled in Synthsonic, we can inspect the ϕ_K correlation matrix (Baak et al., 2020). The ϕ_K correlation constant supports categorical and continuous features and captures non-linear dependencies, with a value in the range $[0, 1]$. As input, it takes two contingency tables for each feature pair of the true and generated data, where near-identical (highly different) tables result in a ϕ_K value close to zero (one). In Synthsonic we can make this comparison at any intermediate step in the modeling process.

In Fig. 3a-c, in the off-diagonal matrix elements, we observe the non-linear correlations present in the Titanic dataset (Kaggle, 2012), after learning the Copula Bayesian network model and after learning and applying the calibrated discriminator. (The Titanic dataset was chosen for its small number of features allowing for easy visualisation, while containing a mixture of continuous and categorical features.) It is clear that the levels of residual dependencies get smaller with both modeling steps, reaching close to the identity matrix – signifying no residual feature dependencies. For the Titanic dataset, the dependencies are primarily resolved by the Copula Bayesian network, and to a smaller extent by applying the weights from the calibrated discriminator. In general however, the level of dependency reduction – and how much by which step – is highly dataset dependent.

5 PERFORMANCE EXPERIMENTS

To evaluate the performance of Synthsonic, we use the benchmark setup and metrics developed by SDGym (Xu et al., 2019), version 2.2.0, consisting of 7 artificial and 6 real-world datasets.[3] The three sets of experiments performed are Gaussian Mixture simulations (GM Sim.), Bayesian Network simulations (BN Sim.) and classification and regression problems on simulations of the real-world datasets.

[3]The MNIST datasets are not considered as the rows are not pairwise independent, similar to Xu et al. (2019); Leduc and Grislain (2021).



(a) X (b) X_{bn} (c) $P(X_{\text{bn}})$

Figure 3: ϕ_K correlation matrices capturing the structure remaining after each modeling step on the Titanic dataset. $X_{\text{bn}}[0]$ and $X_{\text{bn}}[1]$ are discretized representations of 'Fare' and 'Age' features. The diagonal values equal 1 by construction.

We compare against the algorithms `CTGAN` and `TVAE`, as these are top-ranking on the benchmark, and with `PrivBN` and `CLBN`, both powered by a Bayesian Network and closest to our approach. For example, the `CLBN` baseline (Chow and Liu, 1968) is the structure learner used for Synthsonic's Copula BN, albeit a different implementation. For these algorithms, the default parameters from SDGym are used to conduct the experiments. Table 1 shows the aggregated replication scores for all compared algorithms on the data synthesis tasks. For precise definitions of the metrics used we refer to Xu et al. (2019). (A discussion of per-dataset results can be found in the supplementary material.) A good algorithm fits two criteria: the likelihood L_{syn} should be close to L_{test}, as this indicates generalization, and L_{test} should be close to the `identity` dataset (c.f. Table 2 in Xu et al. (2019)). Based on this, Synthsonic outperforms all other algorithms on the artificial datasets. For the real datasets, Synthsonic scores second place on the classification problems (clf), based on average $F1$ score, and first place on the regression problem (reg), ranked on R^2 score.

Per dataset, Synthsonic ranks first place on 10 out of 13 datasets. It is noteworthy that Synthsonic performs relatively poorly on the highly-imbalanced `credit` dataset. In practice, this could be easily resolved by specifying the target label as the class node of the TAN.[4]

Table 2 shows the average fit and sample times of Synthsonic, compared with the other algorithms.[5] (See the supplementary material for the results per dataset.) Note that the implementation of `PrivBN` does not allow for distinguishing between fit and sample times. Synthsonic efficiently samples using an optimized forward-sampling technique implemented in `pgmpy` (Ankan and Panda, 2015).

In comparison, `CLBN` uses a quite slow sampling technique. Compared with GPU times, on average Synthsonic fits at least 19 times faster than `TVAE` and 29 times faster than `CTGAN`; both are computationally expensive.

To understand the effectiveness of individual components of Synthsonic, we perform an ablation study on the following parts on each of SDGym's benchmark datasets: without PCA (sec 3.1, step 2), without classifier (sec 3.4), or without calibration (sec 3.5). Also shown in Table 3 are the default and optimized settings.

Synthsonic's computational efficiency allows for the user to tune the handful of parameters on any given dataset. Chosen parameters are listed in the supplementary material. Each optimized configuration is replicated 3 times with different random seed values.

Note that Synthsonic gives a good out-of-the-box performance. The most significant improvements to the Copula BN come from adding PCA, the classifier and its calibration.

6 LIMITATIONS

Although Synthsonic shows performance results on the SDGym benchmark datasets marginally to significantly better than all prior BN-based approaches, whilst also being competitive with neural-net-based

[4] Modeling the data per target class gives an $F1$ score of 0.580 for the `credit` dataset, and would give an average $F1$ score of 0.539.

[5] All CPU experiments have been run on 6-Core Intel Core i7 with 16 GB RAM. GPU experiments have been run on Google Colab instances, with Nvidia K80s, T4s, P4s and P100s.

Table 1: Performance of Synthsonic on artificial datasets, compared against top-performers on SDGym leaderboard v0.2.2. The top scores are printed in bold. For the definitions of L_{syn} and L_{test} see (Xu et al., 2019) or the appendix.

	GM. Sim		BN. Sim		Real	
Method	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	clf F_1	reg R^2
`Identity`	−2.93	−2.94	−9.34	−9.39	0.638	0.14
`CLBN` (Chow and Liu, 1968)	−3.22	−21.3	−10.67	−9.92	0.349	−6.47
`PrivBN` (Zhang et al., 2017)	−3.29	−5.71	−10.38	−9.80	0.257	−4.49
`TVAE` (Xu et al., 2019)	−2.91	−3.77	−10.12	−9.89	0.398	−0.24
`CTGAN` (Xu et al., 2019)	−7.89	−4.26	−12.86	−10.84	**0.491**	−0.07
`Synthsonic`	−2.97	**−2.97**	−9.65	**−9.57**	0.455	**0.02**

Table 2: Efficiency of Synthsonic on real datasets, compared against top-performers on SDGym leaderboard v0.2.2. Reported times (sec) are averages over the six datasets, unless otherwise mentioned. Table 9 contains the per-dataset measurements.

	Time (s)			
	CPU		GPU	
Method	fit	sample	fit	sample
`CLBN`	126	232	-	-
`PrivBN`	16725		-	
`TVAE`	17173	4	4082	4
`CTGAN`	45462	16	7595	10
`Synthsonic`	214	17	-	-

approaches, the characteristics of these datasets should be considered. MIT's SDGym benchmark is limited to 13 datasets that are relatively low-dimensional (≤ 59 dimenstions, excluding MNIST). As such, the performance on higher dimensional data has not yet been tested. Synthsonic's performance is sensitive to the automatic root-node selection of the BN on datasets with severe class imbalance. Manual selection of, for example, the target label as class node of the TAN restores performance to very reasonable levels. Datasets that are not pairwise independent over the rows, *e.g.* images or time-series data, have not been considered, although we believe that it is possible to extend Synthsonic to relax this constraint. Additionally, the initial fit of the Copula BN must be good enough such that the data sampled from it cannot be fully separated by the discriminative learner; only then can the classifier learn to transform synthetic into realistic data.

7 CONCLUSIONS

We have proposed a hybrid, probabilistic approach for the synthesis of tabular data called Synthsonic, incorporating a novel concept to model the Copula space. By combining transformations to Copula space, a simple Bayesian Network and a calibrated discriminative learner, we can describe tabular datasets using a probabilistic representation that jointly models continuous and categorical variables. We have shown that Synthsonic is competitive with the performance of GANs and VAEs on the SDGym benchmark datasets, scoring first place in 10 out of 13 datasets, at a much lower computational cost. Interpretable intermediate representations of Synthsonic's probabilistic model allow easy application of diagnostics and tailored corrections to guard against the above-mentioned considerations. We believe the combination of out-of-the-box performance, speed, and interpretability make this method an excellent addition to the synthetic data generation toolbox.

BROADER IMPACT

Where synthetic data generation is used to share synthetic versions of privacy-sensitive data, special care must be taken in ensuring no unwanted disclosure takes place, *e.g.* through overfitting the original data (Jordon et al., 2018). Where the use case is data augmentation, practitioners must be aware of the limitation of synthetic data generators, *e.g.* the modeling does not account for the tails of distributions or any data points beyond the support of the training data. Synthetic data generation should also not be misused to whitewash biases in the observed data, on the misunderstanding that synthetic data must somehow be more objective.

Table 3: Ablation study results ($\mathcal{L}_{test}$ for simulations, F_1 for all-but-one real datasets, R^2 for the news dataset. Higher is better). The right column shows the absolute performance of our model. The optimized configuration is replicated 3 times with different random seed values; the average metric score and Mean Absolute Deviation (MAD) are reported. The other columns contain the performance change relative to that. Any differences exceeding two times the MAD are marked in bold.

Dataset \ Model	w/o PCA	w/o clf	w/o calibration	w/o tuning	Synthsonic
Grid	−0.00	0.05	**−0.10**	0.05	-3.50 ± 0.04
Gridr	0.00	0.02	−0.02	−0.01	-3.67 ± 0.03
Ring	0.00	−0.00	**−0.05**	−0.01	-1.71 ± 0.00
Asia	0.01	−0.01	0.00	−0.00	-2.25 ± 0.01
Alarm	0.02	**−0.54**	**−0.73**	0.01	-10.64 ± 0.02
Child	−0.00	**−0.09**	**−0.94**	−0.00	-12.19 ± 0.01
Insurance	−0.01	**−0.60**	**−0.87**	−0.02	-13.22 ± 0.01
Adult	−0.01	−0.01	**−0.03**	**−0.04**	0.62 ± 0.00
Census	**0.03**	**−0.14**	−0.01	−0.01	0.39 ± 0.01
Credit	**−0.16**	−0.07	**−0.17**	−0.09	0.17 ± 0.06
Covtype	−0.01	**−0.08**	−0.00	**−0.04**	0.43 ± 0.01
Intrusion	**−0.23**	0.01	0.02	**−0.06**	0.65 ± 0.01
News	**−0.09**	0.00	−0.02	**−0.25**	0.02 ± 0.01

References

Abowd, J. M., Benedetto, G. L., Garfinkel, S. L., Dahl, S. A., Dajani, A. N., Graham, M., Hawes, M. B., Karwa, V., Kifer, D., Kim, H., LeClerc, P., Machanavajjhala, A., Reiter, J. P., Rodriguez, R., Schmutte, I. M., Sexton, W. N., Singer, P. E., and Vilhuber, L. (2020). The modernization of statistical disclosure limitation at the us census bureau.

Ankan, A. and Panda, A. (2015). pgmpy: Probabilistic graphical models using python. In *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*. Citeseer.

Assefa, S. (2020). Generating synthetic data in finance: opportunities, challenges and pitfalls. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Workshop on AI in Financial Services*.

Baak, M., Koopman, R., Snoek, H., and Klous, S. (2020). A new correlation coefficient between categorical, ordinal and interval variables with pearson characteristics. *Computational Statistics & Data Analysis*, 152:107043.

Berg, D. (2013). Copula goodness-of-fit testing: an overview and power comparison. *Copulae and Multivariate Probability Distributions in Finance*, pages 79–106.

Camino, R., Hammerschmidt, C., et al. (2020). Oversampling tabular data with deep generative models: Is it worth the effort? In *"I Can't Believe It's Not Better!"NeurIPS 2020 workshop*.

Chakravarti, N. (1989). Isotonic median regression: a linear programming approach. *Mathematics of operations research*, 14(2):303–308.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Choi, E., Biswal, S., Malin, B., Duke, J., Stewart, W. F., and Sun, J. (2017). Generating multi-label discrete patient records using generative adversarial networks. In Doshi-Velez, F., Fackler, J., Kale, D., Ranganath, R., Wallace, B., and Wiens, J., editors, *Proceedings of the 2nd Machine Learning for Healthcare Conference*, volume 68 of *Proceedings of Machine Learning Research*, pages 286–305, Boston, Massachusetts. PMLR.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467.

Chubarian, K. and Turán, G. (2020). Interpretability of bayesian network classifiers: Obdd approximation and polynomial threshold functions. In *ISAIM*.

Da Silva, B. and Shi, S. S. (2019). Style transfer with time series: Generating synthetic financial data. *arXiv preprint arXiv:1906.03232*.

Durante, F. and Sempi, C. (2016). *Principles of copula theory*, volume 474. CRC press Boca Raton, FL.

Farhadyar, K., Bonofiglio, F., Zoeller, D., and Binder, H. (2021). Adapting deep generative approaches

for getting synthetic data with realistic marginal distributions. *arXiv preprint arXiv:2105.06907*.

Freedman, D. and Diaconis, P. (1981). On the histogram as a density estimator: L 2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 57(4):453–476.

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning*, 29(2):131–163.

Goncalves, A., Ray, P., Soper, B., Stevens, J., Coyle, L., and Sales, A. P. (2020). Generation and evaluation of synthetic patient data. *BMC medical research methodology*, 20:1–40.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., Norouzi, M., and Swersky, K. (2020). Your classifier is secretly an energy based model and you should treat it like one. In *International Conference on Learning Representations*.

Guo, H. and Hsu, W. (2002). A survey of algorithms for real-time bayesian network inference. In *Join Workshop on Real Time Decision Support and Diagnosis Systems*.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Jaworski, P., Durante, F., Hardle, W. K., and Rychlik, T. (2010). *Copula theory and its applications*, volume 198. Springer.

Jordon, J., Yoon, J., and Van Der Schaar, M. (2018). Pate-gan: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*.

Kaggle (2012). Titanic - machine learning from disaster. `https://www.kaggle.com/c/titanic/data`. Accessed on 22 May 2021.

Kamthe, S., Assefa, S., and Deisenroth, M. (2021). Copula flows for synthetic data generation. *arXiv preprint arXiv:2101.00598*.

Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410.

Koenecke, A. and Varian, H. (2020). Synthetic data generation for economists. In *American Economic Association Annual Meeting*.

Leduc, J. and Grislain, N. (2021). Composable generative models. *arXiv preprint arXiv:2102.09249*.

Matthews, G. J. and Harel, O. (2011). Data confidentiality: A review of methods for statistical disclosure limitation and methods for assessing privacy. *Statistics Surveys*, 5(none):1 – 29.

Meyer, D., Nagler, T., and Hogan, R. J. (2021). Copula-based synthetic data generation for machine learning emulators in weather and climate: application to a simple radiation model. *Geoscientific Model Development Discussions*, 2021:1–21.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

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

Rubin, D. B. (1993). Statistical disclosure limitation. *Journal of official Statistics*, 9(2):461–468.

Sklar, M. (1959). Fonctions de repartition an dimensions et leurs marges. *Publ. inst. statist. univ. Paris*, 8:229–231.

Sturges, H. A. (1926). The choice of a class interval. *Journal of the american statistical association*, 21(153):65–66.

Timmer, S. T., Meyer, J.-J. C., Prakken, H., Renooij, S., and Verheij, B. (2017). A two-phase method for extracting explanatory arguments from bayesian networks. *International Journal of Approximate Reasoning*, 80:475–494.

Wong, C.-K. and Easton, M. C. (1980). An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113.

Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems*, 32:7335–7345.

Yale, A., Dash, S., Dutta, R., Guyon, I., Pavao, A., and Bennett, K. P. (2019). Assessing privacy and quality of synthetic health data. In *Proceedings of the Conference on Artificial Intelligence for Data Discovery and Reuse*, pages 1–4.

Zadrozny, B. and Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699.

Zhang, J., Cormode, G., Procopiuc, C. M., Srivastava, D., and Xiao, X. (2017). Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41.

Supplementary Material: Synthsonic: Fast, Probabilistic modeling and Synthesis of Tabular Data

A EXTENDED EXPERIMENT RESULTS

This section presents a breakdown of the performance of each synthetic data generator by SDGym benchmark dataset.

A.1 Synthetic data fidelity by dataset

In addition to Table 1 above, which shows performance aggregated over all synthetic or real datasets, the tables below show a more granular breakdown by dataset type and per individual dataset. Table 4 shows a breakdown of performance by dataset over the Gaussian Mixture artificial datasets, and Table 5 shows performance over the remaining artificial datasets.

As defined in Xu et al. (2019), values of $\mathcal{L}_{syn}$ and $\mathcal{L}_{test}$ closer to zero are better. $\mathcal{L}_{syn}$ is defined by evaluating the generated synthetic dataset under the likelihood which generated the training dataset. This metric is therefore prone to favouring overfitting. $\mathcal{L}_{test}$ is in turn defined by evaluating the likelihood of the test data under the likelihood which takes as ground truths the parameters fitted on the synthetic data. This latter metric therefore provides a metric for how likely the test data is under a model which uses parameters fitted from the synthetic data used as ground truth. The row labelled 'Identity' shows the score expected if the synthetic data is a copy of the training data. Scores on $\mathcal{L}_{syn}$ near or even higher than this value are a clear sign of likely overfitting when the corresponding $\mathcal{L}_{test}$ score is poor.

Table 4 shows that Synthsonic outperforms competing models on all the Gaussian Mixture artificial datasets. From Table 5 we see that Synthsonic also outperforms other models on both $\mathcal{L}_{syn}$ and $\mathcal{L}_{test}$ on all but two datasets, where it is either tied or marginally worse than the best model. The differences between Synthsonic and the runner up can be marginal in places, however this must be judged in conjunction with computational cost, which we discuss in Section A.2.

Table 4: Performance of Synthsonic benchmarked on the Gaussian Mixture artificial datasets of SDGym. Values closest to identity are marked bold. These are the leaderboard results version 0.2.2 from (Xu et al., 2019), with Synthsonic and PrivBN scores added. PrivBN was computed by us since it was missing from the original v0.2.2 leaderboard.

	`grid`		`gridr`		`ring`	
Method	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$
Identity	−3.47	−3.49	−3.59	−3.64	−1.71	−1.70
CLBN (Chow and Liu, 1968)	−3.88	−9.20	−4.01	−7.43	−1.77	−47.2
PrivBN (Zhang et al., 2017)	−3.99	−8.31	−4.07	−7.12	−1.81*	−29.95*
TVAE (Xu et al., 2019)	−3.27	−5.66	−3.87	−3.71	−1.58	−1.94
CTGAN (Xu et al., 2019)	−8.76	−5.06	−8.31	−5.05	−6.59	−2.67
Synthsonic	**−3.42**	**−3.50**	**−3.79**	**−3.67**	**−1.70**	**−1.71**

Table 5: Performance of Synthsonic benchmarked on the Bayesian Network artificial datasets of SDGym.

	`asia`		`alarm`		`child`		`insurance`	
Method	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$
Identity	−2.24	−2.24	−10.23	−10.30	−12.03	−12.04	−12.85	−12.96
CLBN	−2.40	−2.27	−12.46	−11.19	−12.63	−12.31	−15.17	−13.92
PrivBN	**−2.29**	**−2.24**	−12.15	−11.14	−12.36	−12.19	−14.70	−13.64
TVAE	**−2.29**	−2.27	−11.44	−10.76	−12.46	−12.30	−14.30	−14.24
CTGAN	−4.19	−2.46	−15.88	−13.10	−14.35	−12.84	−17.03	−14.97
Synthsonic	**−2.29**	−2.25	**−10.64**	**−10.62**	**−12.30**	**−12.19**	**−13.39**	**−13.22**

Table 6 shows an equivalent breakdown of the aggregate performances over the real datasets, previously aggregated over all real datasets in Table 1, using F_1 scores for classification-problem datasets and R^2 for the regression-problem dataset.

Table 6: Performance of Synthsonic benchmarked on the real datasets of SDGym. These are the leaderboard results version 0.2.2 from (Xu et al., 2019), with Synthsonic and PrivBN scores added. PrivBN was computed by us since it was missing from the original v0.2.2 leaderboard.

Method	`adult` F_1	`census` F_1	`credit` F_1	`covtype` Macro F_1	`intrusion` Macro F_1	`news` R^2
Identity	0.66	0.46	0.55	0.65	0.86	0.14
CLBN	0.31	0.29	0.44	0.33	0.39	−6.47
PrivBN	0.43	0.25	0.01	0.22	0.38	−5.58*
TVAE	**0.62**	0.38	0.10	**0.46**	0.43	−0.24
CTGAN	0.61	0.33	**0.66**	0.32	0.54	−0.07
Synthsonic	**0.62**	**0.39**	0.17	0.43	**0.65**	**0.02**

For real datasets, Synthsonic outperforms all others in three of the six datasets, and is the runner-up or tied in two others. As previously discussed in Section 5, with standard settings Synthsonic underperforms on the highly-imbalanced `credit` dataset.

A.2 Computational efficiency by dataset

Tables 7 to 9 show the corresponding breakdowns of run times for fitting and sampling, following the same structure as the tables above in Section A.1.

Note that for PrivBN the fitting and sampling stages cannot be separated so run times for this model are included under the 'fit' column alone.

In these tables, sub-second measures are denoted as "$<$ 1". The difference between any sub-second timings is relatively insignificant considering fit times, in particular on real datasets 9. We judged it to be impossible to easily provide meaningful sub-second measurements given background (e.g. operating system-related) loads which can affect running times at the sub-second level and which are outside of our control.

Table 7: Run times for fitting and sampling, on the Gaussian Mixture artificial datasets (seconds).

	grid		gridr		ring	
Method	fit	sample	fit	sample	fit	sample
CLBN	< 1	< 1	< 1	< 1	< 1	< 1
PrivBN	< 1		< 1		< 1	
TVAE	116	< 1	111	< 1	147	< 1
CTGAN	298	< 1	311	< 1	315	< 1
Synthsonic	3	< 1	4	< 1	3	< 1

Table 8: Run times for fitting and sampling, on Bayesian Network datasets (seconds).

	asia		alarm		child		insurance	
Method	fit	sample	fit	sample	fit	sample	fit	sample
CLBN	< 1	2	5	9	3	5	4	6
PrivBN	1		69		19		16	
TVAE	113	< 1	138	< 1	142	< 1	160	< 1
CTGAN	370	< 1	709	1	530	< 1	658	6
Synthsonic	2	< 1	12	1	6	< 1	8	< 1

Table 9: Run times for fitting and sampling, on the real datasets (seconds). Times are based on cpu unless otherwise stated.

	adult		census		credit		covtype		intrusion		news	
Method	fit	sample	fit	sample	fit	sample	fit	sample	fit	sample	fit	sample
CLBN	3	8	121	202	104	170	378	589	116	376	32	48
PrivBN	12		106		1574		63849		14954		19870	
TVAE	491	< 1	15779	4	23823	7	33787	7	27254	7	1933	1
with gpu	161	< 1	2813	4	4906	3	8509	10	7194	5	912	< 1
CTGAN	1933	1	49297	15	69156	16	67128	38	79798	21	5462	2
with gpu	400	1	6896	11	8130	9	14455	22	14421	17	1267	2
Synthsonic	17	1	111	12	333	15	382	44	365	29	79	2

The pattern in computational cost results is clear in the most expensive step of the modelling, which is the fitting (sampling is comparably inexpensive): Synthsonic is nearly as fast as the Bayesian Network models, and at least one —or in some instances, two— orders of magnitude faster than the VAE or GAN models.

B PARAMETERS

The parameters for w/o PCA, w/o clf and w/o calibration are booleans turning off the respective functionality. w/o tuning uses the default parameter settings. The full model's parameters are presented below.

For the synthetic datasets the parameters are chosen with respect to the validation set, however for the real datasets no validation set is provided, hence we only change parameters if issues are encountered with the default parameters (in case of imbalanced datasets we reduce the `test_size parameter` and if continuous variables are not modelled expressively enough we increase the `n_uniform_bins` parameter).

Here we provide a selection of the relevant altered configurations. The full (default) parameters can be found in the code[6].

[6]https://doi.org/10.5281/zenodo.6143990

```
'default': {
    'pdf_args': {
        # The train/test split used for calibrating the classifier
        'test_size': 0.35,
        # The number of bins to use to discretize continuous variables
        'n_uniform_bins': 30,
        # TAN structure estimation
        'estimator_type': "tan",
        # Use the normalized mutual information as edge weights
        'edge_weights_fn': "normalized_mutual_info",
        # Automatically infer the class node
        'class_node': None,
    },
}
```

For tuning, we consider the following options per parameter. In future work we plan to set these parameters heuristically (as is done for the structure learning parameters). The `n_uniform_bins` parameter can be set to any of the options of `numpy.histogram_bin_edges` or to 'knuth' for `astropy.stats.knuth_bin_width` to determine bin size based for each dimension individually. We have tested the `n_uniform_bins` parameter with multiple values and have found that the set $[50, 40, 30]$ works well in practice for most datasets.

```
# The test size can be reduced to prevent issues with imbalanced datasets
'test_size': [0.25, 0.35],

# Uniform binning parameter. Controls the granularity needed of
#   the discretized representation of continuous variables.
#   A lower value results in a more efficient Bayesian Network.
'n_uniform_bins': [50, 40, 30],
```

Parameters used for the full model in the ablation study. Note that for six of the datasets the default parameters were used.

```
# ========================================
# Gaussian Sim.
# ========================================
'ring': {
    'pdf_args': {
        'test_size': 0.25,
        'n_uniform_bins': 50,
    },
},
'grid': {
    'pdf_args': {
        'n_uniform_bins': 50,
    },
},
'gridr': {
    'pdf_args': {
        'n_uniform_bins': 50,
    },
},
# ========================================
# Bayesian Network Sim.
# ========================================
'asia': {
    'pdf_args': {}
},
```

```
'child': {
    'pdf_args': {}
},
'insurance': {
    'pdf_args': {}
},
'alarm': {
    'pdf_args': {}
},
# ========================================
# Real
# ========================================
'census': {
    'pdf_args': {},
},
'credit': {
    'pdf_args': {},
},
'adult': {
    'pdf_args': {
        'test_size': 0.25,
    },
},
'intrusion': {
    'pdf_args': {},
},
'covtype': {
    'pdf_args': {
        'n_uniform_bins': 40,
        'test_size': 0.25,
    },
},
'news': {
    'pdf_args': {
        # Classifier (for news dataset the classifier is turned off
        #   based on the ablation analysis results - but competitive
        #   performance still obtained even with the default
        #   configuration which includes the classifier.)
        'clf': None,
    },
}
```

C FEATURE SELECTION FOR REFINED MODELLING

Specific features can be selected for refined modeling of higher-order dependencies. These are modeled using a discriminative learner, as described in Section 3.3. Feature selection is beneficial when there are hundreds of features, and not all need to be modeled by the learner.

Our feature selection process is based on the ϕ_K correlation constant (Baak et al., 2020). ϕ_K supports categorical and numerical features and captures non-linear dependencies. As input it takes the contingency table of a feature pair. From this the expected frequency table is derived, which assumes no dependency between the features. A χ^2 comparison is performed between these tables, from which the ϕ_K coefficient is derived, in the range $[0, 1]$. A small χ^2 value results in a small value for ϕ_K, a large discrepancy gives a value close to one.

We make a slight alteration to the inputs of ϕ_K. Observed contingency tables are formed from the discretized input data X_{bn}, as usual. A large, synthetic dataset is generated from the Bayesian network model, see Sec. 3.6

for details on sampling. This sample is used to form the expected contingency tables. ϕ_K values are calculated for all feature pairs and ordered in descending value. Large values are assigned to pairs with a large discrepancy between the input data and the Bayesian network. For the discriminative learner, the top-n features can be selected with the largest ϕ_K values, or alternatively passing a configurable threshold, although by default all features are used for further modeling.